THE JANUARY 1983
VOL. 62, NO. 1, PART 2
BELL SYSTEM
TECHNICAL JOURNAL

COMPUTING SCIENCE AND SYSTEMS

THE 3B20D PROCESSOR & DMERT OPERATING SYSTEM

J. O. Becker, Guest Editor

(Contents continued on back cover)

THE BELL SYSTEM TECHNICAL JOURNAL (ISSN0005-8580) is published by the American Telephone and Telegraph Company, 195 Broadway, N.Y., N.Y. 10007, C. L. Brown, Chairman and Chief Executive Officer, W. M. Ellinghaus, President; V. A. Dwyer, Vice President and Treasurer; T. O. Davis, Secretary.

The Journal is published in three parts. Part 1, general subjects, is published ten times each year. Part 2, Computing Science and Systems, and Part 3, single-subject issues, are published with Part 1 as the papers become available.

The subscription price includes all three parts. Subscriptions: United States—1 year $35; 2 years $63; 3 years $84; foreign—1 year $45; 2 years $73; 3 years $94. Subscriptions to Part 2 only are $10 ($11 Foreign). Single copies of the Journal are available at $5 ($6 foreign). Payment for foreign subscriptions or single copies must be made in United States funds, or by check drawn on a United States bank and made payable to The Bell System Technical Journal and sent to Bell Laboratories, Circulation Dept., Room 1E-335, 101 J. F. Kennedy Parkway, Short Hills, N. J. 07078.

Comments on the technical content of any article or brief are welcome. These and other editorial inquiries should be addressed to the Editor, The Bell System Technical Journal, Bell Laboratories, Room 1J-319, 101 J. F. Kennedy Parkway, Short Hills, N. J. 07078. Comments and inquiries, whether or not published, shall not be regarded as confidential or otherwise restricted in use and will become the property of the American Telephone and Telegraph Company. Comments selected for publication may be edited for brevity, subject to author approval.

Printed in U.S.A. Second-class postage paid at Short Hills, N.J. 07078 and additional mailing offices. Postmaster: Send address changes to The Bell System Technical Journal, 101 J. F. Kennedy Parkway, Short Hills, N. J. 07078.

THE BELL SYSTEM
TECHNICAL JOURNAL

DEVOTED TO THE SCIENTIFIC AND ENGINEERING
ASPECTS OF COMPUTING

Volume 62 January 1983 Number 1, Part 2

The 3B20D Processor & DMERT Operating System:

Prologue

By J. M. SCANLON

(Manuscript received June 23, 1982)

The 3B20 Duplex Processor (3B20D) and its DMERT operating system represents a new and very important building block for the Bell System's stored program network. The first system to cut over was the network control point application in Kansas City, Missouri, on September 3, 1981. By year end 1982, the common 3B20D/DMERT processor system will have been in service supporting six different telecommunication applications across 65 different sites in 17 of the 20 Bell Operating Companies, including the Long Lines Division of AT&T. By year-end 1983, the number of different applications is expected to grow to ten and the number of sites to 300, covering all Bell Operating Companies. This very rapid buildup from first introduction to substantial deployment throughout the Bell System establishes the 3B20D/DMERT processor as a key element for the continued evolution of the Bell System's stored program network.[1]

The 3B20D Processor had its origins in exploratory work, begun in 1976, to establish a successor to the 3A processor[2] deployed with No. 2B ESS and No. 3 ESS.[3] At that time, a specialized replacement processor for 3A was envisioned, which is the origin of the designation "3B." At about the same time, research in operating systems charac-

terized by the *UNIX** operating system[4] and MERT[5], and portable higher-level languages characterized by C,[6] had matured to the point that the use of these techniques in real-time telecommunications applications became feasible and highly desirable. In addition, it was becoming clear that processor capabilities to support the future needs of the stored program network would have to be significantly enhanced beyond the capabilities of then-deployed ESS processors. These enhancements were required to support what was foreseen as the broadening role of the ESS processor from primarily a highly reliable call-handling and switch-maintenance controller to a sophisticated data base manager and terminal/data link controller as well. The use of a common processor and operating system throughout the telecommunications network also was seen as providing significant economic benefits occurring both from manufacturing and operational standardization, and most importantly, from achieving a high degree of software standardization. By early 1978, these factors had crystallized into five major development goals to:

(*i*) Provide the high degree of reliability and fault tolerance traditionally expected of ESS processors.

(*ii*) Provide efficient support of the high-level language, C, and a new real-time operating system, DMERT.

(*iii*) Provide the ability to directly execute programs written for earlier ESS processors, permitting these ESSs to follow a low-cost migration path to new processor and software technology.

(*iv*) Provide hardware and software architectural features to efficiently support large real-time data base and extensive data link capabilities.

(*v*) Provide a highly efficient software-development system based on the *UNIX* operating system as an integrated capability of the processor.

These goals were met successfully by a development program that designed the processor, 3B20D, and the operating system, DMERT, in parallel. The design approach used to bring the processor and operating system into fruition are discussed in the papers included in this issue.

The issue begins with a paper that discusses the first four applications of the 3B20D/DMERT system, followed by an overview and a discussion of the architecture of the complete system. There are then 15 other papers, grouped in sequence along the major topics of hardware (5 papers), software (4), maintenance and craft interface (5), and, finally, system integration and test (1). While omitting design-level details, this collection of papers provides a comprehensive overview of the 3B20D Processor and DMERT Operating System.

---

* Trademark of Bell Laboratories.

It is impossible to adequately acknowledge the contributions of everyone involved in a project of the magnitude of 3B20D/DMERT—people from many organizations of Bell Laboratories, Western Electric, AT&T, Long Lines, and the operating telephone companies, all participated in important ways. The authors of this volume would like to express their gratitude to all of these people for the unity of purpose and free communication that overcame the complex organizational interfaces and technical problems, and permitted successful project completion.

REFERENCES

1. *Stored Program Controlled Network,* B.S.T.J., *61,* No. 7 (September 1982).
2. W. N. Toy, "Fault Tolerant Design of ESS Processors," Proc. IEEE, *66* (October 1978), pp. 1126–45.
3. E. A. Irland and U. K. Stagg, "New Developments in Suburban and Rural ESS (No. 2 and No. 3 ESS)," 1974 ISS Symposium Record, pp 512/1–6.
4. D. M. Ritchie and K. Thompson, "The *UNIX* Time-Sharing System," B.S.T.J., *57* (July–August 1978), pp. 1905–29.
5. H. Lycklama and D. L. Bayer, "The MERT Operating System," B.S.T.J., *57* (July–August 1978), pp. 2049–86.
6. D. M. Ritchie, S. C. Johnson, M. E. Lesk, and B. W. Kernighan, "The C Programming Language," B.S.T.J., *57* (July–August 1978), pp. 1991–2019.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

The 3B20D Processor & DMERT As a Base for Telecommunications Applications

By R. W. MITZE, H. L. BOSCO, N. X. DeLESSIO, R. J. FRANK, N. A. MARTELLOTTO, W. C. SCHWARTZ, and R. M. WOLFE

(Manuscript received March 18, 1982)

The 3B20D Processor and the Duplex Multienvironment Real-Time (DMERT) Operating System provide a versatile processing base to fulfill the varied processing needs of telecommunications systems. To illustrate the versatility of the 3B20D/DMERT system, this article describes the structuring of four diverse telecommunication system applications.

I. INTRODUCTION

The 3B20D Processor and the Duplex Multienvironment Real-Time (DMERT) operating system were designed to provide a base for a wide variety of high-availability real-time and time-sharing applications. The wide applicability of the 3B20D/DMERT combination is due to the flexible hardware configuration of the processor coupled with the multilevel structure of the DMERT operating system. The versatile, intelligent input/output processor and the high-speed dedicated dual-serial channels provide for varying peripheral configuration interconnection needs.[1, 2] The microcode support for multiple resident instruction sets and the operating system support for application software at several different operating system functionality levels provide the diverse operating environment needs of teleprocessing system applications as well as data processing oriented applications.[3]

The combined 3B20D Processor and DMERT operating system can be viewed as a hierarchy of capability levels. This is depicted in Fig. 1. The lowest capability level is the 3B20D Processor hardware and

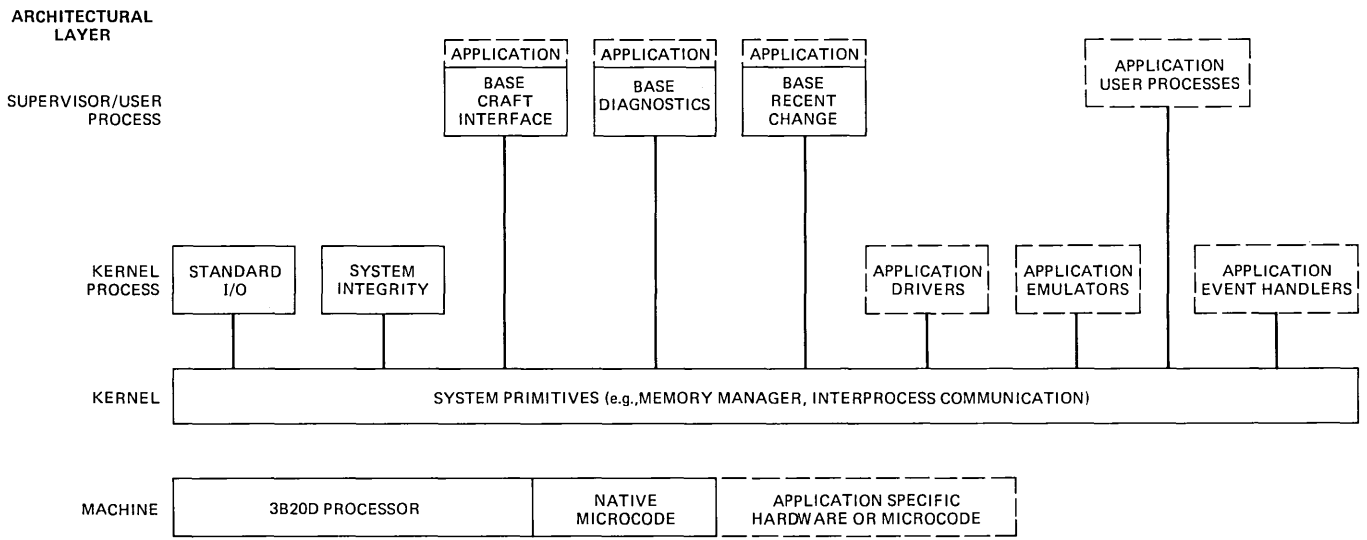


Fig. 1—3B20D/DMERT structure.

microcode. This level provides the machine instruction primitives. The next level of capability is the DMERT operating system kernel. This level provides the basic operating system primitive functions such as memory management and interprocess communication and scheduling. The next level of capability is the kernel process level. This level provides the ability to directly interface with the hardware for maximum real-time efficiency. The highest level of capability is the supervisor/user process level. This level provides full user-oriented input/output capability combined with the powerful and convenient development environment available to *UNIX** operating system users.

The combination of a multilevel operating system and support of multiple resident instruction sets gives 3B20D/DMERT exceptional adaptability. We will describe how these two capabilities can be used to provide different kinds of services to applications and illustrate their use in four systems applications.

1.1 Multilevel operating system

From an application viewpoint the multilevel capability of DMERT allows an application to tune its software according to the most appropriate trade-off of real-time control versus ease of programming and maintainability. The DMERT kernel process interface allows applications to attach directly to interrupts in order to support application-specific peripherals or other custom hardware. It also allows applications to write real-time event software handlers that run at any desired hardware priority level in order to provide real-time response to events. In effect, the application shares the hardware-scheduling decisions with the DMERT operating system.

If direct hardware scheduling or control is not needed, DMERT offers the supervisor/user level of the *UNIX* operating system for time-sharing applications. This level provides a large amount of program protection and development support to make writing and maintaining user-level processes easier. At the user level all access to machine resources is controlled by DMERT so that unauthorized accesses are prevented and the user is shielded from most of the specific hardware details. In most applications, the bulk of the software is at the user level and only a small percentage needs to be written at the kernel process level.

The user-level environment of the *UNIX* operating system supplied with DMERT need not be the only user-level environment. Applications also have the ability to construct additional environments to provide specific application services for programmers. This can be especially useful as a means of ensuring uniform treatment of appli-

* Trademark of Bell Laboratories.

cation-specific resources and as a way to provide a standard development base in the situation where multiple systems may be used by a single application.

1.2 Multiple resident instruction sets

The 3B20D Processor is a microcoded machine that provides sufficient microstore and control to support multiple instruction sets resident on the processor at the same time. The DMERT operating system running in 3B native code is designed to coexist with emulated software running on a different instruction set. This allows the 3B20D Processor to be used by certain applications to preserve any software investment involved in previously released systems. Since the processor dynamically shifts between the emulated and native environment by executing a single instruction, new software can be written in either emulated or native mode as needed.

II. NO. 5 ESS

No. 5 ESS[4] is a Bell System-developed local digital switching system designed as a world-class product for both domestic and international markets and employing the latest technology. The No. 5 ESS system can be configured to serve rural offices with as few as 1000 customer lines, as well as metropolitan offices with up to 100,000 lines. Early applications of No. 5 ESS are as replacements of electromechanical switching systems. The first such system was placed in commercial service in March, 1982. This represents an important first step in the evolution of the Bell System's nationwide digital network.

In the overall No. 5 ESS architecture, the 3B20D Processor is used as the central processor complex and is responsible for a variety of functions including maintenance control, administration, human interface, system integrity, and certain centralized call-processing functions such as routing and control of global data. Some of these functions are performed by the No. 5 ESS application software, some are performed by DMERT, and some are joint responsibilities requiring close coordination between the No. 5 ESS application and DMERT. In this capacity, the 3B20D Processor is playing several different roles, and the multilevel hierarchy in DMERT allows the different functions to be placed at the most appropriate level. For each function, the tradeoffs involved were evaluated and the most appropriate capability level for No. 5 ESS processes was selected. For example, many functions such as administration, recent change, and basic diagnostics are not real-time sensitive. Therefore, the performance of the kernel process level was not required and the easier development environment of the user-process level was considered to be the deciding criterion. On the other hand, the call-processing function required the best possible real-

time performance even at the cost of lesser development convenience, so the kernel process level was chosen.

The No. 5 ESS application added an application-specific environment having features such as task management, message handling, and timing specifically oriented to No. 5 ESS call-processing needs. This same environment is provided on other processing components within the distributed No. 5 ESS architecture. This resulted in a uniform interface for No. 5 ESS developers with subsequent advantages in training, portability, and ease of development. Thus, the No. 5 ESS application demonstrates how the multiple-level architecture of DMERT allows applications the capability of making their own choices and trade-offs between performance versus flexibility and ease of development. Also, this approach shows how the kernel process interface allows an application to implement an environment most suitable for its particular needs.

III. TRAFFIC SERVICE POSITION SYSTEM

Since its introduction in 1969, the Traffic Service Position System No. 1[5, 6] (TSPS No. 1) has been deployed rapidly throughout the Bell System's nationwide telecommunications network to the point where there are more than 150 systems in the continental United States. Over 95 percent of Bell System customers and a large number of customers of other telephone companies are served by TSPS No. 1. The continuing growth of operator services traffic, plus the continuing addition of new features, have steadily reduced the remaining real-time capacity and available memory in the TSPS No. 1 Processor. Additionally, plans to utilize TSPS as an Action Point in the emerging stored program controlled network dictated the need for new processor peripherals, such as a mass memory disk that cannot be provided by the current TSPS No. 1 processor.

To meet these needs, a major evolution from TSPS No. 1 to a new system, called TSPS No. 1B, was required. The goals of this evolution were to provide substantially increased call capacity, memory, and processor peripheral capability.

The size of the investment in existing TSPS No. 1 software programs and peripheral hardware is very large. To preserve as much of this investment as possible, the existing software and peripheral hardware were retained in the transition to TSPS No. 1B. At the same time, the capability of adding new features using high-level language software was provided.

The 3B20D Processor and DMERT provide the basis for meeting the goals of TSPS No. 1B. The 3B20D Processor replaces the existing TSPS No. 1 processor. The existing TSPS No. 1 periphery is retained and, through emulation, the existing TSPS software is preserved. The

retention of existing TSPS No. 1 periphery is achieved through the use of a Peripheral System Interface (PSI) circuit designed to interface the TSPS buses with the 3B20D Processor. The software preservation is accomplished by defining (through microcode) one of the multiple 3B20D processor instruction sets to be that of the existing processor, thus emulating that processor and allowing existing TSPS software to be transported to the 3B20D Processor almost intact. The ability exists to switch between the emulated instruction set and the 3B20D native instruction set within a single process and with a single instruction. Thus, new software can be added into either environment, as appropriate. Both emulated and native-mode software are run under the DMERT operating system, allowing operating system services to be available to both forms of software. The emulated existing TSPS No. 1 assembly language code is structured as a single kernel process executing under DMERT. As explained in the No. 5 ESS discussion, this permits efficient control of real-time resources where required. Other administrative and diagnostic functions that are not real-time sensitive are implemented as user-level processes. The DMERT operating system provides processor maintenance and administration. Thus, both the multilevel operating system and multiple-resident instruction set capabilities of 3B20D/DMERT are essential elements of the TSPS No. 1B design.

The first TSPS No. 1B started serving telephone customers in Fresno, California in November, 1981. As of September 1982, about 20 TSPS No. 1B's are in service and, by year-end, it is planned to have a total of about 35 in service. Performance data from all TSPS No. 1B sites indicate that all design objectives have been achieved.

IV. NETWORK CONTROL POINT

The Network Control Point[7] (NCP) is a new Bell System development that adds an on-line real-time data base to the Common Channel Interoffice Signaling (CCIS) network. Using the alternate-routing and direct-signaling features of the CCIS network, the NCP provides a high-reliability, distributed data-base capability. The first applications of the NCP are to support 800 Service and Expanded 800 Service and to provide billing validation for the Automated Calling Card Service.

Since the NCP was designed to use the 3B20D Processor and the DMERT operating system, the software is written entirely in the C programming language. Those portions of the NCP software that are real-time critical are implemented at the kernel process level, while those portions with less stringent real-time requirements are implemented at user level. Thus, all features of the DMERT operating system are used to create an efficient, homogeneous system at a significant savings in project development cost.

The primary purpose of the NCP is to provide reliable, fast access to a data base. Hence, special emphasis is placed on the access to the moving head disks and to the communication via data links to the data base administration centers. A cache algorithm is implemented that allows the most frequently used entries to be queried without access to the disk. Multiple copies of the data base, beyond the two copies maintained by DMERT, are kept to protect against loss of the data base. Special spoolers are provided to aid in the communication between the NCP and the data base administration centers. The flexibility of DMERT permits these application-specific features to be easily implemented and integrated into the system.

Hardware for the NCP is composed of standard 3B20D Processor units except for one special peripheral controller board used to link the NCP to the signal transfer point of the CCIS network. The DMERT input/output driver module is modified to handle this board, and a diagnostic module is integrated with the standard DMERT diagnostics. The remainder of the NCP hardware consists of duplex processor systems equipped with 10 to 16 megabytes of memory each, four input/output processors, five moving head disk units, six to ten BX.25 data links, and a magnetic tape unit, making the typical NCP one of the larger 3B20D Processor systems in operation.

The NCP was the first user of the 3B20D Processor and the DMERT operating system to go into commercial service. The straightforward architecture of the NCP, both hardware and software, and the system test capability at the NCP development laboratory allowed sufficient operational confidence to be established so that four systems were placed into service on September 3, 1981. By November 1, 1981, 14 systems were in service, handling 15 million queries per business day across the United States from all calls prefixed by 800.

V. ATTACHED PROCESSOR SYSTEM FOR NO. 4 ESS AND NO. 1A ESS

The 1A Processor is the central processing unit of both the No. 4 Electronic Switching System (No. 4 ESS) and the No. 1A Electronic Switching System (No. 1A ESS).[8,9] The No. 4 ESS is designed to handle toll and tandem switching functions while the No. 1A ESS is designed to handle local, tandem, and toll switching functions; both systems have been in service for more than six years. Currently there are about 90 No. 4 ESS and 800 No. 1A ESS systems in service. The growth of telephone traffic and customer demand for new services on these systems established a need to increase processing capacity, implement new service features, and expand the memory spectrum. A new system architecture involving the attachment of an additional processor to the 1A Processor configuration was designed to meet these needs. The 3B20D Processor with the DMERT operating system

was chosen as the appropriate processor because of its low cost, reliability, processor and memory resource capacity, and the availability of a high-level language in the software development environment.

The No. 4 ESS and No. 1A ESS are designed to use the powerful 1A Processor for central control and memory. There are three types of memory: program store containing the fixed set of software instructions for operational functions; call store containing translation data describing office configuration and parameters; and file store on disk containing backup copies for both program and call store as well as less frequently exercised programs, such as diagnostic routines. The 1A Processor file store was the earliest resource to exhaust under the pressure of increased traffic load and new feature development. The Attached Processor System (APS), with the 3B20D Processor and DMERT operating system, has been designed as a replacement for the 1A Processor file store. In addition, APS also provides a mechanism to deload the 1A Processor of operational and administrative tasks which potentially limit its real-time performance at high traffic loads. It should be emphasized that the 1A and 3B20D Processor support different programming languages and new development software can be written in the more appropriate language.

The APS system includes hardware and software to connect the 3B20D Processor to the 1A Processor. The hardware includes an Attached Processor Interface (API) unit to interconnect the Direct Memory Access (DMA) channels of the two processors. The software includes an API driver consisting of 3B20D Processor program modules and corresponding 1A Processor program modules. Together these modules administer and maintain a 10 megabit/second fully duplicated interprocessor communication link. The 3B20D Processor modules are designed at the kernel process level to meet the stringent real-time requirements for processor intercommunication. To maintain integrity with the existing No. 4 ESS and No. 1A ESS software environment, the 1A Processor disk administration mechanisms were provided in the APS. The 1A Processor was also provided with full access to the 3B20D/DMERT File Manager and the entire 3B20D file system addressing spectrum.

The first No. 4 ESS with APS was scheduled for commercial service in 1982. The first No. 1A ESS with APS is scheduled for service in 1983. To take full advantage of the 3B20D Processor capabilities for future development, several new No. 4 ESS and No. 1A ESS features are planned to be implemented using DMERT and the high-level C language.

VI. SUMMARY

The 3B20D/DMERT system has achieved its objective of providing a cost and real-time effective base for a wide variety of telecommuni-

cations systems. The key concepts of multiple levels of functional support, emulation microcode support, and versatile input/output interfaces as combined in the 3B20D/DMERT system provide an adaptable base that can be tailored to many differing needs. In addition to the four applications described above, the 3B20D/DMERT system is the basis of a number of other telecommunication system designs currently under way in the Bell System. This widespread use of 3B20D/DMERT marks it as a processor/operating system combination of significance in the Bell System.

REFERENCES

1. W. N. Toy and L. E. Gallaher, "Overview and Architecture of the 3B20D Processor", B.S.T.J., this issue.
2. A. H. Budlong and F. W. Wendland, "3B20D Input/Output Systems," B.S.T.J., this issue.
3. M. E. Grzelakowski, J. H. Campbell, and M. R. Dubman, "DMERT Operating System," B.S.T.J., this issue.
4. W. B. Smith and F. T. Andrews, "No. 5 ESS Overview," Int. Switching Symp., 1981, Montreal, Canada.
5. N. X. DeLessio and R. E. Staehler, "The 3B Duplex Processor System and Its Application to TSPS, Part 2," Int. Switching Symp., 1981, Montreal, Canada.
6. R. E. Staehler, "Traffic Service Position System No. 1B—Overview and Objectives," B.S.T.J., *62*, No. 3 (March 1983).
7. "Common Channel Interoffice Signaling," B.S.T.J., *57*, No. 2 (February 1978).
8. "The 1A Processor," B.S.T.J., *56*, No. 2 (February 1977), pp. 119–327.
9. "No. 4 ESS: Prologue," B.S.T.J., *60*, No. 6, Part 2 (July–August 1981), pp. 1041–1048.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

Overview and Architecture of the 3B20D Processor

By W. N. TOY and L. E. GALLAHER

(Manuscript received March 8, 1982)

The 3B20D Processor is designed to meet a broad range of telecommunication applications. New features such as memory management are incorporated into its design to support a modern operating system. Hardware supports are provided to efficiently execute a high-level language. A major design objective is to meet the stringent reliability requirements of electronic switching systems. The proven technique of self-checking duplex operation forms the basic architecture structure of the 3B20D Processor.

I. INTRODUCTION

The 3B20D Processor is the first member of a family of processors designed for a broad range of Bell System applications. Its development is a natural outgrowth of the continued need for high availability and real-time control of Electronic Switching Systems (ESSs),[1-3] including existing as well as new telecommunication applications. With the rapid growth of integrated-circuit technology, the processor architecture is evolving to include as many features as possible to significantly reduce software development and maintenance costs.

The 3B20D Processor architecture takes advantage of the LSI technology to expand its functionality and yet maintain a high reliability standard. Some of the design goals are to:

(*i*) Achieve highest performance that is consistent with system cost, e.g., provide hardware facilities such as data cache, high-speed interrupt stack, address-translation cache, and microprogramming for critical functions that require too much time in software.

(*ii*) Reduce software complexity, e.g., provide a modern real-time

operating system to manage system resources, thereby creating a more useful and more reliable programming environment for the user.

(*iii*) Reduce programming effort, e.g., provide both an efficient high-level language, such as the C language,[4] and a comprehensive set of software development tools.

(*iv*) Provide a high level of reliability and fault tolerance, e.g., built-in error-detection and correction codes, recovery features, and fault diagnostics.

(*v*) Provide features for system integrity and security, e.g., memory management protection and privileged instructions.

These goals are considered from the viewpoints of both hardware and software architecture in order to realize the most cost effective system for a wide spectrum of applications. Much of the development effort has been directed to achieve these goals.

II. GENERAL DESCRIPTION

High availability is one of the major objectives in the design of the 3B20D Processor. The successful deployment and field operation of many ESS systems have demonstrated the simplicity and robustness of duplex configuration in meeting the ESS reliability requirements.[5] Hence, duplex configuration forms the basic structure for both the hardware and software architecture. Experience gained in the design and field operation of the No. 3A Processor provided valuable input for the 3B20D Processor design.[6]

The 3B20D Processor employs concurrent self-checking design. Extensive checking hardware is incorporated as an integral part of the processor. Faults occurring during normal operation are quickly discovered by detection hardware. This eliminates the need both to run the standby processor in the synchronous mode of operation and to run the fault-recognition program to identify the defective unit when a mismatch occurs. Self-checking implementation simplifies the maintenance program. Reconfiguration into a working system is immediate, without extensive diagnostic programs to determine which subsystem unit contains the fault. Furthermore, the self-checking design will permit more straightforward expansion from simplex to duplex, or multiple processor arrangements.

2.1 Duplex configuration[7]

Figure 1 shows the general block diagram of the 3B20D Processor. The Central Control (CC), the memory, and the I/O disk system are duplicated and grouped as a switchable entity although each CC can access each disk system. The quantity of equipment within the switchable block is small enough to meet the reliability requirement; therefore, the complexity of a recovery program to manage additional

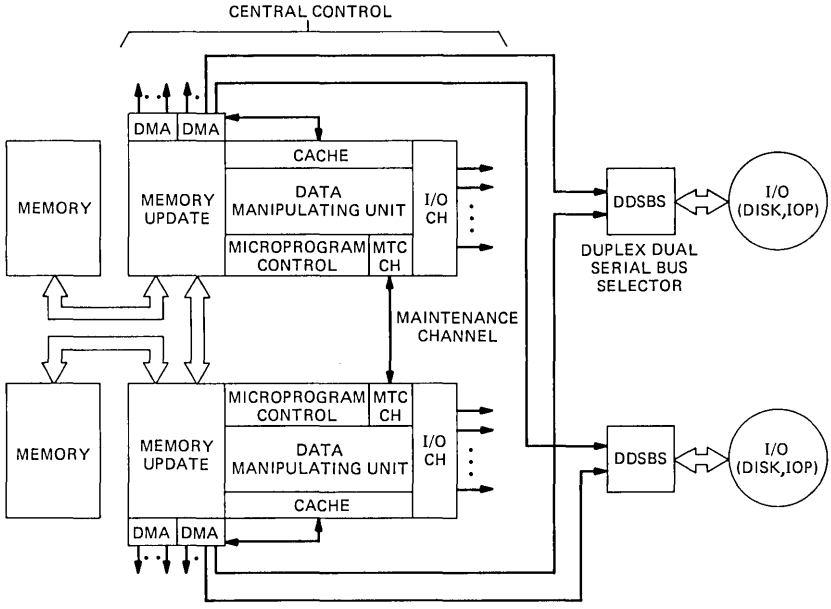


Fig. 1—General block diagram of the 3B20 Processor.

working states is avoided. Although each CC has direct access to both disk systems, this capability is used mainly to provide a valid data source for memory reload under trouble conditions. The processors are not run in the synchronous and match mode of operation as is done in early systems.[1-3] However, both stores (on-line and standby) are kept up-to-date by the memory update hardware concurrent with instruction execution. This is achieved by having the on-line memory-update circuit write into both memories simultaneously when memory data are written by the CC. Under trouble conditions, when the control is switched to the standby processor, its memory will contain up-to-date information without performing a complete transfer from one processor to another. The Direct Memory Access (DMA) circuits interface directly with the memory update circuit in order to have access to both memories. A DMA write also updates the standby memory. Communication between the DMA and the peripheral devices is accomplished by using a high-speed dual-serial channel (DSCH). The duplex dual-serial bus selector (DDSBS) allows both processors to access a single I/O device. For maintenance purposes, the duplex 3B central controls are interconnected via the Maintenance (MTC) channel. This high-speed serial path provides diagnostic access at the microcode level. It has the capability of transmitting a stream of microinstructions to exercise the processor from the on-line processor or from an external unit for diagnostic purposes.

2.2 Peripheral devices[8,9]

A broad range of general-purpose peripheral devices is provided for the 3B20D Processor system. High reliability and maintainability continues to be the design philosophy of the 3B20D peripheral system.

The critical components are duplicated and the software ensures that valid data sources are maintained. The DDSBSs permit controlled switching of a working standby device for a faulty on-line device when duplication of peripheral devices is needed. Some major peripheral devices developed for the 3B20D system are:

(*i*) Moving head disk system—The disk system provides a reliable and flexible mass storage medium for program and data. A backup copy of system programs and critical parameters can be reloaded quickly in the event of a duplex main-store failure. The disk system comprises the Disk File Controller (DFC) and the Moving Head Disk drive (MHD). The DFC interprets and executes commands from the processor to cause information transfer from and to the MHD. Each DFC occupies 1 of 128 channel slots and supports up to 16 MHD drives which are available in 80 and 300 megabyte sizes.

(*ii*) I/O processor (IOP)—The IOP provides the control for a wide range of data-link facilities and is the most flexible of the family of devices. An IOP supports up to 16 Peripheral Controllers (PCs) with each being a microprocessor-based controller programmed to handle a specific terminal or device. For example, one type of PC is the Line Controller (LC); each LC can support up to four independent lines (data links or terminals).

(*iii*) Magnetic tape system—The tape drive accepts the industry-standard (IBM compatible) 9-track tapes at a density of 1600 bits per inch. The tape controller is derived from the basic PC and occupies one of the 16 slots of the IOP.

(*iv*) Scanner/signal distributor (SC/SD)—This device is useful in monitoring and controlling power, equipment states, environment conditions, etc. The SC/SD circuit board provides 48 scan points and 32 signal-distributer points. It occupies one of the PC slots of the IOP. When an IOP is fully equipped with 16 SC/SD circuit packs, a total of 768 scan points and 512 signal-distributor points are provided.

III. SOFTWARE SUPPORT FEATURES[10]

The high cost of designing, updating, and maintaining software dominates the cost of producing computer systems. Considerable attention has been focused on providing various types of support, i.e., high-level language, operating system, and software test, in the development of 3B20D Processor. The combined software and hardware effort has yielded an integrated and cost-effective system.

3.1 High-level language support

The most common approach to increasing software productivity and reducing software maintenance cost is the extensive use of a high-level language suitable for the application. The design of the 3B20D Processor instruction set was based on the fact that C language programs would dominate the programming environment. C is a general-purpose programming language featuring economy of expression, modern control flow and data structures, and a rich set of operators. Many studies were directed to measure and determine the characteristics of a large, diverse sample of C programs. Based on the result of these studies, the instruction set was optimized to be space and time efficient for compiled C programs. Some features provided for the instruction set are concerned with:

(*i*) Symmetrical resources
(*ii*) Addressing modes
(*iii*) Address manipulation
(*iv*) Flexible data structure
(*v*) Stack instructions
(*vi*) Procedural instructions.

From the compiler's viewpoint, the most important attribute of a processor instruction set is regularity. It is the key feature needed to abstract the various processor resources for uniform treatment by the complier. The 3B20D instruction set includes a wide range of address modes—i.e., indexing, direct, indirect—covering various data structures. The treatment of the addressing modes, applied identically to all data types (bytes, half words, full words, and instructions) without exceptions, makes it possible to compile compact and efficient codes.

The subroutine is one of the most important concepts in software. The principal idea in modular, structured programming is the partitioning of large programs into many small, understandable procedures or subroutines. Efficient instructions have been provided to handle subroutine entry and exit in addition to stack manipulation.

3.2 Operating system support

Higher productivity in application programming is made possible by the high-level, simplified facilities provided by the operating system. The Duplex Multienvironment Real Time (DMERT) operating system for the 3B20D Processor is a general manager of processor, memory, input/output, and software processes. The functional description of DMERT will be described in more detail in the next section. This section describes the hardware that has been incorporated into the design to reduce the overhead of the operating system.

As previously indicated, a high-speed address translation cache

memory called the Address Translation Buffer (ATB) is provided to reduce the overhead associated with the address translation function.

Context switching is necessary upon interrupt. A memory stack is provided to facilitate the saving and restoring of the hardware context. In the 3B20D Processor, a local high-speed 8K-byte RAM is provided for this function. The addressing of the stack is part of the kernel virtual address space and has been assigned a fixed segment number and pages 0 to 3. Whenever the kernel virtual address falls into this range, the store operation is directed to the high-speed RAM; otherwise, the virtual address is translated by the ATB and pointed to the main memory. The combination of fixed mapping by special circuit and dynamic address translation by ATB allows the high-speed stack to be extended into the main memory when the use of the high-speed RAM is exceeded.

3.3 Software test support

The software test facility is an option provided at both the microprogram level and the macroprogram level. The Microlevel Test Set (MLTS) is attached to the microcontrol section of the central control. It has the capability of direct access to a support computer system for assembling and loading the writable microstore through the MLTS. The primary purpose of the MLTS used in the development of the 3B20D Processor is for initial debugging and troubleshooting the processor core hardware and, subsequently, the microprogram sequences. Features incorporated into the MLTS allow stepping, freezing, examining, and tracing the execution of a microprogram sequence.

The Utility Circuit (UC), on the other hand, provides a similar set of facilities, except at the macroprogram level for software debugging and troubleshooting. The UC and its associated software form an extensive Test Utility System (TUS) for software testing. A small number of matchers are incorporated into the UC for the tracing and monitoring of a variety of system conditions so that a programmer can observe and follow the execution of a program sequence. Much of the program debugging can take place in real time concurrent with program execution. The UC thus directly extracts and records information such as transfer trace from the internal data buses, thereby "capturing" the history of the machine while it is running at normal speed.

IV. DMERT OPERATING SYSTEM[11]

DMERT is a general-purpose operating system. It is structured as cooperating processes that provide different levels of virtual machines. Protection is built into the structure, preventing these virtual machines from destructively interfering with each other. For processes to cooperate in accomplishing their task, DMERT provides a rich set of

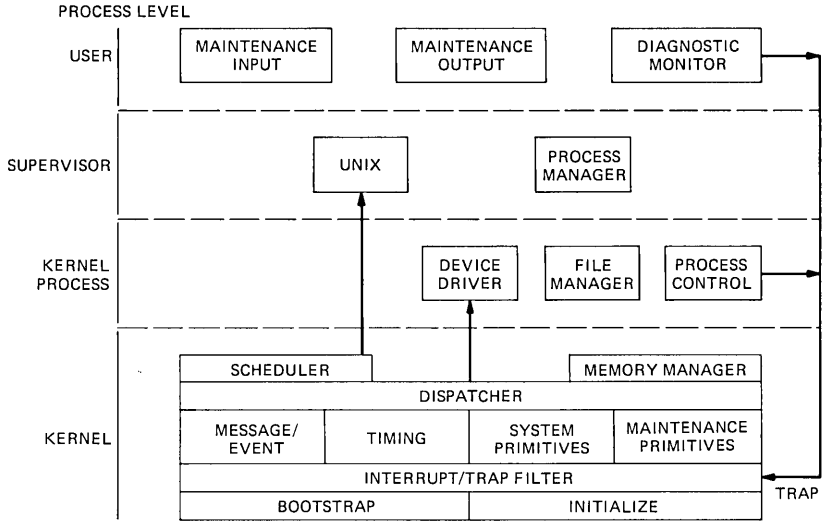


Fig. 2—The DMERT structure.

interprocess communication and synchronization mechanisms, including messages, events, process ports, interprocess traps, and shared memory. By means of these communication primitives, system services can be provided easily to a requesting process.

4.1 Multilayered virtual machines

DMERT provides four levels of virtual machines. They are the kernel, kernel process, supervisor, and user (Fig. 2). Successive levels put additional restrictions on access rights of system resources. This helps free programmers from the details of the physical machine. The higher level may take advantage of services provided by the lower levels.

(*i*) Kernel—The kernel provides the most primitive virtual machine. Programs at this low level are closest to the hardware. They directly control the system hardware and do not have access to other system functions. The kernel services are primitive, yet are efficient in their execution. The user cannot introduce code at this level.

(*ii*) Kernel process—Kernel processes are also strongly hardware related and are structured to provide time-critical processing in a real-time environment. DMERT uses this level for the file manager and device drivers. In addition, there are several special processes that provide scheduling, memory management, and other services. Users may add a kernel process that communicates efficiently with other kernel processes making efficient use of their services.

(*iii*) Supervisor—This third level is comprised of programs that are

generally hardware independent. These processes can use all the services provided by the kernel and its processes. The process manager is implemented in this level. So is a *UNIX** operating system supervisor, which provides time-shared usage of processor hardware through services provided by a scheduler. In general, supervisor segments are not locked in memory and can be moved out onto the disk. Consequently, supervisor processes take a much longer time to dispatch than do kernel processes.

(*iv*) User—All applications programs for which time is not a critical factor are written at the user level. The user process is linked directly with a *UNIX* operating system supervisor in which DMERT treats the supervisor/user pair as a single process. These user programs only see the software environment and services of the supervisor and are well protected from other user processes. Since user-level code must interface its own supervisor to use the lower-level primitives, additional overhead is added for user-level processes.

4.2 Multiple environment support

An application may add code at the level of the kernel process, the supervisor process, or the user. The multilevel structure makes DMERT a flexible system for real-time use. In general, the higher the level, the more services that are available to an application; the lower the level, the more efficient is the program execution. This level structure of virtual machines permits DMERT to manage real-time applications, while at the same time providing the flexibility of a time-sharing system for background tasks. Figure 3 shows how telephone switching software can be allocated to these different levels. By means of this hierarchical execution-level structure, application programs can customize their control and distribution of real time.

The portion of real time that is not utilized by the kernel or kernel processes is time shared among supervisor and user processes. Deferrable jobs such as traffic reports, recent changes, and diagnostics, as shown in Fig. 3, are implemented at the highest user level. The DMERT architecture thus simultaneously supports both a real-time and a time-sharing environment to fully utilize physical resources in the most efficient manner.

V. MAINTENANCE FEATURES

Increased support in this area is most appropriate to facilitate a more reliable and more maintainable system, thereby reducing the maintenance cost. For real-time applications, as in the Electronic Switching Systems (ESS), high availability and uninterrupted opera-

* Trademark of Bell Laboratories.

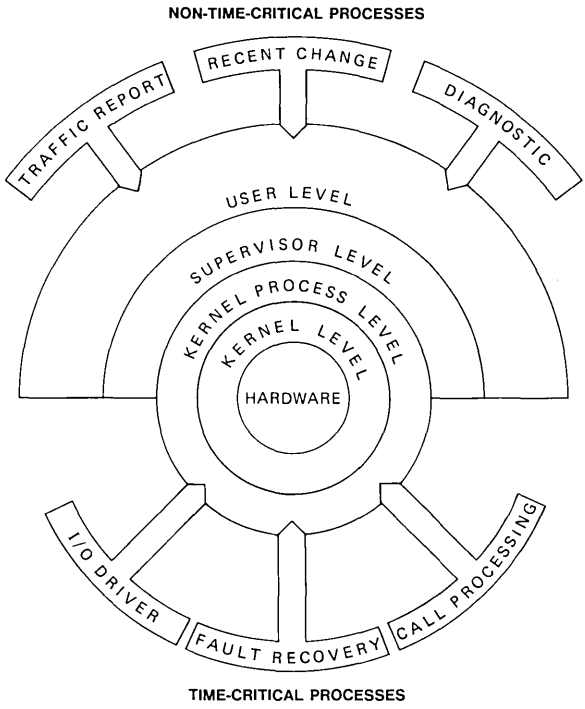


Fig. 3—Example of DMERT multienvironment structure.

tion is essential. This requires the system to function correctly even when a fault is present and maintenance is being performed. The 3B20D is designed to meet the ESS standard so that the expected amount of accumulated processor downtime does not exceed an average of 2 minutes per system per year.[5] Software and hardware are designed to function jointly to ensure that reliability objectives are met. Software features include such components as fault recovery programs, audits, and diagnostics.[12] Hardware features include redundant processors, error-detection circuits, maintenance-access controls, and diagnostic microcode. These components contribute to the effective maintenance design.

VI. SUMMARY

The 3B20D Processor is a high-availability system capable of supporting a broad spectrum of applications. A comprehensive set of software tools and facilities is provided to improve programming productivity and also to reduce the cost of software development and maintenance. The hardware architecture is designed to efficiently support high-level languages, particularly the C language.

A wide range of general-purpose peripherals have been provided with the 3B20D Processor. Some of these are the moving head-disk system, magnetic tape system, high-speed printer, scanner and signal distributor, and data terminals.

DMERT is the duplex multienvironment operating system for the 3B20D Processor. It was designed concurrently with the hardware to meet the high-availability demands of real-time switching and telecommunication systems. DMERT provides a set of procedures that enables users to efficiently share the 3B20D Processor and the physical resources such as processor time, storage space, and peripheral devices. The multiple environment permits time-critical code to coexist with time-shared software as background tasks.

An important provision in the 3B20D Processor is a complete set of maintenance facilities, from error detection through fault recovery and diagnostics. Approximately 30 percent of the internal central control logic is devoted to self-checking. This allows concurrent error detection and immediate recovery. The combined hardware and software features give an integrated package of maintenance facilities to meet the high ESS reliability requirements.

VII. ACKNOWLEDGMENTS

The design of the 3B20D Processor was accomplished through the combined efforts of many designers in Bell Laboratories and Western Electric. The authors wish to acknowledge the leadership of R. J. Watters who helped direct the early development of the 3B20D Processor system, and the design efforts of C. W. Hoffner, D. E. Gearhart, C. M. Narayanan, D. J. Matter, and D. J. McGraw.

REFERENCES

1. "No. 1 ESS," B.S.T.J., *43*, No. 5 (September 1964).
2. "No. 2 ESS," B.S.T.J., *48*, No. 8 (October 1969).
3. "No. 1A Processor," B.S.T.J., *56*, No. 2 (February 1977).
4. B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Englewood Cliffs, NJ: Prentice-Hall, 1978.
5. W. N. Toy, "Fault Tolerant Design of ESS Processors," *Proc. IEEE*, *66* (October 1978), pp. 1126–45.
6. T. F. Storey, "Design of a Microprogram Control for a Processor in an Electronic-Switching System," B.S.T.J., *55*, (February 1976), pp. 183–232.
7. M. W. Rolund, J. T. Beckett, and D. A. Harms, "3B20D Central Processing Unit," B.S.T.J., this issue.
8. A. H. Budlong and F. W. Wendland, "3B20D Input/Output System," B.S.T.J., this issue.
9. R. E. Haglund and L. D. Peterson, "3B20D File Memory System," B.S.T.J., this issue.
10. B. R. Rowland and R. J. Welsch, "Software Development System," B.S.T.J., this issue.
11. M. E. Grzelakowski, J. H. Campbell, and M. R. Dubman, "DMERT Operating System," B.S.T.J., this issue.
12. R. C. Hansen, R. W. Peterson, and N. O. Whittington, "Fault Detection and Recovery," B.S.T.J., this issue.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

3B20D Central Processing Unit

By M. W. ROLUND, J. T. BECKETT, and D. A. HARMS

(Manuscript received March 18, 1982)

The 3B20D Processor has been developed to meet the need for very reliable, real-time control of a variety of Bell System applications. To achieve its high-reliability goals, most of the major subsystems within the processor are duplicated, including the Central Processing Unit (CPU). The CPU uses a 32-bit architecture throughout, including the memory and input/output buses. Extensive self-checking logic is employed. The 3B20D CPU is microprogrammed to select dynamically up to four instruction sets. The microstore uses a 64-bit word with up to 16K words of high-speed bipolar PROM or RAM available. This rich emulation capability makes the 3B20D Processor ideal for emulating existing instruction sets and porting existing software. Peripheral units are connected to the CPU via the Direct Memory Access unit (DMA). The DMA controllers provide direct memory transfers between the main store and peripheral devices, reducing the load placed on the central control to process input/output requests.

I. CENTRAL CONTROL STRUCTURE

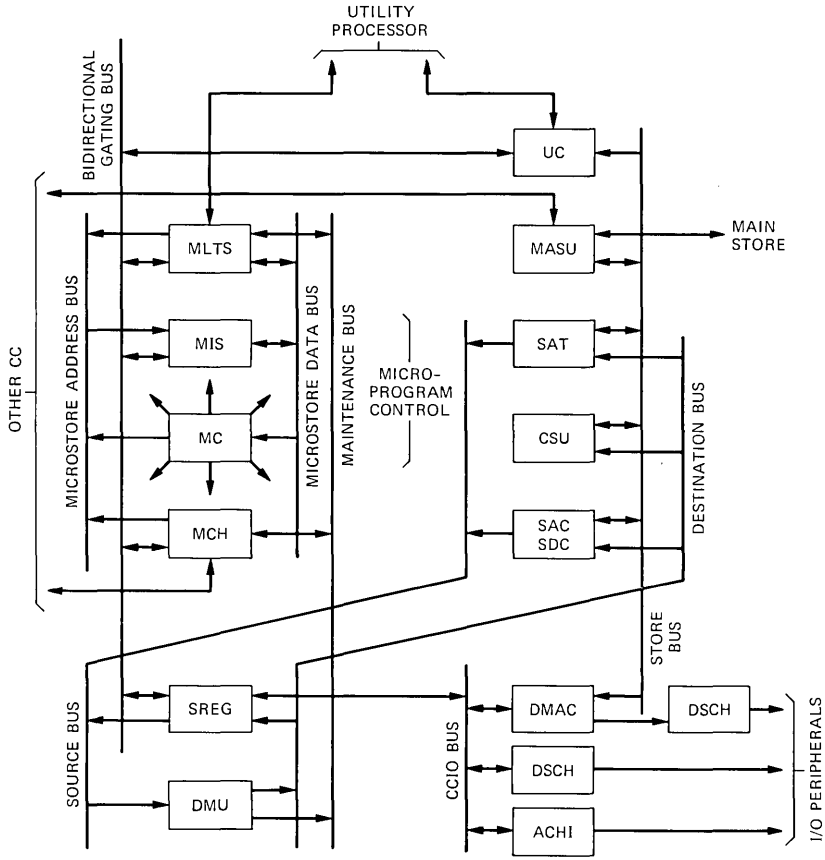
The 3B20D Processor performs all the functions normally associated with a CPU and several unique functions as well. The unique functions include the features necessary for reliable duplex operation, efficient emulation of other machines, and communication with a flexible and intelligent periphery. The processor employs an extensive microprogram capability to minimize the amount of hardwired decoding and to simplify the writing of microcode. There is substantial flexibility in the choice of instruction formats that can be efficiently interpreted.

The CPU is a 32-bit machine with a 24-bit address. Most of the data paths in the Central Control (CC) are 32 bits wide (plus 4 parity-check

bits). A fully self-checking philosophy is applied within each CPU without dependence upon matching between the duplex CPU's. The CC design is based on the register type of architecture, with multiple buses to allow concurrent data transfers. Separate I/O and store buses provide the capability of concurrent store and I/O operations.

1.1 Major subsystems

A detailed block diagram of the CC structure is shown in Fig. 1. The



ACHI – APPLICATION CHANNEL INTERFACE
CC – CENTRAL CONTROL
CCIO – CENTRAL CONTROL INPUT/OUTPUT
CSU – CACHE STORE UNIT
DMAC – DIRECT MEMORY ACCESS CONTROLLER
DMU – DATA MANIPULATION UNIT
DSCH – DUAL SERIAL CHANNEL
MASU – MAIN STORE UPDATE
MC – MICROCONTROL UNIT
MCH – MAINTENANCE CHANNEL
MIS – MICROINSTRUCTION STORE
MLTS – MICROLEVEL TEST SET
SAC – STORE ADDRESS CONTROLLER
SAT – STORE ADDRESS TRANSLATOR
SDC – STORE DATA CONTROLLER
SREG – SPECIAL REGISTERS
UC – UTILITY CIRCUIT

Fig. 1—The 3B20D Processor central control.

major subsystems and their associated functions are described in the following sections.

1.1.1 Microcontrol

This subsystem provides nearly all of the complex control and sequencing operations required to implement the instruction set. The microcode can support up to three different emulations in addition to its native instruction set. Other complicated sequencing functions are also stored in the microstore (MIS). The Microcontrol (MC) unit sequences the microstore and interprets each of its words to generate the needed control signals specified by the microinstruction. To optimize the execution of microinstructions, execution time depends upon the complexity of the microinstruction. Each is allocated sufficient time to implement the microinstruction in multiples of 50 nanoseconds. These times are 150, 200, 250, and 300 nanoseconds. The wide 64-bit word allows a sufficient number of independent fields within the microinstruction to perform a number of simultaneous operations. Some common and high-runner instructions are implemented with a single microinstruction.

1.1.2 Data manipulation unit

The arithmetic and logic operations are carried out in the Rotate Mask Unit (RMU) and the Arithmetic/Logic Unit (ALU). These two units are connected in series to comprise the Data Manipulation Unit (DMU) shown in Fig. 2. The RMU can rotate or shift any number of bit positions from 0 to 31 through a two-stage barrel-shift network. In addition, a selection of AND/OR operations can be performed on bits, nibbles (4 bits), bytes, half words, full words, and miscellaneous predefined patterns. The RMU outputs feed directly into the ALU. Any bit fields within a word can easily be manipulated and processed by the DMU, greatly enhancing the power of the microcode. The ALU is implemented using 2901 bipolar 4-bit processor elements (see Fig. 3).[1] Eight such chips provide two key elements: the 2-port (A and B), 16-word (RAM) and the high speed ALU. Data in any of the 16 words addressed by the 4-bit A-address-field input can be used as an operand to the ALU. Likewise, data in any of the 16 words defined by the 4-bit B-address-field input can be simultaneously read and used as a second operand to the ALU. The result can be directed to the RAM word specified by the B-field. To take advantage of the above feature, the internal 16-word RAM is dedicated as general registers. This enables the arithmetic and logical operation involving general registers and/or the output of the RMU to be performed at the optimum speed.

The logic blocks of Fig. 2 reveal the way the RMU-ALU is self-checked. The rotate unit is checked by word parity, which it preserves,

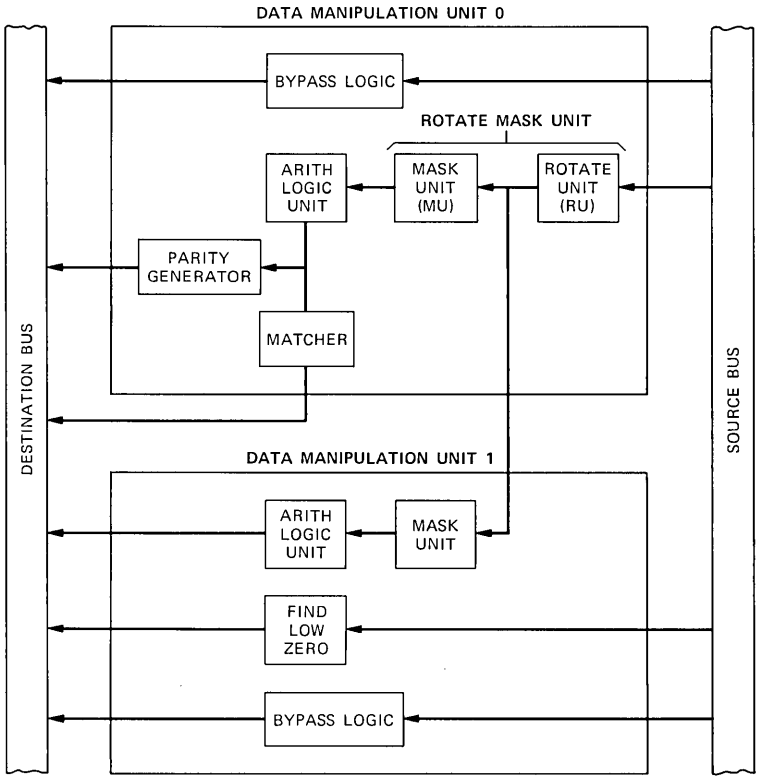


Fig. 2—Block diagram of the data manipulation unit.

and the mask unit is checked by duplication. The ALU is also duplicated. The data is taken from one ALU and parity is generated from the other. The data from one ALU is also matched with the duplicate. The underlying self-checking strategy, illustrated here and used throughout the CPU, is to use parity checking in those cases where parity is preserved and duplication of logic elements where parity is not preserved.

1.1.3 Special registers

The 16 general registers reside inside the DMU and are available to the programmer. A number of Special Registers (SREG) associated with the operation of the CC are external to the DMU. Most of them are not explicitly specified by the instruction. They are characterized by their special dedicated functions with additional inputs from sources other than the internal data bus. Their outputs are used to control and direct the operation of the processor. Some of the SREG's are: the Error Register, Program Status Word, the Hardware Status Register,

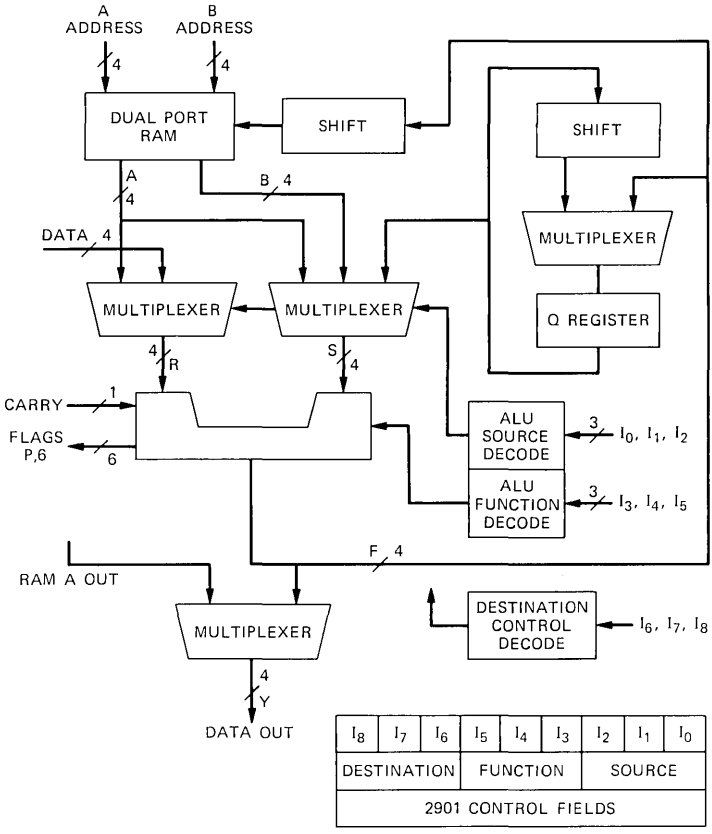


Fig. 3—The 2901 internal architecture.

the System Status Register, the Interrupt Register, timers, etc. In addition, a 32-word RAM is provided within the SREG block and is essentially available only at the microcode level. It is used for scratchpad space and preassigned registers to facilitate and enhance the power of microprogram sequences. The registers supporting memory management[2] are an example of such preassigned registers.

1.1.4 Store interface control

The store interface circuit controls the transfer of data or instructions from the memory system[2] to the CC. Several SREGs are associated with the store interface. Associated with the Store Address Control (SAC) are the Program Address (PA), the Store Address Register (SAR), and the Store Control Register (SCR). Associated with the Store Data Control (SDC) are the Store Data Register (SDR), the Store Instruction Register (SIR), and the Instruction Buffer (IB).

The SAC and SDC together make up the store interface that handles the memory addressing, the updating of the program counter, and the fetching and prefetching of instructions. The circuit ensures a continuous flow of instructions to be interpreted by the microcontrol unit.

1.1.5 Store address translation

Memory mapping is required in the implementation of a virtual addressed multiprogramming system. The Store Address Translation (SAT) facility containing the Address Translation Buffer (ATB) is the mechanism that provides a mapping between a program-specific virtual address and its corresponding physical address. Address translation hardware[2] is provided to facilitate memory management in a more efficient manner. The store address space is divided into 128 segments, each having up to 64 pages, with a page containing 2K bytes. Both the virtual and the physical address spaces are 24 bits. The complete set of virtual-to-physical address translation tables are stored in the main memory. A significant amount of time would be required by the CC in the repetitive task of dynamic address translation in using the main store tables. This translation time is reduced substantially by storing the likely-to-be-used physical address translations in a high-speed cache-like Address Translation Buffer (ATB).

1.1.6 Main store update

The Main Store Update (MASU) unit[2] provides a multiport interface to the memories as both DMA and CC attempt to use the memory. The update circuit arbitrates asynchronous requests from the on-line CC, both DMAs, and the off-line CC. The cross coupling between the memory update units permits the on-line CPU to access either memory or both memories for concurrent write operations.

1.1.7 Input/output interface

The communication path between the CC and the I/O channels is through the CCIO bus. It is a local, high-speed, direct-coupled, parallel bus. Direct memory access between the main store and peripheral units is provided by a Direct Memory Access Controller (DMAC) that communicates with intelligent peripheral units via Dual Serial Channels (DSCH). I/O channels including user-specific interfaces can be connected directly to the CC via the CCIO bus. Two such standard interfaces are the DSCH, a high-speed multiport serial interface, and the Application Channel Interface (ACHI), a high-throughput, parallel bus, peripheral communication path.

1.1.8 Cache

The cache[2] is an optional circuit equipped to improve the overall

system performance by reducing the effective memory access time. The cache is a fourway set associative memory containing a total of 8K bytes.

1.1.9 Maintenance channel

This circuit provides diagnostic access to the CC at the microinstruction level. It also is used to control basic fault recovery and system sanity functions in the off-line processor.

1.2 Major buses

The processor structure diagrammed in Fig. 1 allows three key transfers to proceed simultaneously. The first is a microcontrol path through the Microinstruction Store (MIS). The second is a computation path through the Data Manipulation Unit (DMU). The third is a memory path through the cache. The CPU is pipelined at both the microstore and store levels: microinstruction execution is overlapped with microstore read and instruction execution is overlapped with fetch. In addition, separate transfer paths are provided for maintenance and input/output.

Microaddresses pass from the Microcontrol Unit (MC) to the MIS over the 16-bit Microstore Address (MSA) Bus. Microinstructions pass from the MIS to the MC over the 64-bit Microstore Data (MSD) Bus. Control signals derived from the microinstruction fan out to the subsystems that connect to the 32-bit Source (SRC) bus and 32-bit Destination (DST) bus.

The SRC and DST buses are the primary gating paths for computations. Data and addresses pass from the DMU to the SDC and SAC, respectively, over the DST bus. The SREG are also accessed over the SRC and DST buses. The Store Bus includes both a 32-bit data field and a 24-bit address field. The store can also be accessed by the DMAC over the same bus structure.

The Maintenance Channel (MCH) controls the processor using the MSA and MSD buses. The 32-bit Bidirectional gating bus (BGB) and Maintenance (MTC) bus are used by the MCH to observe the CPU. The BGB is also used to load the writeable microstore.

The CCIO bus is a bus linking the CC with the I/O channels. Operations over the bus are programmed I/O instructions. Control of the DMA circuits is exercised over this bus but the data transfers via DMA are routed directly onto the main store bus.

II. I/O FACILITIES

The I/O facilities[3,4] are designed to meet a wide range of applications with different needs and capabilities. A modular and flexible I/O communication structure is provided by means of dedicated point-to-

point channels. The loose coupling of the processor to the peripherals allows considerable freedom to grow and expand the system with minimum physical constraints. The I/O architecture is shown in Fig. 4. Two channels have been designed for the CCIO bus, the dual serial channel and the application channel interface.

2.1 Input/output channels

2.1.1 Dual serial channel

The Dual Serial Channel (DSCH) (see Fig. 5) is a modular circuit that provides high-speed serial links between the CC or DMA and the peripheral devices. Direct memory access is only supported on devices connected to DSCHs on a DMAC. Transfers for devices directly connected to the CCIO bus are completely controlled by CC software using specific I/O instructions.

The DSCH supports word (36 bits) or block (16 words) transfers at either of two rates. For peripheral devices located within 100 feet of the DSCH a 10-MHz clock rate is used. For devices up to 250 feet away a 5-MHz clock rate must be used.

The DSCH provides a private serial point-to-point data path to each of the 16 intelligent devices it supports. Each link uses a five twisted-pair cable as the transmission media. Two of the pairs are used for bidirectional data transmission, two pairs for unidirectional clocks and the fifth pair for the peripheral device to set flags in the DSCH. Data is transmitted using RS422 compatible signaling. The other end of the cable connects to a Duplex Dual-Serial Bus Selector (DDSBS) that converts the signals from the DSCH into a parallel format. Each DDSBS can interface to two DSCHs allowing the peripheral device to be connected to both CPU's of the duplex processor.

2.1.2 Application channel interface

The Application Channel Interface (ACHI) (see Fig. 6) provides a differential dc parallel bus and control signals for high-throughput programmed I/O. The circuit can only reside on the CCIO bus and has no autonomous sequencer. The ACHI's relies on command sequences from the CC to carry out peripheral operations.

The ACHI's CCIO bus interface is similar to that of the DSCH. The same data bus, addressing, control, and response signaling is used. The parallel bus generated on the peripheral side of the ACHI consists of two unidirectional 36-bit data buses and 11 control and response leads. One data bus is for input to the ACHI and one is for output with concurrent data transfers allowed.

2.2 Direct memory access controller

The Direct Memory Access Controller (DMAC) (see Fig. 7) provides

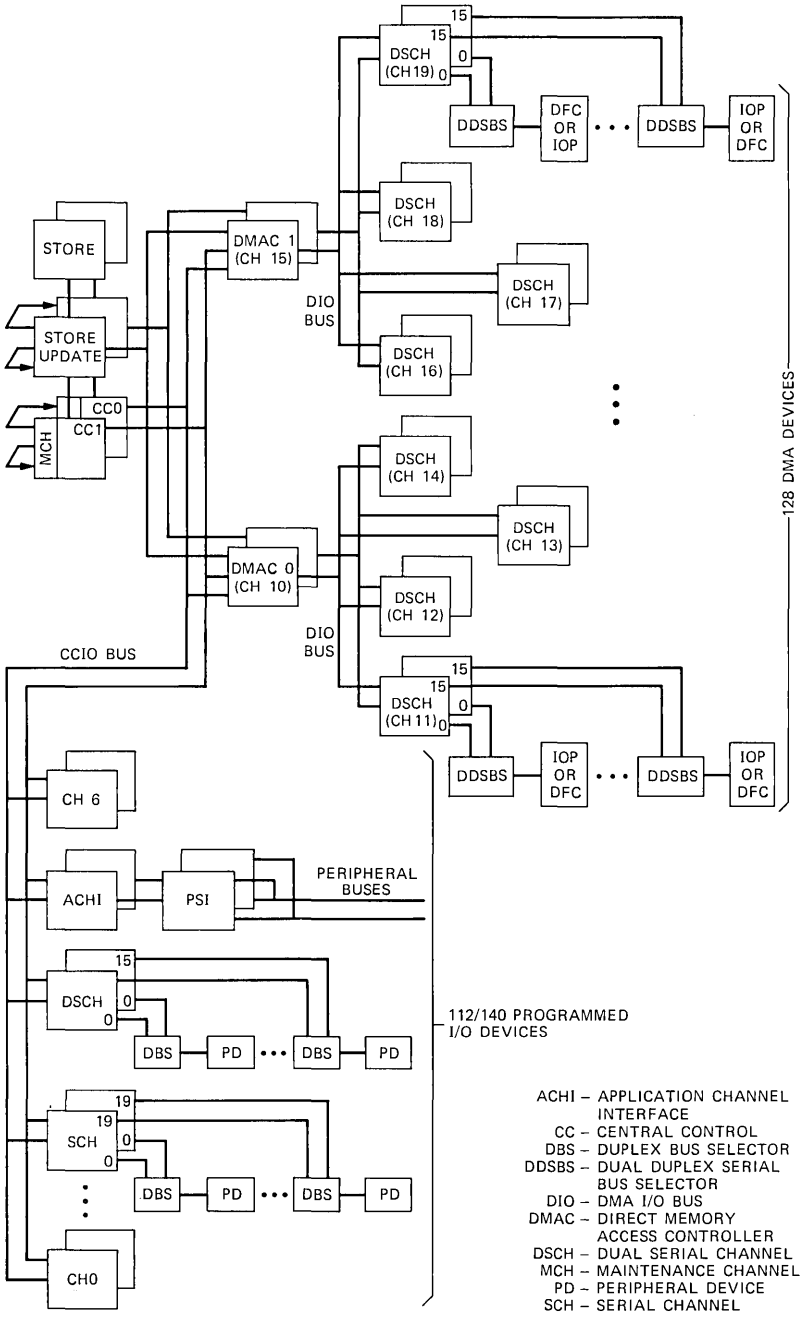


Fig. 4—The 3B20D Processor input/output architecture.

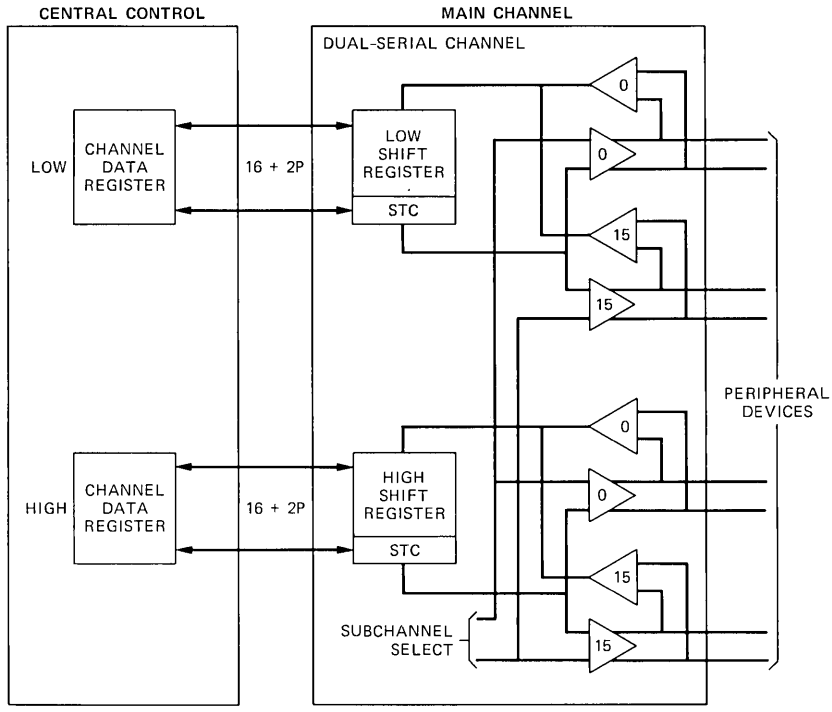


Fig. 5—Dual-serial channel.

the facility for moving data between I/O devices and main memory without having the processor involved in transferring the data. The DMA circuit consists of a DMAC and one to four DSCHs. A DMA can accommodate up to 64 devices, all of which may be active concurrently. The DMAC supports virtual addressing, word or block (16 word) transfers, device-initiated transfers, and multiple jobs for a given device. The latter function allocates a unique segment in memory to each job for a device. This prevents one job from mutilating another job's memory.

A DMA transfer is a two-step operation. Half the time is spent with the channel and peripheral device passing data and half with the channel-DMAC-main store passing data. While the former operation is in progress, the DMAC can be loading or unloading another channel from main store. The DMAC has a hardware priority circuit that gives channel 0 priority over channel 1, which has a higher priority than channel 2, etc. The devices on a given channel are prioritized in the same manner, i.e., device 0 having the highest priority and device 15 the lowest. The channel does not permit interleaving devices on less than a block or word boundary. Once a transfer of a block or word is

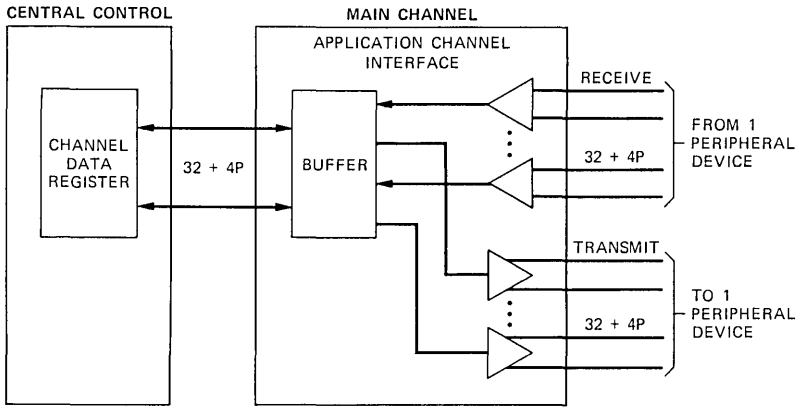


Fig. 6—Application channel interface.

initiated to a device, that transfer must be completed before another device on that channel can be serviced.

A DMA transfer is initiated by the device passing the DMAC the starting virtual address and, optionally, a transfer count. The DMAC translates the address into a physical address as described below. The transfer count is used as a check verifying that the device and DMAC are in agreement as to the number of transfers to take place.

The address-translation process used by the DMAC is the same as that used by the CC with both using the translation-page tables that are stored in main memory. Each page has protection bits defining DMA read/write access capability. The maximum size transfers that can be accomplished with a single address setup is one segment or 64 2K-byte pages. As part of the initialization process for the DMAC, the processor passes a unique page table pointer to the DMA for each of its active devices. The DMAC uses the page table pointer and the virtual address to obtain the desired physical page pointer. As the DMA transfer crosses a page boundary, the DMAC automatically accesses the page table to obtain the next physical page pointer.

After setting up the DMAC the device initiates the transfer by sending a transfer request. The DMAC will ask for the data from the device or send the data to the device in a word (32 bits) or block (16 words) mode. The device then sends another transfer request and the handshaking continues until the entire job is completed.

III. MAINTENANCE FEATURES

The 3B20D places great emphasis on maintenance features. A complete, fully developed package of such features includes self-checking, fault recovery, and diagnostic capabilities.

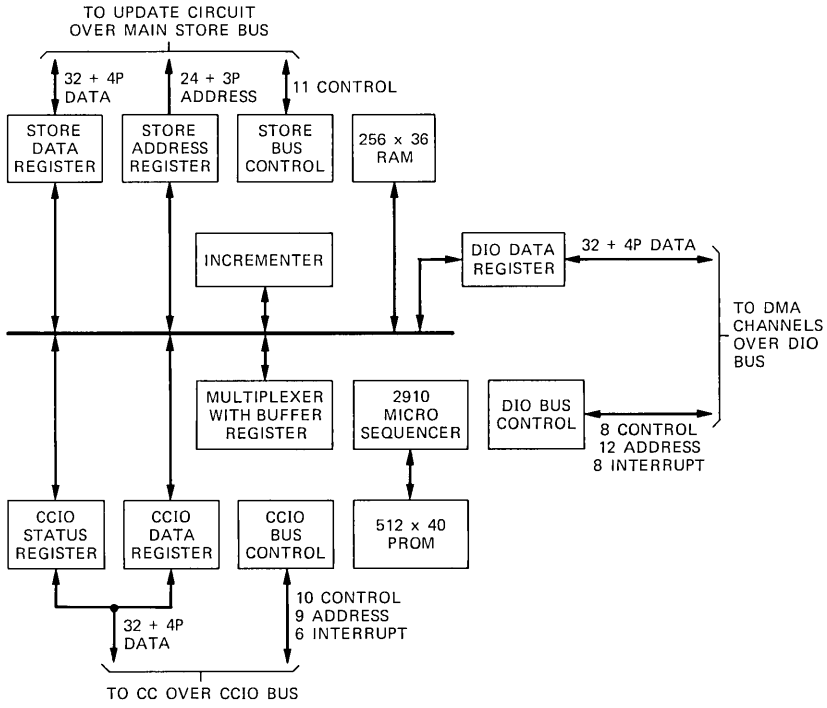


Fig. 7—Direct memory access controller.

3.1 Self-checking hardware

Self-checking is implemented in the 3B20D design for concurrent error detection. The maintenance philosophy is to provide a sufficient amount of hardware to enable detection of nearly all service-affecting single hardware faults.[5] To minimize the potential sources of errors in the main memory,[2] single-bit Hamming correction and double-bit error detection are employed. Most software faults such as memory-protection violations, illegal instructions, and out-of-range addresses also are detected. Some of the fault-detection techniques used in the 3B20D Processor are:

(*i*) Parity per byte on data paths throughout the internal CC, memory buses, and peripherals

(*ii*) Single-bit Hamming correction and double-bit error detection on the main store data

(*iii*) Duplicated arithmetic and logic unit and other control logic

(*iv*) "Watchdog" sanity timers for software faults.

Faults detected by the processor check hardware are collected together into a single error register. The action taken for a particular fault depends on its impact on the system. These actions range from

microinterrupts handled in firmware to a stop-and-switch response where control is transferred to the standby half of the duplex processor.

3.2 Hardware fault-recovery features

Fault detection is the first and most important step in realizing a highly reliable system. Almost of equal importance is the rapid recovery by the system. Recovery is achieved by a coordinated combination of hardware, microcode, and software actions. As soon as an error is detected, immediate action takes place to reconfigure the system into an error-free working system. The recovery process involves two steps: reconfiguration and initialization. To facilitate this rapid recovery from system faults, the 3B20D Processor provides the following.

3.2.1 A memory update unit

This unit couples the on-line memory to the off-line memory. The off-line processor's memory is updated on each memory write operation, whether CC or DMA originated, to provide continuous agreement with the on-line memory.

3.2.2 A maintenance channel

This unit directs the off-line processor to initialize and recover when the on-line processor has detected critical error conditions.

3.2.3 Initialization microcode

This nondestructive microcode makes critical recovery decisions when error conditions are detected. This microcode is particularly important if total software sanity is lost.

3.3 Diagnostic features

Hardware has been incorporated into the design of this system to permit a systematic approach for identifying failures via software. This diagnostic software[6] depends heavily on the maintenance channel and its associated circuitry. The primary function of the MCH is the diagnosis of one CPU by the other. The MCH is an autonomous portion of the processor which, under control of the other CPU, can provide information about the state of the machine. The MCH thus exercises the machine at its most basic level by direct access to the microprogram control. The MCH characteristics are like those of the DSCH, and the access protocols are compatible with each other.

IV. MICROCODE FEATURES

The large microstore address space in the 3B20D Processor provides a relatively inexpensive means of specifying complex instructions and

special system functions. These can be added or modified with little difficulty compared to hardwired functions.

4.1 Types of features

Microcode can reside in either PROM or in writeable control store. The following sequences have been implemented in the 3B microcode:

(*i*) Microboot—This sequence of code initializes the processor and loads both the Writable Control Store (WCS) and the first-level bootstrap program from disk. If it is unable to load from the primary disk, it automatically tries to load from the backup disk.

(*ii*) Tapeboot—This sequence of microcode initializes the processor and then copies data from a standard nine-track magnetic tape to the disk. The microboot routine can then be used to initiate program execution.

(*iii*) Basic instruction set—This set of microcode implements the native instruction set and special instructions that are specific to the 3B20D Processor.

(*iv*) External interrupt routine—This sequence is invoked only upon the completion of an instruction with an interrupt pending. The microcode saves the state of the system, then transfers to an interrupt event handler routine.

(*v*) Error interrupt routine—This section of microcode is entered in case of a hardware or software error. The microcode saves the state of the machine and then transfers control to a routine that attempts to recover processing without switching to the other machine. This interrupt can be encountered between any microinstructions.

(*vi*) Memory management trap—This sequence of microcode is entered if the virtual address of the memory fetch cannot be translated directly to a physical address by the address-translation hardware. The microcode reads the segment and page tables of the active process, loads the hardware translation unit with the translation information and then reactivates the memory access.

(*vii*) Maintenance channel instructions—These instructions allow the processor to communicate with its duplex mate via the maintenance channel.

(*viii*) Diagnostic sequences—A section of the WCS is reserved for the diagnostic programs to load and then execute special microcode sequences.

(*ix*) Miscellaneous routines—There are several miscellaneous microcode routines that are used for functions such as to load the writable control store and to load the hardware matcher registers in the utility circuit.

(*x*) Emulation routines—Up to three additional sets of microcode can be loaded that allow the 3B20D to emulate other machines.

4.2 Microcode development

Several software and hardware tools were developed that allowed microcode to be generated and debugged with relative ease. Of principal importance are the MICA[7] bit-sliced assembler and the Micro-level Test Set (MLTS).[6] These tools allowed a laboratory utility computer to be used to assemble microcode, and to control the 3B20D via a link to the MLTS. The facility was used to load writable microstore, access main store, access registers and control sequencing at the microcode level, set breakpoints and provide trace capability.

During the development phase of the 3B20D processor, substantial advantages were realized by the fact that the system was microprogrammed. Major improvements in the instruction set architecture were accommodated by microcode changes. Features have been added to enhance error recovery, to permit transferring bootstrap programs from magnetic tape to disk, and to improve system performance.

V. SUMMARY

The 3B20D CPU is a 32-bit machine with 24-bit addressing. Hardware features have been provided to support a modern general-purpose operating system, e.g., virtual-to-physical address translation. Other features include microprogram implementation, emulation capability, high-speed data cache, high-speed interrupt stack, self-checking circuits, extensive diagnostic access, and high-availability duplex operation.

The standard I/O communication between the CC and the peripherals is by means of a DMA controlling dual-serial channels, capable of transmitting an effective rate of 2M-bytes/s. The DMA can operate in a word-transfer mode or a block mode of 16 words per block. The loose coupling of the channels between the processor and the peripherals permits considerable freedom in expanding a system.

VI. ACKNOWLEDGMENTS

Many individuals have contributed in a significant way to the organization, design, and implementation of the 3B20D central processing unit. The authors would like to particularly acknowledge the contributions of C. A. Borcher and C. W. Hoffner to both the development of the 3B20D Processor and to the content of this paper.

REFERENCES

1. *The AM2900 Family Data Book*, Sunnyvale, California: Advanced Micro Devices, 1979.
2. I. K. Hetherington and P. Kusulas, "3B20D Memory Systems," B.S.T.J., this issue.
3. A. H. Budlong and F. W. Wendland, "3B20D Input/Output System," B.S.T.J., this issue.

4. R. E. Haglund and L. D. Peterson, "3B20D File Memory Systems," B.S.T.J., this issue.
5. R. C. Hansen, R. W. Peterson, and N. O. Whittington, "Fault Detection and Recovery," B.S.T.J., this issue.
6. J. L. Quinn, R. L. Engram, and F. M. Goetz, "Diagnostic Tests and Control Software," B.S.T.J., this issue.
7. J. T. Beckett and S. W. Ng, "A General Purpose Microcode Assembler," Proc. IEEE/COMPSAC, 1978, pp. 84–89.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

3B20D Processor Memory Systems

By I. K. HETHERINGTON and P. KUSULAS

(Manuscript received March 17, 1982)

The memory system supplied with the 3B20D Processor provides a high-reliability, high-performance, main-frame memory for use by the 3B20D Central Control and Input/Output system. The memory system is designed using a collection of high-speed, static and dynamic memory devices and appropriate logic controllers. In addition to providing basic on-line storage for program text and data, the memory system provides hardware assistance for virtual-to-physical address translation, access protection, memory resource arbitration, and performance enhancement utilizing a high-speed cache memory. The technology used in implementing these functions includes state-of-the-art 64K dynamic random access memory devices and high-speed TTL-compatible gate-array integrated circuits.

I. INTRODUCTION

The memory system associated with the 3B20D Processor[1] includes a 16-megabyte memory, a high-speed cache memory, and hardware assistance for the virtual-to-physical address translation process, access protection, and memory resource arbitration functions.[2] The memory system utilizes high-speed, static and dynamic memory devices and appropriate logic controllers.

The block diagram shown in Fig. 1 highlights the major components and interconnections of the 3B20D Memory System. The diagram indicates the memory system related control, address, and data paths, including the interconnection to the fully duplicated system. Internal central control data paths associated with the Store Address Translator (SAT) are not shown.

As indicated, the 3B20D Memory System is comprised of a 16-

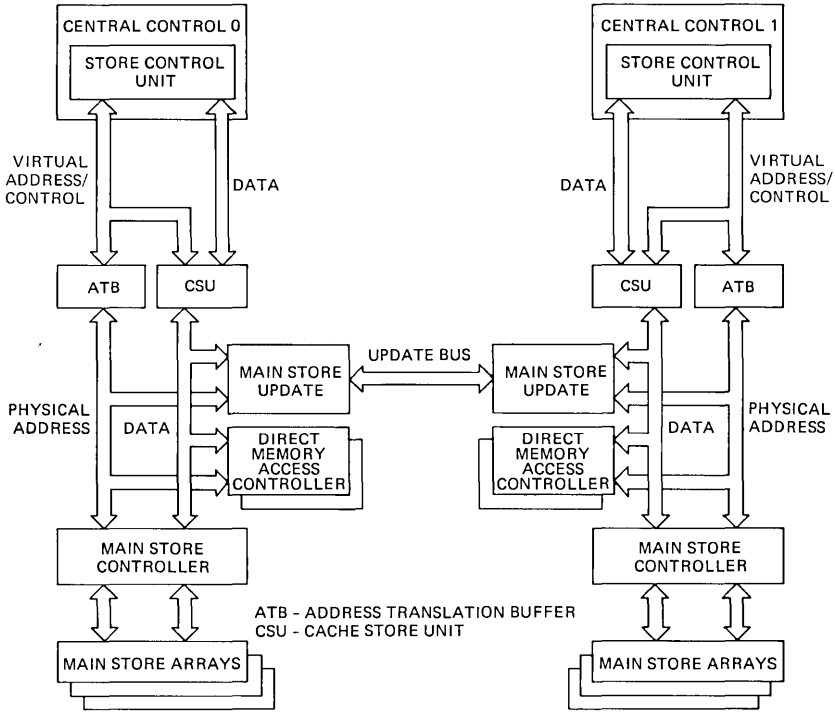


Fig. 1—Block diagram of the 3B20D Processor memory system.

megabyte main store, a Main Store Update (MASU) circuit providing interconnection control between the duplicated memory systems, a store address translator, and an optional Cache Store Unit (CSU). The subsystems that use the memory system are the central control and the Direct Memory Access Controller (DMAC). Memory operations are initiated by either of the duplicated central controls or DMACs. In the DMERT environment, one CC/DMAC combination has control of the system and initiates all memory operations. The MASU circuit performs write operations to both duplicated main stores. In this way, both main stores are kept up to date with currently executing programs, data, and I/O transactions. If a central control switch is needed, it can be accomplished quickly since the off-line central control can immediately use a fully updated main store.[3]

The 3B20D Processor supports memory management that allows programs to be written using virtual addresses without regard to where they actually reside in memory. An address translation hardware assistance circuit, the SAT, is provided in the 3B20D. The SAT provides high-speed access to the most recently used address-translation and access parameters. It serves as a cache memory for the

translation parameters that apply to the software processes executing in the central control. The same address-translation mechanism implemented by the SAT is used by the DMAC.[4,5] Thus, processes executing in the central control can share virtual address spaces with peripheral devices communicating through the DMAC. In addition, once this address-translation mechanism is established by the operating system, the central control or the peripheral devices may initiate memory operations independently.

The remainder of this paper discusses, in more detail, the major components of the memory system. The discussions cover the operational aspects of the design, self-checking, error reporting, error recovery, and diagnostic features[6] provided in the memory system design. Other topics addressed include performance, reliability, device technology, and environmental considerations.

II. STORE ADDRESS TRANSLATOR

In any computer system, the main memory is an expensive resource and has to be efficiently managed for improved system performance. Because the 3B20D Processor supports a multiprogramming environment, the memory may be shared by several user programs, each having access to the full virtual memory spectrum. Hence, a mechanism for relocation and protection of a user's address space (text and data) is required. The 3B20D translates the virtual addresses used within the processor and I/O subsystem to a real or physical address used within the main store. To reduce the overhead associated with the address-translation and protection mechanism, the 3B20D uses an SAT that contains a high-speed address cache called the Address Translation Buffer (ATB).

2.1 Address space partitioning

For the purpose of sharing, relocating, and protecting, the 24-bit address space is partitioned into "segments." A segment is a contiguous block of sequential virtual addresses the size of which may range from 1 byte to 128K bytes. To reduce memory breakage due to fragmentation, each segment is further partitioned into 2K-byte blocks called "pages." A segment has up to 64 pages and an address space can have up to 128 segments. The virtual address generated by the processor is divided into a 7-bit segment field, a 6-bit page field, and an 11-bit byte offset (displacement) as shown in Fig. 2.

2.2 Address translation process

The SAT translates the 24-bit virtual address to a 24-bit physical address used by the CSU and the main store. This translation is conceptually a two-step table-lookup operation and is controlled by

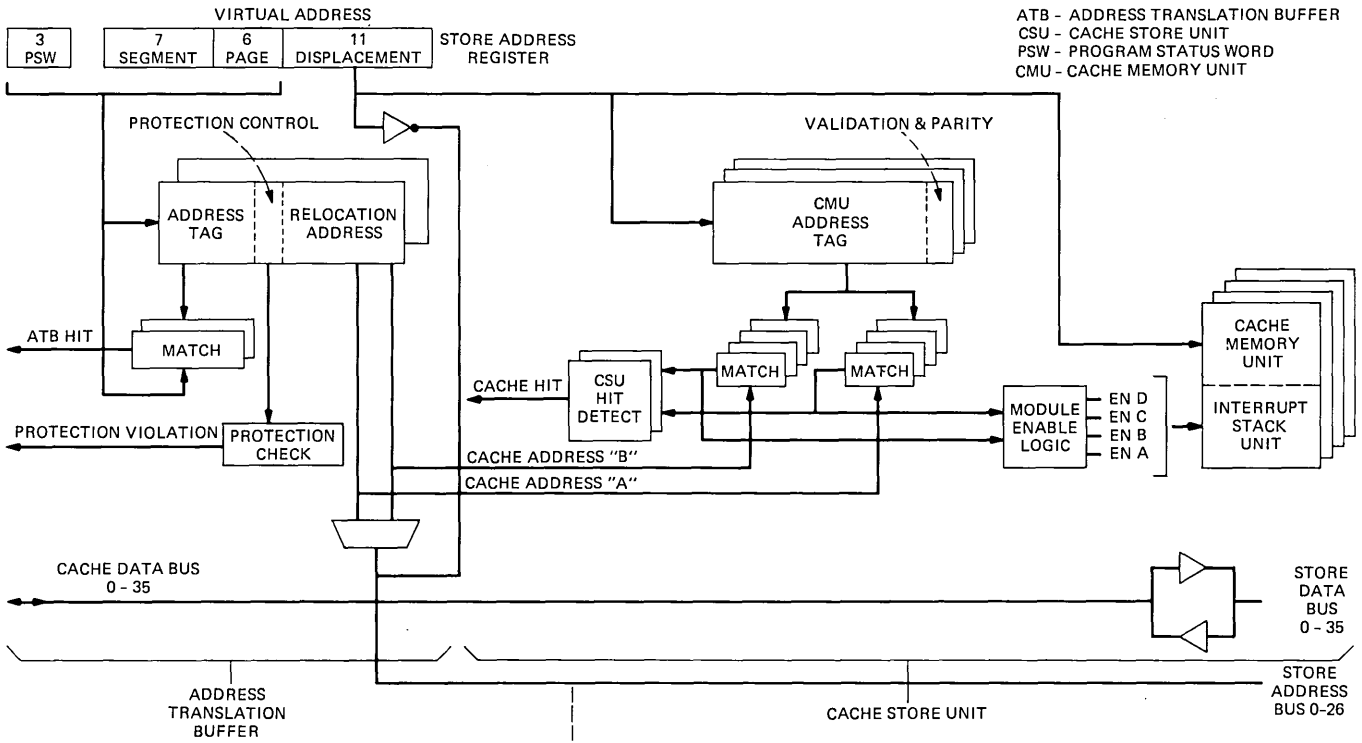


Fig. 2—Block diagram of the 3B20D Processor store address translator and cache store unit.

the Program Status Word (PSW), the segment base register (SBR), and two main store resident tables (the segment table and the page table). The resident operating system manages these registers and tables. Recently used translation information is kept in the high-speed ATB and is accessed concurrently with the CSU.

2.3 Segment base register

The currently executing process's[2] address space is defined by the SBR. The SBR is a 32-bit register that contains the physical address pointer to the beginning of the process's segment table and its length (which is equal to the number of segments allocated to the process). There are eight such SBRs in the processor to accommodate eight independent address spaces at any time. These eight address spaces are assigned by software, and in a DMERT environment, a minimum subset of four is allocated to the kernel, a kernel process, a supervisor process, and a user process. The fields in an SBR are allocated as follows:

(*i*) Segment table address (22 bits)—This field points to the beginning of the segment table in main store.

(*ii*) Unused (3 bits)—This field is presently not used.

(*iii*) Segment limit (7 bits)—This field designates the length of the segment table.

2.4 Segment table

The segment table (one exists for each process) contains a descriptor for each segment of the process. Each entry is 4 bytes long and resides on a full word boundary in the main store. It is partitioned into three fields as follows:

(*i*) Page table address (22 bits)—This field points to the beginning of the page table.

(*ii*) Page table length (6 bits)—The length of the page table is one more than the value in this field.

(*iii*) Protection bits (4 bits)—Three of these bits indicate whether the segment is readable, writable, or executable. One bit indicates the validity of the entry.

2.5 Page table

The page table (one exists for each segment) contains a descriptor for each page in the segment. Each entry is 4 bytes long and resides on a full word boundary in the main store. The entry has four fields that indicate:

(*i*) Relocation Address

(*ii*) DMA usage

(*iii*) Protection
(*iv*) Unused.

The process of virtual to physical address translation is described in the following sections.

2.5.1 Segment table lookup

The segment field of the virtual address is first compared against the segment limit field of the SBR to establish whether the addressed entry is within the table. If the segment limit is less than the segment field, a "segment length error" exception is recognized and control of the processor is transferred to handle the exception. If there is no such error, the segment field of the virtual address and the segment table address in the SBR are used to index into the segment table in the main store and access an entry.

The fetched segment table entry is checked for validity. If the entry is invalid, control is transferred to exception handling microcode. If the entry is valid, the page field of the virtual address is compared against the page table length field in the segment table entry. If the value of the page field is greater than the maximum number of pages in the segment, the control is transferred to a microcode routine to handle the error. In case of no such error, a page table lookup is initiated.

2.5.2 Page table lookup

The contents of the page-field of the virtual address and the page table address field of the fetched segment table entry are used to index into the page table to fetch a page table entry. The protection bits obtained from the segment table entry and the page table entry are ANDed, and a check is made for a possible protection violation. In case of such an error, an error-handling routine is initiated. If there is no protection violation, then the physical address is generated by concatenating the relocation address field of the fetched page table entry and the byte offset field of the virtual address.

2.6 Address translation buffer

The Address Translation Buffer (ATB) is provided to reduce the overhead associated with the address-translation mechanism. The ATB is capable of holding 128-page table entries for eight different address spaces (for a total of 1024 entries) and is organized as a two-way set associative memory. When the processor initiates a store access, the ATB checks whether the corresponding physical address is available in its memory. If the physical address is available (called a "hit"), it is sent to the main store. But, if the physical address is not available (called a "miss"), an "ATB Miss" processing microroutine is

initiated which does the segment and page table lookups and loads the obtained physical address into the ATB. The access is then restarted. Subsequent access to the same page results in hits.

2.7 ATB control bits in PSW

The PSW contains a field that provides control for the memory management. The ATB functions can be enabled or disabled. When disabled, address-translation and protection check functions are disabled, and the virtual addresses are directed to the CSU and the main store as physical addresses.

The PSW also controls process communication capabilities between virtual address spaces. The PSW designates the Primary Segment Base Register (PSBR) and the Secondary Segment Base Register (SSBR). Each of these registers defines a virtual address space. Generally, the PSBR is used to select one of the eight blocks of the ATB for address translation. Under PSW control, however, the SSBR can be used for read, write, or both read and write memory operations. All instruction fetches use the PSBR irrespective of the contents of the PSW. Special instructions are provided to manipulate the PSW. This hardware feature can be used to move data between two address spaces very efficiently.

2.8 ATB operation

Figure 2 shows the block diagram of the ATB. When an access is initiated by the processor, the ATB is accessed using 3 bits of the PSW (selecting one of eight address spaces), low order 3 bits of the segment field, and low order 3 bits of the page field of the virtual address. The address tag fields of the ATB are matched against the corresponding bits of the virtual address. Simultaneously, the two relocation address fields are directed to the cache. A hit is generated in case of a successful match.

If a hit is detected, then a check is done to see whether the access is allowed on that page. The processor and the cache are informed in the case of any protection violation.

If a miss is detected, the cache ignores the relocation addresses. An "ATB Miss" microroutine is initiated by the processor and the new ATB entry is loaded into one set of the ATB using a defined replacement algorithm. The access is then restarted.

As mentioned before, the ATB is capable of handling translations for eight tasks at any instant. But, when a new task is allocated to a block of the ATB, the entries associated with the previous task have to be invalidated. A dedicated 8-bit counter is provided to do this invalidation with minimum microcode overhead.

The address translation mechanism provided by the SAT is com-

patible with that provded by the DMAC. In this way, processes executing on peripheral controllers can share virtual address spaces with processes executing on the central control.

2.9 Redundancy

The ATB hardware is completely self-checked. Parity is checked over the Store Address Register (SAR), the ATB entries and the ATB related bits in the PSW. Parity bits are regenerated for the physical addresses to be checked at the cache and main store. The hit and protection check logics are duplicated and matched, and can be exercised under maintenance control. In case of any hardware faults in the circuit-pack, the access is aborted and control is transferred to a fault-handling routine. The ATB memory can be written and read over the source and destination buses of the processor.

III. CACHE STORE UNIT

The Cache Store Unit (CSU) reduces the effective access and cycle time of store operations for the 3B20D Processor. Combining a relatively small, high-speed "cache" memory with a large main store results in a system with an average access time approaching that of the high-speed cache but with the low cost per bit and storage capacity of the Main Store.

The concept of a cache takes advantage of a general programming characteristic of locality of reference. Most references to memory tend to be highly localized or clustered into small groups at any given time, and regions tend to change relatively slowly during the course of program execution. Thus, a relatively small, high-speed CSU contains the most often used words from the main store and thereby reduces the average access time of the reference.

3.1 Organization

The CSU is organized into two sections: an interrupt stack section and a cache section. Since the main store contains a much larger storage capcity than the CSU, a mapping function is required to compress the main store address range into the much smaller CSU. The compression is achieved by adding a tag which contains the physical page address to each word of cache data storage. To increase the probability that a main store word is in the CSU (a "hit"), the cache is organized as a four-way set-associative memory. Each one of the four cache modules contains 2K bytes of high-speed storage. The interrupt stack section also contains 8K bytes of memory.

Since the cache is four-way set-associative with the main store, there is a one for one correspondence to the page offset address but full associativity for the page portion of the address. Thus, word 0 from

any of the four cache modules may correspond to word 0 from any page in main store.

3.2 Operation

Since the cache section of the CSU functions as an associative memory, a match search in the cache is performed in parallel with the translation of the virtual-to-physical address by the ATB. This hardware parallelism is illustrated in Fig. 2. An access causes the low 11 bits of the virtual address to select a unique page offset on each of the four cache modules. The page portion of the virtual address (high 13 address bits) is translated by the two-way set-associative ATB. Each of the four cache tag modules is matched to the two translated physical page addresses. The ATB will indicate to the cache which of the two translated addresses is valid. If one of the four cache tag modules matches this translated page address, the CSU will generate a hit signal to the CPU and gate the associated word onto the cache data bus.

Functionally, the CSU interconnects the CPU and main store. The CSU connects to the CPU via the cache address bus, cache data bus, and control leads. Because the CSU interconnects the CPU and main store, DMA transfers to main store will not prevent the CSU from being accessed by the CPU. Thus, CPU contention for the main store is reduced since most references will result in a CSU hit.

During system initialization, all locations in the CSU are invalidated. When the CPU references the memory system and a cache miss results, the referenced word is automatically copied into the CSU. If a word needs to be copied from the main store and all four cache modules contain valid data, a random replacement algorithm selects the cache group in which a word will be replaced. Once the cache has been initialized by the system, the operation is transparent to the software with the exception of enhanced performance.

The CSU contains arbitration and sequencer control logic to automatically update its data and arbitrate between CPU and DMA write operations. When the DMA writes into the main store the CSU checks if the DMA reference is in the cache. If the word is in the cache, it is automatically invalidated by the CSU sequencer. While the CSU is checking for a DMA write hit, the CSU indicates a busy condition to the CPU. DMA reads from the main store do not result in any operation from the CSU.

3.3 Interrupt stack

The CSU also contains an 8K-byte high-speed memory which is used by the CPU to reduce the interrupt response time when the CPU is in the kernel operating mode. When the CSU is in the interrupt

stack mode, the CSU will generate 100-percent hits for both read and write operations.

3.4 Software support

The CSU is designed to support the software organization of the DMERT operating system and the "C" programming language. Call and Return are frequent operations with the C language. The CSU has built-in hardware algorithms to function as a high-speed data stack. A stack write operation (CSAV) will force the data to be copied into both the cache and main store. A stack read operation (CRET) will cause the data to be read from the cache and then invalidated. This location in the cache is then available for use by new data. The data stack is part of the cache section and is totally separate from the interrupt stack.

3.5 Self-checking

The self-checking philosophy of the CSU is to provide immediate detection of faults that cause errors and nonimmediate detection of faults that affect the performance of the CSU. The CSU has built-in self-checking hardware that monitors the operation of the CSU. Faults detected by the self-checking hardware include CSU sequencer errors, multiple writes into the CSU, multiple cache hits, accessing errors in the tag memories, and cache write errors.

A multiple hit is an example of a catastrophic fault since it would result in the contents of two or more cache words being ORed onto the cache data bus. This type of failure is prevented by duplication of the hit logic and by providing multiple hit detectors that monitor the hit logic and the module enable logic. The cache also generates and checks parity over the tag and address bits.

The CSU also provides diagnostic access to the "internals" of the circuit. When the CSU is configured in the maintenance mode, the normally associative memory tages are configured to function as conventional RAM memories. In addition, special access is provided to the counters, CSU sequencer, and various status bits.

IV. MAIN STORE

4.1 Configuration

Physically the main store can consist of one or two modules. Each module contains one Main Store Controller (MASC) and up to sixteen Main Store Arrays (MASA). The central control can directly address 16 megabytes of text or data.

The preproduction design of the main store was based on the initial use of 16K Dynamic Random Access Memory (DRAM) devices. The design was organized to allow evolution to higher-density devices. The

production design initially used a Western Electric 64K DRAM[7] that utilized two power supply voltages. The current design uses a newer Western Electric 64K DRAM that permits denser packaging. This design is implemented using a single circuit pack main store controller and a 1-megabyte MASA circuit pack. Thus, all 16 megabytes of addressable memory are contained in one module.

4.2 MASU operational aspects

The MASU circuit provides control of main store bus communication between duplex Control Units (CU), allowing the MAS in the standby CU to be kept up to date. The MASU also controls the use of the main store buses by the central control, DMA, and other CUs. The MASU in the active CU in a duplex configuration gives the highest priority to the central control followed by DMA 0, DMA 1, and operations from the other CU, respectively.

Communication with the Main Store is over the address bus, data bus and the command bus. The command leads indicate whether the operation to be performed is a write, read, clear, byte, halfword, or maintenance operation. Valid operations to the MAS include:

(*i*) Write full word
(*ii*) Write halfword/byte
(*iii*) Read word
(*iv*) Read and clear fullword
(*v*) Read and clear halfword/byte
(*vi*) Maintenance write (nonmemory operation)
(*vii*) Maintenance read (nonmemory operation).

The MASU communicates asynchronously with the MASC by the use of the Store Go signal (SGO) and Store Complete signal (SCM). Prior to issuing the SGO to the MASC, the MASU issues an address and data bus enable to the requesting unit with the highest priority. The MASU then issues the SGO to the MASC. The MASC upon receiving the SGO begins a timing sequence that selects the addressed MASA and issues a GO signal (GOI) to the MASA. The selected MASA then allows data to be read or written at the specified address depending upon which main store operation was decoded by the MASC.

The MASC performs various error checks during the timing sequence to ensure the integrity of the MASC and MASA. In the event an error does occur, the MASC sends an error signal to the requesting unit. Depending upon the operation being performed and the state of the error sources, the MASC at a specific point in the timing sequence issues an SCM to the MASU to indicate that the operation has been completed. The MASU can then grant the main store buses to the next requesting unit with the highest priority.

4.3 Self-checking

The 3B20D main store was designed with a significant amount of self-check circuitry. The partitioning of the circuitry on the circuit packs also was designed to improve fault detection. For example, the data bus transceivers on the memory array have only one bit from each parity field partitioned in each device. In this way a fault affecting either one of the bits or all the bits in the device will be detected by a simple parity check. Similarly the address, RAS, and CAS drivers for the memory devices on the MASA were partioned so that a fault associated with one of these drivers would affect at least two bits in each of the data-parity fields. By affecting more than one byte the probability of detecting the error by failing a byte-parity check increases.

To ensure bus integrity the MASC checks the address, data, and command bus parity between the MASC and the MASU. If the address parity check fails during a write cycle to the memory, circuitry in the MASC prevents the writing of data into the addressed memory location. Thus, invalid information will not be written into the memory. If the other parity checks fail, an error is signaled but an undesired memory operation may take place. The MASC internally monitors the timing sequencer and refresh address counters to ensure that they are functioning properly. Circuitry also checks the SGQ and SCM signals between the MASC and MASU to ensure that "handshaking" that takes place between the two units is functioning properly. The MASC by checking four selected responses from the MASA also can check communication to the MASA. The MASC can determine from the select responses whether an MASA was selected or not, and it can also detect if more than one MASA responded.

4.4 Error correction

The 3B20D main store performs error detection and correction. Error correction circuitry on the MASC and Error Correction Coding (ECC) of data in the MASA result in the correction of all single bit errors. This circuitry also flags all double and detectable multi-bit errors.

In the 3B20D central control, byte parity is maintained over each byte of the data word. The main store uses the existing byte-parity bits in a modified form of the Hamming code.[8] By adding four additional ECC bits (in addition to the byte-parity bits) the main store can then perform single-bit correction and double-bit failure detection. When presenting data to the system the MASC maintains byte parity by gating the byte parity bits to the MASA from the MASU, and vice versa. On the MASC five gate arrays are used to implement the error-checking and correcting circuitry. One gate array code is used for each

of the four bytes. The fifth gate array code is used for the additional four ECC bits.

The MASC issues an error signal when it detects a location in MAS in need of correction. The MASC presents the corrected data to the system but will not automatically rewrite correct data at the location. The actual rewriting of data is handled by the software error interrupt handler. In the event of a noncorrectable error, the error interrupt handler reads the data from the standby CU—if it is in a duplex configuration—and uses that data to rewrite the faulty location.

4.5 Diagnostics

The 3B20D MASC provides maintenance access to a significant portion of the circuitry in the main store. This maintenance access is used by microcode to initialize the MASC and by diagnostics to test the functional operation of the MAS.

The MASC issues various maintenance commands within the MAS when the maintenance command, address, and SGO signal are presented to the MAS. The maintenance operation is decoded off the address and is only valid for one MAS cycle. By using an address decoded maintenance command, the state of certain data bits can be latched, providing the latched maintenance state until cleared. The MASC under diagnostic control uses the various maintenance commands and states to perform the following operations:

(*i*) Control address loop-around (address returns on data bus)
(*ii*) Control the refresh circuitry
(*iii*) Control the error correction circuitry
(*iv*) Control the error detection/reporting circuitry.

By using the maintenance capability, the diagnostics can verify the integrity of the bus structure, the MASA, and the MASC. The data integrity of each MASA is tested by performing a series of data pattern tests on each MASA equipped.

V. ACKNOWLEDGMENTS

The authors would like to thank D. J. Matter, C. M. Narayanan, and D. M. Trampel for their assistance in the preparation of this manuscript and their contributions to the 3B20D memory system design. D. R. Draper and W. A. Read also have made significant contributions to the main store design.

REFERENCES

1. M. W. Rolund, J. T. Beckett, and D. A. Harsm, "3B20D Central Processing Unit," B.S.T.J., this issue.
2. M. E. Grzelakowski, J. H. Campbell, and M. R. Dubman, "DMERT Operating System," B.S.T.J., this issue.

3. R. C. Hansen, R. W. Peterson, and N. O. Whittington, "Fault Detection and Recovery," B.S.T.J., this issue.
4. A. H. Budlong and F. W. Wendland, "3B20D Input/Output System," B.S.T.J., this issue.
5. R. E. Haglund and L. D. Peterson, "3B20D File Memory Systems," B.S.T.J., this issue.
6. J. L. Quinn, R. L. Engram, and F. M. Goetz, "Diagnostic Tests and Control Software," B.S.T.J., this issue.
7. R. P. Cenker, D. G. Clemons, W. R. Huber, J. B. Petrizzi, F. J. Procyk, G. M. Trout, "Fault Tolerant 64K Dynamic Random Access Memory," IEEE Transactions of Electronic Devices, *ED26*, No. 6 (June 1979), pp. 853–60.
8. R. W. Hamming, *Coding and Information Theory*, Englewood Cliffs, N.J.: Prentice-Hall, 1980.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

3B20D Packaging and Technology

By S. H. KULPA, J. M. BROWN, and A. W. FULTON

(Manuscript received March 18, 1982)

The 3B20D Processor is built using a broad range of complex and high-performance integrated circuits. These integrated circuit devices are packaged and interconnected utilizing Bellpac™ *packaging system technology. Used throughout the processor are high-density multilayer printed wiring boards with high pin-out connectors. The elements of the technology were combined using computer-aided design tools to assure optimized system thermal and electrical performance.*

I. INTRODUCTION

The cost, performance, and schedule objectives of the 3B20D Processor and the overall complexity of the design required:

- A full spectrum of circuit integration
- Common integrated circuit specifications
- A broad range of semiconductor memory devices
- A standard hardware packaging technology
- Quick turnaround prototype circuit packs
- High interconnection capability at the circuit pack level
- A hierarchy computer aided design (CAD) process
- Comprehensive design audits in the CAD process.

The use of *Bellpac** packaging system technology provides a dense, high performance packaging system.[1] This includes a substantial increase in the number of devices allowed per circuit pack and the number of contacts per pack over previous technologies. The use of *Bellpac* packaging technology also allowed a low system cost since the

* *Bellpac* is a trademark of Western Electric.

hardware is Bell System standard and manufactured in high volume. Complex integrated circuits from a variety of vendors were employed throughout the 3B20D Processor. A wide variety of integrated circuits and comprehensive specifications were used to optimize processor performance, cost, and reliability. *Bellpac* packaging technology and the integrated circuits were coupled together through a sophisticated CAD system. The CAD system is comprised of design, analysis, simulation, and audit tools at device, circuit pack, unit, and frame levels.

II. INTEGRATED CIRCUITS AND APPARATUS PERFORMANCE

2.1 General device requirements

The integrated circuit device family used in the 3B20D Processor consists of approximately 280 codes of TTL (Transistor-Transistor Logic) compatible SSI (Small-Scale Integration), MSI (Medium-Scale Integration), and LSI (Large-Scale Integration) devices. These codes were selected to optimize the cost and performance of the wide range of processor functions included in the central control, memory, and peripherals. Device performance range varies greatly from the high-speed, speed-selected, bipolar Schottky TTL parts in the central control to relatively slow metal oxide semiconductor microprocessor peripherals in the I/O area. The scale of integration varies from SSI devices driving heavily loaded buses to VLSI (Very Large-Scale Integration) memory and microprocessor devices. Whenever possible, where system performance and/or cost was not jeopardized, the same parts were used in several different areas of the design to minimize the number of unique integrated circuits. Where the same device could be used for multiple functions, the device specification was written to cover the extreme needs of all applications.

To assure that all of these devices would interface with each other, common device specifications were established. The temperature range for the devices was specified as 0–95°C to accommodate central office equipment requirements and to keep cooling costs low. Supply voltage limits were established at 5 volts ± 10 percent to minimize power generation and distribution costs. The interface levels were established based on TTL standards (i.e. Vol $\leq$ 0.4 volt, Voh $\geq$ 3.4 volts) consistent with the temperature and supply standards described above. Other common standards included dual in-line package (DIP) dimensions and lead finish specifications to ensure manufacturability and reliability.

As the project progressed through its phases, the reliability requirements for devices varied. For the initial system models, high-reliability devices with good electrical performance were essential to minimize

interference with software/hardware integration. This was particularly true with devices on prototype wire-wrap hardware, which generate substantially more noise and crosstalk than the production multilayer designs. Since much of the prototype hardware was built before final device specifications could be negotiated with vendors, high-quality military grade parts were procured wherever possible and special screens were imposed where high-reliability parts could not otherwise be obtained. For production units, reliability requirements were established based on system availability objectives.[2] For most of the codes this worked out to a 100 FIT* objective per device. This objective was translated into specific packaging, burn-in, screens, and life-testing requirements in the device specifications.

Special devices required creation of a number of expanded specifications. For example, to simplify circuit pack testing or to provide a low-cost method for changing information contained in PROMs, reliable socketing was required which, in turn, required special gold lead finishes on the devices. A small number of other devices, for cost or availability reasons, were specified at reduced temperature and/or supply voltage ranges.

2.2 Small-scale and medium-scale integration devices

There are about 160 codes of devices that can be classified as having small-scale integration (SSI) or medium-scale integration (MSI). These consist of two basic device groups: LSTTL and STTL. The LSTTL gates are typically characterized as having 10 ns, 2 mW parts, and the STTL gates as having 5 ns, 20 mW parts. Both families meet industry standards and are widely available. The identical functions are generally available in both families. Due to the much lower power of the LSTTL devices, they were preferred except where speed was critical. To maximize the use of the LSTTL devices a special output current drive requirement was specified that effectively doubled their capacitive drive performance under worst-case conditions. Since the STTL devices were used in the speed-critical paths, their performance had a major part in establishing system performance. On a subset of these devices, special speed screens at high-capacitive loads were specified. The increased cost of these screens on these few device codes was minimal at the system level, but the system performance gain was substantial.

In addition to the standard logic devices in the SSI and MSI category, there were a variety of special devices such as delay lines (10 codes) and oscillators (10 codes). These devices generally had STTL and LSTTL interfaces to the other devices.

---

* FIT is defined as one failure in 10^9 operating hours.

2.3 Memory devices

The semiconductor memory device requirements for the various processor functions were broad. They included dynamic RAMs, static RAMs, PROMs, Electrically alterable PROM (EPROM), and First In First Out (FIFO) memories in a variety of functional organizations and speed ranges.

The dynamic RAMs used in the 3B20D for main storage are 64K-bit chips manufactured by Western Electric. They are organized 64K words by 1 bit and have an internal redundancy that was used to increase yield.

Static RAMs are used in a variety of applications including cache, microstore, memory management, and the address/data storage for the microprocessor-based circuits. The first three applications require high-speed memory, while the address/data storage requires high density at moderate speed (16K bits at 200 ns). There are 10 codes of static RAM in the 3B20D Processor.

The programmable memories (PROM, EPROM) are used in the microstore and as program memory in microprocessor-based designs. In addition, smaller PROMs have been used in sequencer designs. In many designs these parts are used interchangeably with static RAMs. Again organization, speed, and power requirements are varied and result in 17 codes of programmable memories. The FIFO memories used in the 3B20D Processor are moderate-speed devices, primarily used for data buffering.

2.4 Large-scale integration logic

Because of the low cost per gate available with LSI logic its use was chosen wherever available parts could meet performance and functional requirements. The parts that were chosen provide large general-purpose functional blocks, such as microprocessors and associated peripherals, protocol controllers, dynamic RAM controllers, etc. Because of the relatively long development time and high development cost for LSI parts, new custom LSI designs were not undertaken during the initial 3B20D development stages. Cost reduction and new feature designs completed after the initial design phase have tended to use LSI gate arrays. There are a total of 30 catalog LSI codes and 16 LSI gate array codes in the 3B20D.

2.5 Circuit packs

The smallest replaceable module of the 3B20D is the circuit pack. Each pack consists of a collection of integrated circuits, in dual in-line packages, interconnected on circuit boards using the *Bellpac* packaging system (see Fig. 1). Typical circuit boards use a multilayer printed wiring board structure, which consists of six layers with plated through

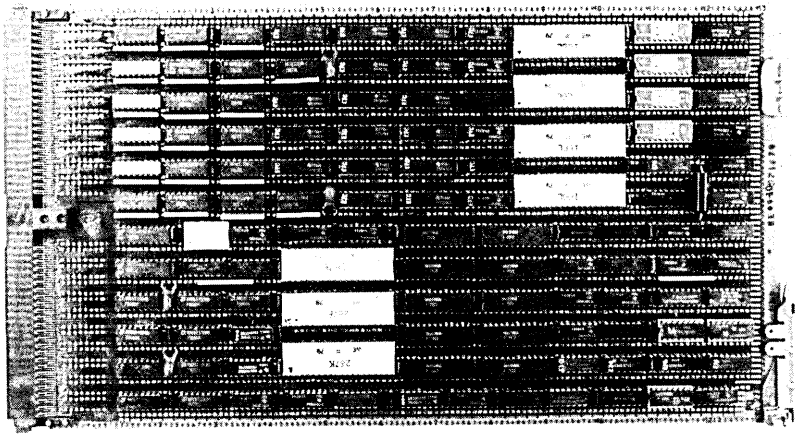


Fig. 1—The 3B20D Processor circuit pack.

holes to interconnect the layers. The structure of the printed circuit board is as follows: a component surface/signal layer, a signal layer, a power layer, a ground layer, a signal layer, and a wiring surface/signal layer. Board design used a standardized 0.100-inch grid to facilitate computer aided design and also to permit automated assembly and test. Protection for surface-layer printed wiring is provided by a cover coat that is applied prior to component assembly. The circuit pack multilayer board is normally 7.67 by 13.375 inches (although a few 7.67 x 9.375-inch boards are also used). All boards are 0.062 inch thick.

The circuit pack designs use either a 200-contact or 300-contact connector to interconnect the pack to a unit backplane. The connectors consist of a matrix of contacts contained in a plastic housing soldered to the edge of the printed circuit board. These contacts are bifurcated and selectively gold plated to provide a low-resistance, low-cost connection that will perform well over a 40-year design life. The circuit pack connector mates with a matrix of 0.025-inch-square pins mounted in the backplane on a 0.125-inch grid. The mating of the circuit pack contact with the backplane pin results in a loading of the bifurcated contact with a minimum normal contact force of 100 grams at the end of a 40-year life. The backplane pin is also selectively gold plated to provide a high-reliability, low-cost contact.

2.6 Frame unit

The next level of circuit packaging is the frame unit. Circuit packs mount into a frame unit that consists of a backplane printed wiring board with pins and apparatus mountings to support the packs. The backplane board is approximately 8.00 by 22.55 inches in size and

typically interconnects 21 circuit packs. Three apparatus mountings mount across the backplane to guide and support the circuit packs. The structure of the printed wiring backplane consists of six circuit layers that are interconnected with plated through holes. Pins are staked into the plated through holes on a 0.125-inch grid to match the circuit pack connector. These pins have low assembly cost and make highly reliable contact with the circuit boards.

The apparatus mounting provides circuit pack support and alignment with the backplane pins (Fig. 2). The apparatus mounting is designed to provide minimum impedance to air flow moving vertically through the unit. A designation strip is located at the top front of the mounting identifying the appropriate circuit packs. Matching the designation strip with the plastic faceplate of each circuit pack ensures proper installation.

The pins in the backplane extend on both sides of the printed wiring board. On the side of the backplane away from the circuit packs, additional connections between circuit packs can be made by wire, automatically wrapped to the pins. The majority of circuit pack

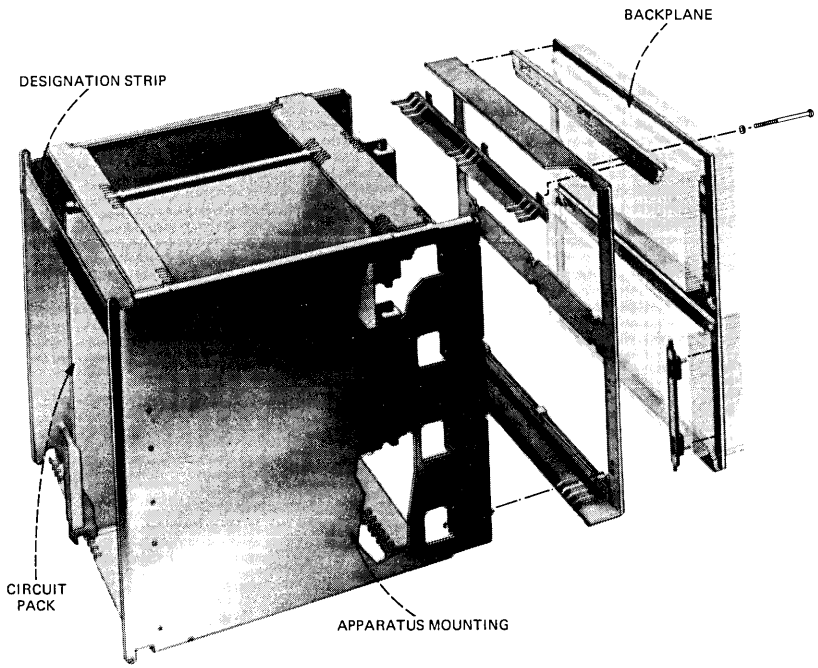


Fig. 2—The 3B20D Processor frame unit.

interconnections are made by signal paths within the backplane multilayer board; wires are used for changes or for connections that could not be routed in the printed circuit board. Also this side of the backplane is used for interconnecting different frame units or frames via connectorized cables. Many different connectors are used varying in size from 6 contacts to 24 contacts. This cable connector family supports both discrete wire (or switchboard) cable as well as multiconductor tape cable. The cable connectors are guided and retained on the backplane with special cable connector apparatus mountings that are attached to the backplane.

III. 3B20D UNIT PHYSICAL DESIGNS

3.1 Control unit frame

The 3B20D control unit frame is the core of the processor. Its modular equipment design permits the equipage of optional hardware features suitable to support a wide range of applications. The control unit frame in its maximum configuration contains the following units: central control, main store module 0 and 1, direct memory access input/output, cooling, and power. Figure 3 shows a maximally configured frame and denotes those units not required in a basic 3B20D Processor.

The 3B20D Processor frame units are mounted in a 7-foot-high, 24-inch-deep, 2-foot 2-inch-wide framework as shown in Fig. 3. Optional units and/or circuit packs are simply omitted where host system requirements do not require them. Subsequent paragraphs provide a description of the control unit frame units and their functions.

3.1.1 Central control unit

The central control unit is 10 inches high by 2 feet 2 inches wide. It contains the following required circuit functions:

(*i*) Central processing unit
(*ii*) Microstore
(*iii*) Main store update
(*iv*) Maintenance channel.

It has circuit pack positions allocated for the following optional features:

(*i*) Two input/output channel boards
(*ii*) Growth microstore
(*iii*) Cache
(*iv*) Utility circuit.

Also a circuit pack position has been reserved for the connection of a test set. The unit includes a fuse block, extending the full width of the unit, which provides individual fusing for each pack.[3]

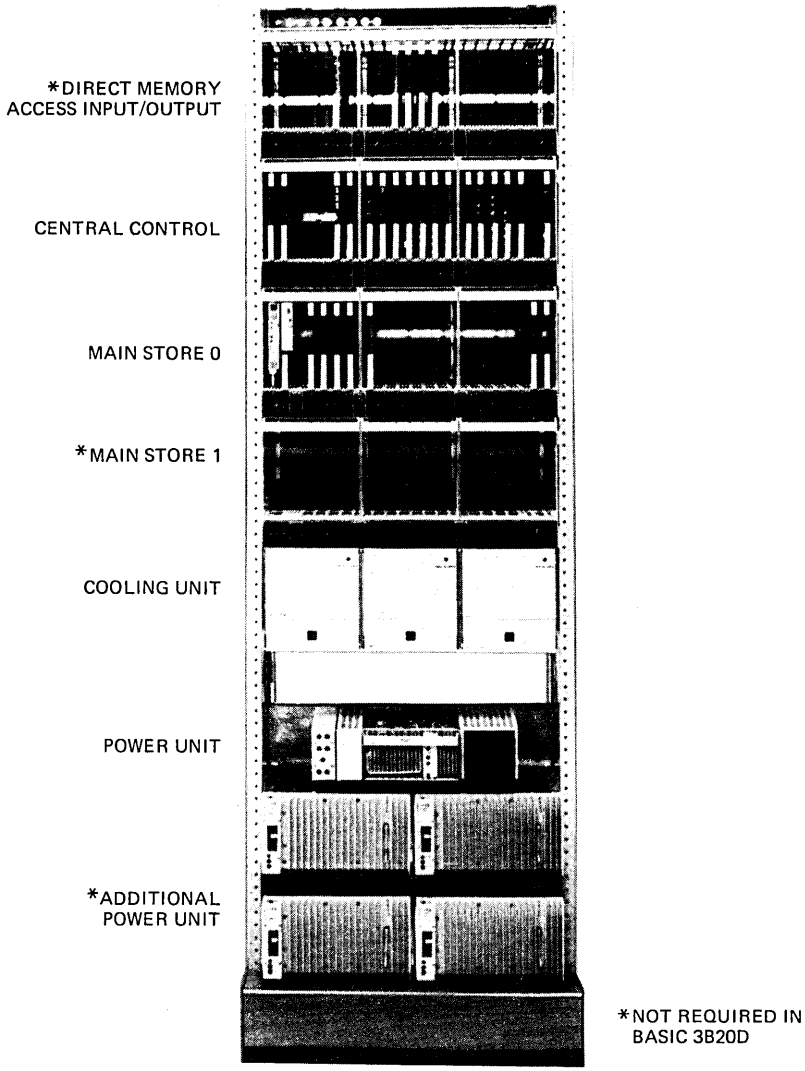


Fig. 3—The 3B20D Processor central control frame.

3.1.2 Main store unit

The main store units are each 10 inches high by 2 feet 2 inches wide and may be equipped with a maximum of 16 megabytes of memory. The unit is minimally equipped with store controller packs and with one array pack. Additional main store array packs can be added one at a time to form a maximum complement of 16 array packs per module. When a unit is using 512K-byte arrays, two main store units

may be used per frame to provide up to 16 megabytes of memory; when 1024K-byte arrays are used, only one memory unit is required for the maximum addressable 16 megabytes of memory.[4]

3.1.3 Direct memory access unit

The Direct Memory Access (DMA) unit is 10 inches high by 2 feet 2 inches wide and is used for expansion of I/O channel requirements beyond the two positions provided in the central control unit. It also houses the DMA controllers and their associated I/O channels.[3] A total of five positions are available for I/O expansion. I/O channels that can be equipped in these positions include the dual-serial channel, serial channel, and applications channel interface packs. There is room in the unit for two DMA controllers, although normally only one is equipped. Each DMA controller is equipped with a dual-serial channel with the option to add up to three additional dual-serial channel packs.

3.1.4 Cooling unit

The cooling unit is 8 inches high by 2 feet 2 inches wide and provides forced air cooling to keep device temperatures within specifications. The cooling unit is divided into three replaceable fan module assemblies, each with a fan and filter.

3.1.5 Frame power system

Power for the control frame is provided by the power unit located at the base of the frame. The power unit includes bulk power converters, a reference supply, and a power distribution/fusing system. Two to four −48 to +5 volts dc power converters are tied in parallel to provide power to all frame units via a frame and fuse panel bus bar system. The +12 volt power required by the devices used on the 512K-byte store arrays is supplied by a −48 to +12 volt dc converter located in the power unit. The power unit is also equipped with input fuses for each of the converters and for the reference supply. The power switch for the frame is implemented as a common design circuit pack which provides control and power sequencing for the frame and is located in the main store unit.

3.2 Peripheral control frame

The Peripheral Control Frame (PCF) houses those units which provide the interface communications between the 3B20D Processor and its periphery. This frame in its maximum configuration is equipped with an Input/Output Processor (IOP) basic unit, IOP growth unit, a Disk File Controller (DFC) unit, and a cooling unit. A port switch unit is also provided in one of the peripheral control frames. The frame is

a framework 7 feet high, 24 inches deep by 2 feet 2 inches wide. The following paragraphs describe the various units within this frame.

3.2.1 Input/output processor basic unit

The IOP basic unit is 16 inches high by 2 feet 2 inches wide and has a self-contained power system. This unit is subdivided into a processor section and two peripheral controller (PC) communities; each of the latter can be equipped with up to four PC circuit packs. The power switch circuit pack provides power control and sequencing functions to this unit and to the growth unit if present. DC to DC converters and a fuse system similar to the one used in the control frame complete the power system.[5]

3.2.2 Input/output processor growth unit

The IOP growth unit can be added to the basic unit to provide additional PC communities. It is 10 inches high by 2 feet 2 inches wide. This unit provides two communities of up to four PCs each to meet additional host system peripheral requirements. The self-contained power system for this unit is controlled by the power switch in the IOP basic unit.[5]

3.2.3 Disk file control unit

The DFC unit is 16 inches high by 2 feet 2 inches wide and includes a self-contained power system. The DFC is used to interface the 3B20D control unit with the disk memory system.[6] Control and data communications are provided for the moving head disk drives.

3.2.4 Cooling unit

The cooling unit used in the PCF is the same design as used in the control unit frame (see Section 3.1). This unit is located below the IOP basic or growth unit as appropriate to provide forced air cooling.

3.2.5 Port switch unit

The port switch unit is 4 inches high by 2 feet 2 inches. This self-powered unit provides one or two communities of up to three port switches each and also has positions for up to five scan and signal distribution interface circuits. One community is required for the port switching of the maintenance terminals.

IV. ELECTRICAL AND PHYSICAL DESIGN PROCESS

The electrical and physical design of the 3B20D Processor was based heavily on the application of a computer aided design process and the use of design standards and audits. Design standards were established and used to control the design process and to assure commonality

among separate design activities. The design audits were used to provide a predicted measure of system performance based on a given design and thus allowed feedback for making design changes to maximize system performance.

4.1 Circuit pack, unit, and frame design

The fundamental hardware building blocks (circuit packs, units and frames) were designed and tied together with the assistance of a CAD system. After the initial paper design, a circuit pack design file was created in the CAD system that contained all connectivity information. The design file represents the official record for the circuit. Circuit board manufacturing information, schematic drawings, design audits, and system connectivity can be created from these files. As the design progressed and various information was created, such as board routing, schematic representation, etc., it was added to the file. This circuit pack design file was also used to generate the software simulation files which were used to functionally verify the design and generate circuit pack test information.[7]

Fundamental to the CAD system were the design standards used throughout the 3B20D Processor. The design standards include a project library with integrated circuit and circuit board topology information, fixed power and ground pinouts for circuit packs, and common hardware definitions. The standards also provided specific rules for electrical design (fanout, crosstalk, etc.) that could be audited by the CAD system.

The interconnection of the different circuit packs within a unit was captured in a unit design file in the CAD system. These files contained all connectivity information and, like the circuit pack files, provided the basis for developing and storing detailed wiring information for the unit backplanes.

Frame drawings were a composite of design information from the unit files, with connectivity between the units added. Also included in the frame drawings was power distribution information.

4.2 Designing for system crosstalk and noise margin

The rapid switching transitions in TTL logic made noise and crosstalk performance a critical design criterion. For the system to operate successfully, noise had to be kept well below the switching thresholds of the integrated circuit devices used over a wide range of device temperature, supply voltage, and packaging variations. The large number of signal nets and the difficulty of tracking down subtle noise problems in a system as complex as the 3B20D dictated a set of design rules based on worst-case conditions. Comprehensive audits were made to assure compliance with these rules.

Two major noise audits were provided for the 3B20D. The first was a crosstalk audit (parallelism audit) that searched the circuit pack and backplane routing files to identify for each net all signal nets immediately adjacent and parallel to the net of interest and sum the exposures. Through theoretical calculations and experimental results a maximum safe exposure limit was established for each device type. All nets exceeding this crosstalk exposure limit were referred for further analysis to the circuit designer. The designer's analysis would determine whether rerouting was required to assure that system performance was not adversely affected. The second major noise audit was for current shuttling. Analysis for this audit is a complex function of the device types and detailed routing information. Using a device parameter library and the circuit pack routing files, the current shuttling audit generates a list of all nets with potential noise problems. These nets were then reviewed by the circuit designer.

A fanout audit also was provided. This audit is based on detailed device drive and loading information in the device library and on the circuit pack and unit connectivity files. Violations of the fanout rules can result in reduced noise margins, or in extreme cases, malfunction of the logic.

4.3 Timing analysis

Timing analysis consisted of computing the minimum and maximum delay for each device in a path of interest under actual load conditions. The delay consists of an intrinsic component through the device itself and an interconnection component that depends on the wiring parameters, fanout, and the output drive capabilities. CAD programs and libraries were developed to extract the interconnection data to compute the device and path delays.

4.4 Thermal analysis

Computer aided tools also were used for thermal analysis. In particular they were used to predict device temperatures on certain heat-sensitive devices in the frame environment. The tools used an experimentally determined heat transfer coefficient and integrated circuit power dissipations to calculate device temperatures. In addition to confirming the adequacies of the design, these tools were used to specify temperatures for selected heat sensitive devices where device cost, performance, and temperature were important trade-offs.

V. FLEXIBLE FLOOR PLAN

Because the 3B20D Processor was designed as a cost-effective processor for a wide range of Bell System applications, it usually will be installed as a group of frames within the host system environment; in

other cases, such as retrofit environments, the processor frames may have to fit into the space available. The 3B20D has been designed to support both of these situations. From an environmental standpoint, no special air-conditioning diffusers are required for the 3B20D equipment. However, sufficient cooling and ventilation must be provided so that the aisle air temperature does not exceed 37°C on a long term basis or 49°C for a short term.

The recommended typical floor plan for the 3B20D is shown in Fig. 4. This plan depicts a basic processor with its most likely growth patterns and satisfies most system applications. Moving head disk cables and associated duct work have been designed to support the typical plan. Other floor plans are permitted providing cable length restrictions are observed. These cases may require special cables and/or duct work which would have to be engineered as part of the host system.

VI. TECHNOLOGY PERFORMANCE

6.1 Rapid development

To assure a rapid development of the 3B20D Processor, standardized hardware components and special prototype hardware were used. The basic *Bellpac* packaging system hardware building blocks were used wherever possible to reduce availability intervals. Quick-turnaround wire-wrap boards were used for prototype circuit packs and backplanes. The use of a CAD system allowed the transition from wire-wrap to multilayer design to be a minimal design effort. The net result

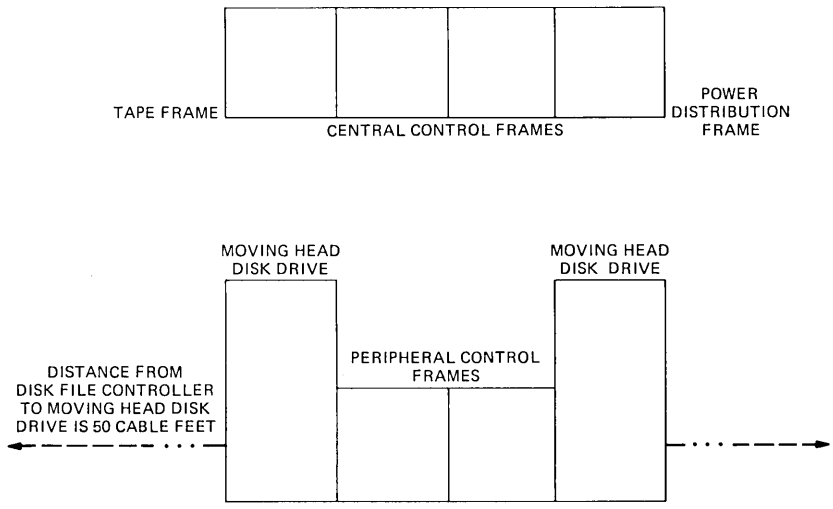


Fig. 4—Recommended floor plan for the 3B20D Processor.

of the use of off-the-shelf hardware, computer aided designs, and wire wrap models was that the objective project development schedule was achieved.

6.2 System electrical performance

The electrical performance of a system is best measured by the successful achievement of system performance objectives and the lack of timing and noise problems. Using the technology system described, all 3B20D performance objectives have been met. Further, despite the high degree of complexity, few noise and timing problems were encountered in the actual hardware. This successful result was achieved by the coordinated and comprehensive set of device and packaging specifications and design aids and audits.

6.3 Thermal performance

The 3B20D Processor is designed to operate in a normal environment of less than 37°C and at a maximum elevation of 6000 feet above sea level. In the event of a building cooling failure, the processor will remain operable in air temperatures of up to 49°C. During these building cooling failures, the circuit board temperatures do not exceed the design goal of 95°C in the control frame and DFC. In the IOP units there are a number of critical LSI devices, which are kept below 70°C to maintain good timing margins. All peripheral equipment (disk drives, tape drives, printers, and terminals) is tested by the respective vendor for operation up to 49°C.

Final thermal acceptability of each 3B20D system to be shipped is verified in a factory heat test where each processor system, with peripherals, is heated to 49°C for a 6-hour functional test.

VII. ACKNOWLEDGMENT

This paper comprises the work of many individuals, in addition to the authors. Special thanks for their participation is given to Messrs. H. A. Devine, S. C. Zemke, and C. E. Miller.

REFERENCES

1. W. L. Harrod and A. G. Lubowe, "The BELLPAC Modular Electronic Packaging System," B.S.T.J., *58*, No. 10 (December 1979), pp. 2271–88.
2. R. C. Hansen, R. W. Peterson, and N. O. Whittington, "Fault Detection and Recovery," BSTJ, this issue.
3. M. W. Rolund, J. T. Beckett, and D. A. Harms, "3B20D Central Processing Unit," B.S.T.J., this issue.
4. I. K. Hetherington and P. Kusulas, "3B20D Memory Systems," B.S.T.J., this issue.
5. A. H. Budlong and F. W. Wendland, "3B20D Input/Output System," B.S.T.J., this issue.
6. R. E. Haglund and L. D. Peterson, "3B20D File Memory System," B.S.T.J., this issue.
7. "LAMP: Logic Analyzer for Maintenance Planning." B.S.T.J., *53*, No. 8 (October 1974).

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

3B20D File Memory Systems

By R. E. Haglund and L. D. Peterson

(Manuscript received March 10, 1982)

The 3B20D file memory system employs microprocessor-based intelligent Disk File Controllers (DFC), each supporting a complement of one to eight 300-megabyte disk drives. The DFC communicates with the system via a high-capacity dual-serial channel link to the Direct Memory Access Controller (DMAC) that accesses the processor's main memory. The DFC communicates with its disk files via an industry standard interface. The disk power system is also microprocessor based. The system operates disk drives on commercial ac power until a power failure occurs. The disks are then switched under microprocessor control to inverter power until the commercial ac fault is cleared, at which time the disk is returned to commercial ac power.

I. INTRODUCTION

The 3B20D file store consists of a microprocessor based Disk File Controller (DFC) plus one to eight 300-megabyte disk files. The files are powered by a microprocessor-based power system consisting of an inverter, a cycloconverter and a static switch. The DFC communicates via the 3B20D Dual-Serial Channel (DSCH) and through the Direct Memory Access Controller (DMAC) to the processor's main memory (see Ref 1). The DFC communicates with its disk files via a disk interface that utilizes the industry standard Storage Module Drive (SMD) interface.

The DFC provides for stand-alone processing of disk accesses via approximately 30,000 bytes of code stored in its PROM memory. Also stored in the PROM memory are 30,000 bytes of diagnostic code.[2]

The DFC uses a bit-sliced bipolar microprocessor to manage the 10-

MHz serial data streams to/from the disk files, the disk control functions, and the system interface. In addition, the processor runs a disk exerciser program to test disk drives during periods when no system accesses are in progress.

II. SUBSYSTEM COMPONENT DESCRIPTION

The disk file memory system for the 3B20D Processor consists of three major elements: a DFC, 300M-byte moving head disk memories, and a 208 Vac power-control system for the disk memory units. The power system consists of a Power Control Unit (PCU) and an emergency -48 Vdc to 208 Vac Disk File Inverter (DFI).

2.1 The 300-megabyte disk drive

The disk drives used in 3B20D Processor are high-speed, random-access, data-storage devices (Fig. 1) supplied by Century Data Systems and the Control Data Corporation. The disk drive unit consists of two physical parts. The basic drive assembly contains the chassis, power supply, air-filtration system, read/write head assembly, control circuit cards, and the input/output interface. The second part is the removable disk pack upon which data is recorded.

The disk pack contains 12 stacked platters. The inner ten are coated on both sides with the recording medium, ferrous oxide. The top and bottom platters are only for disk pack protection. The remaining ten platters have a total of 20 recording surfaces, of which 19 are used for recording data while the other contains prerecorded servo information that is used for head positioning and timing. Each disk pack is capable of storing 300M bytes (unformatted) of information. Each read/write head may be positioned over 815 different cylinders. The 19 read/write heads plus the servo head are mechanically attached together on a single carriage arm. Therefore, all heads are moved in parallel such that all heads are always positioned over the same cylinder. By selecting one of the 19 data surfaces, an individual track is selected. Each track is divided into 32 sectors or blocks of 512 bytes each. A sector is the smallest quantity of data written or read by a disk file controller.

Communication to and from the drive is via the industry-standard SMD type of interface. Two cables interconnect the drive to the disk file controller. One contains the control and status signal lines while the other contains the read and write data signal lines. The disk drive is powered from 208-volt single-phase power. Starting current is a maximum of 50 amperes, with running current of approximately 9 amperes. The disk pack is rotated at approximately 3600 RPM; thus the average rotational latency is 8.5 ms. The average head positioning

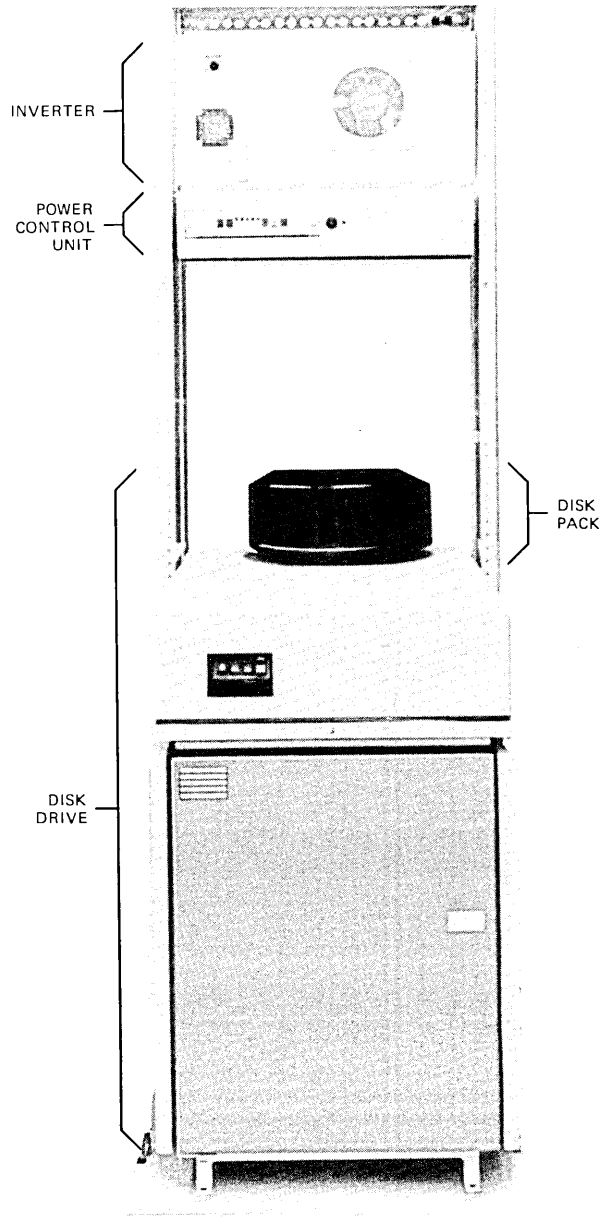


Fig. 1—Moving head disk frame.

(seek) time is 30 ms. Data is transferred to and from the disk drive at a 9.67-MHz serial data rate.

The disk units are designed to meet Bell System environmental requirements. Acoustical noise generated by a disk unit is less than 65

dBA. RFI gasketing material was used extensively to limit electromagnetic radiation from the units and reduce the susceptibility of the units to external electromagnetic radiation sources.

2.2 Disk power control and backup power system

The PCU and the −48 Vdc to 208 Vac DFI are used in combination to supply uninterruptable power to a single 300M-byte disk. The units interconnect with each other and the disk as shown in Fig. 2.

Inputs to the PCU are −48 Vdc, two sources of 208 Vac, and control signals from the DFI. Power for the logic circuitry is derived from the −48 Vdc. One source of the 208 Vac is from the commercial power grid or the essential ac grid in a central office. The other source of 208 Vac is from the DFI. Status and control functions also are received by the PCU from the DFI.

Cooling for the DFI is provided by two fans. The fans are active only when the unit is supplying 208 Vac. The power dissipated by the DFI under a load condition is 500 watts when supplying 2000 watts to the disk. Normally power to the disk is supplied by the commercial or essential ac. Under this condition, the inverter dissipates no more than 150 watts.

Even though nominally continuous, commercial power is subject to minor interruptions of short duration. Since the disk pack has substantial inertia it is possible to power the motor from the commercial ac mains despite such interruptions. The disk drives used in the 3B20D Processor have ferroresonant supplies for their own internal dc power, and these supplies are tolerant of minor interruptions in input voltage.

The 3B20D disk power system takes advantage of these characteristics to deliver power to the disk drive from the most efficient source.

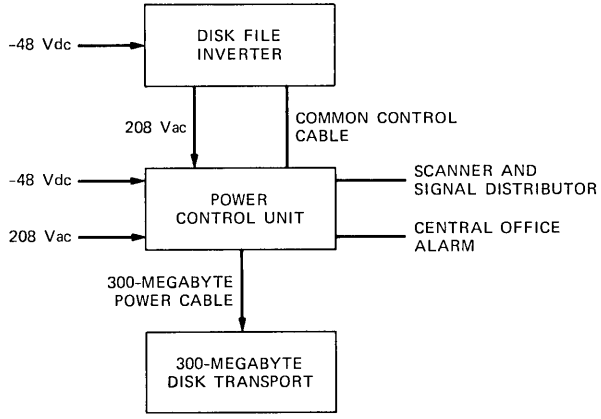


Fig. 2—Interconnections between the control panel, inverter, and disk transport cabling units.

The DFI powered from the battery plant supplies the power for the disk drive when the ac mains are not available or out of tolerance in voltage or frequency. A silicon controlled rectifier static switch provides the changeover within two cycles of the time the ac mains are found to be out of tolerance. The inverter is phase locked to the ac mains voltage when it is present so that the transient due to the switchover to battery power is minimized. The operation of the disk drive is unaffected by the switchover between ac and inverter power.

The standby inverter (Fig. 3) operates at 600 Hz; its output is converted to the required 60 Hz ac by means of a tap-changing cycloconverter. This provides a piecewise approximation to the desired 60-Hz sine wave (Fig. 4). The inverter frequency was chosen high enough to allow the use of a small, efficient inverter transformer and also to allow a sufficient number of intervals in which the output voltage could be selected to make a good approximation to a sine wave. The cycloconverter output has the same peak and rms values and same average value over a half cycle as the sine wave it replaces. The third and fifth harmonics are also reduced to small values by appropriate choice of cycloconverter parameters.

The disk power system is controlled by a microprocessor housed in the inverter. This processor administers the operation of the inverter, cycloconverter, and static switch so that the proper power source is connected to the disk drive. The choice of power source is made on

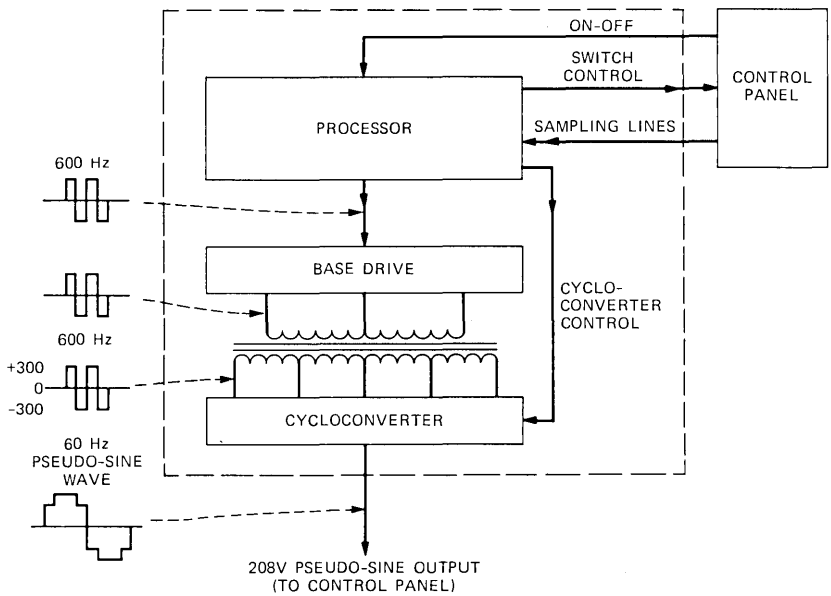


Fig. 3—Disk inverter operation.

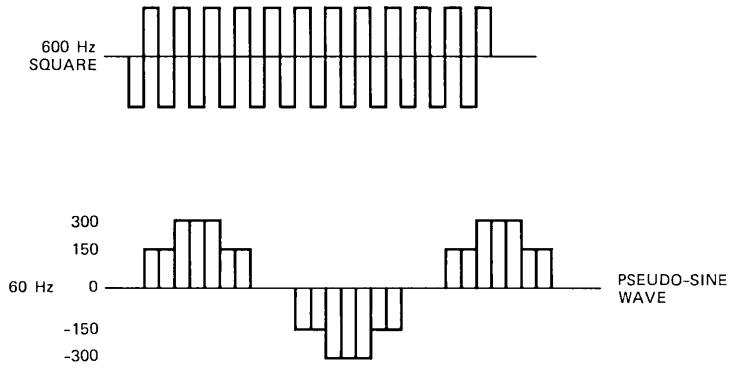


Fig. 4—Pseudo-sine wave generation.

the basis of sampling the ac mains for voltage above the programmed threshold value ten times per half cycle. If the mains voltage supply is found to be deficient for two samples in any half cycle, the static switch disconnects the ac mains and the cycloconverter supplies power to the disk drive. When the mains voltage has been determined to be within acceptable limits in voltage and frequency for a sufficient time, the cycloconverter output is inhibited and the static switch again allows the mains to power the disk drive.

While operating from the battery plant, the inverter regulates its output voltage to compensate for battery variations. A reflex regulator circuit uses some of the 600-Hz inverter output voltage in a phase-controlled rectifier to control the apparent input voltage to the inverter to maintain it constant at −52.2 volts in the face of battery voltages in the range of −42.75 to −52.5 volts. The microprocessor controls the operation of the regulator so that the peak current demands on the inverter transistors are not increased by the operation of the regulator. The regulator operation is inhibited during the times the peak output current is delivered to the load, and regulator operation is only allowed when the 60-Hz output is in the part of the cycle where the output voltage, and therefore the inverter load current, is near zero.

Since it may be necessary to change the disk pack while ac power is not available, the inverter must be able to start the drive motor. The current required to start the disk drive motor is much larger than the current required to run the motor. Since it would be uneconomical to size the inverter to carry the starting load, the microprocessor controls the cycloconverter output voltage during the time the motor is starting so that the starting current is greatly reduced. This power reduction is done at the expense of starting time; the time required to start the drive motor is 45 seconds on battery and 15 seconds on the commercial mains.

With the inverter always powered up even while the disk is being

powered by commercial ac, the possibility exists that the power-handling components could suffer a failure that would be undetected until the inverter was required to supply power. A diagnostic triggered by a disk being out of service is available to detect this failure. The microprocessor continually monitors the state of the power switch to detect when the disk has been taken out of service. When this condition is detected, the microprocessor starts the diagnostic routine, which causes the inverter to supply power to the disk drive for a few minutes while monitoring the cycloconverter output voltage for the correct levels. Thus, any failure of power-handling components will be detected during the regular daily diagnostic exercise of the disk frame.

2.3 Moving head disk frame

The Moving Head Disk (MHD) frame (Fig. 1) is the frame configuration used to mount a disk, PCU and DFI.

The PCU and DFI share a common interface cable for transmitting status and control information. The DFI 208-Vac output is connected to the PCU via a flexible conduit. The disk unit is in turn connected to the PCU via a flexible power cord. The −48 Vdc input for the PCU is connectorized at the top of the frame. However, the DFI −48 Vdc input is routed directly to a power-distribution frame due to the wire gauge and currents involved.

The disk drive is rolled onto and secured to a metal base plate fastened in the base of the MHD frame. The drive protrudes out the front of the frame 18 inches. The base plate is hinged and is raised or lowered by jackscrews on both sides of the base plate.

The I/O cables between disks and the DFC are routed in a rectangular duct at the rear of the MHD frame. The duct serves to provide additional electromagnetic shielding and physical protection for the cables. In multiple, adjacent MHD frame applications, the cable ducts form a continuous trough for the I/O cables.

2.4 Disk growth and floor plan

The MHD frame was designed to provide for growth in a modular fashion. Depicted in Fig. 5 is a typical floor plan for the 3B20D Processor. The Peripheral Control (PC) frame is situated between two sets of MHD frames. Each PC frame contains a DFC (Fig. 6).

The total cumulative control cable bus length between a DFC and all disks controlled by the DFC is 100 cable feet. Thus, all MHDs controlled by a particular DFC are placed adjacent to each other so as not to exceed the 100-foot limit.

2.5 3B20D disk file controller

The 3B20D Disk File Controller (DFC) is a high-performance, microprogrammed processor whose design provides efficient control of

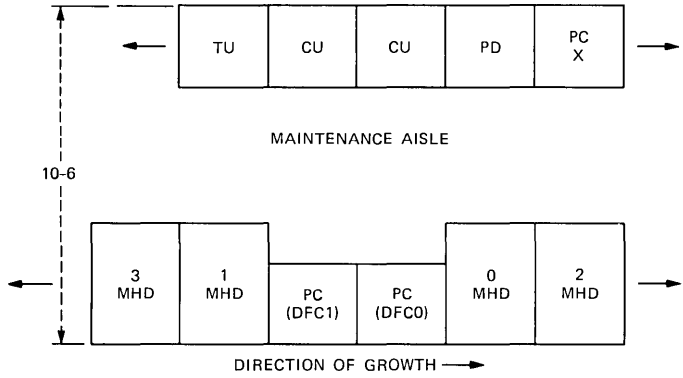


Fig. 5—Typical floor plan of the 3B20D Processor.

MHD drives and data transfers between the 3B20D Control Unit (CU) and MHD units. Each DFC supports a maximum of eight MHD units and utilizes a standard 3B20D input/output peripheral interface. The following sections discuss the hardware architecture and functionality of the DFC.

2.5.1 3B20D DFC architecture

2.5.1.1 DFC communications bus. A DFC with its associated MHD units—as shown in Fig. 7—consists of a processor, MHD interface, and the two circuit boards that provide the interface to the 3B20D Processor. These are linked together via a common communications bus.

The bus, which is dc coupled, provides 16 data bits plus 2 parity bits, 5 bits for source address, 5 bits for destination address, and 1 synchronization clock line. The source/destination addresses are binary encoded allowing a maximum of 32 unique hardware registers to be addressed on the bus. The DFC's processor and the 3B20D Processor interface each utilize 8 address codes, and the remaining 16 codes are assigned to the MHD interface. Since the bus provides unique source and destination address buses, only one bus cycle, which is defined by the synchronizing clock, is required to transfer a 16-bit data word from one hardware register to another. All data transfers on this bus are initiated and controlled by the DFC's processor. The period of the synchronizing clock may be 150, 200, 250, or 300 ns and is dependent on the instruction being executed by the processor, the capabilities of the hardware, and the timing requirements of specific firmware routines.

2.5.1.2 DFC/MHD interface. Each DFC will support a maximum of eight MHD units equipped with an SMD interface. As depicted in Fig. 7, each MHD is connected to the DFC via two cables, one of which is common to all MHDs and is daisy chained from one unit to the next.

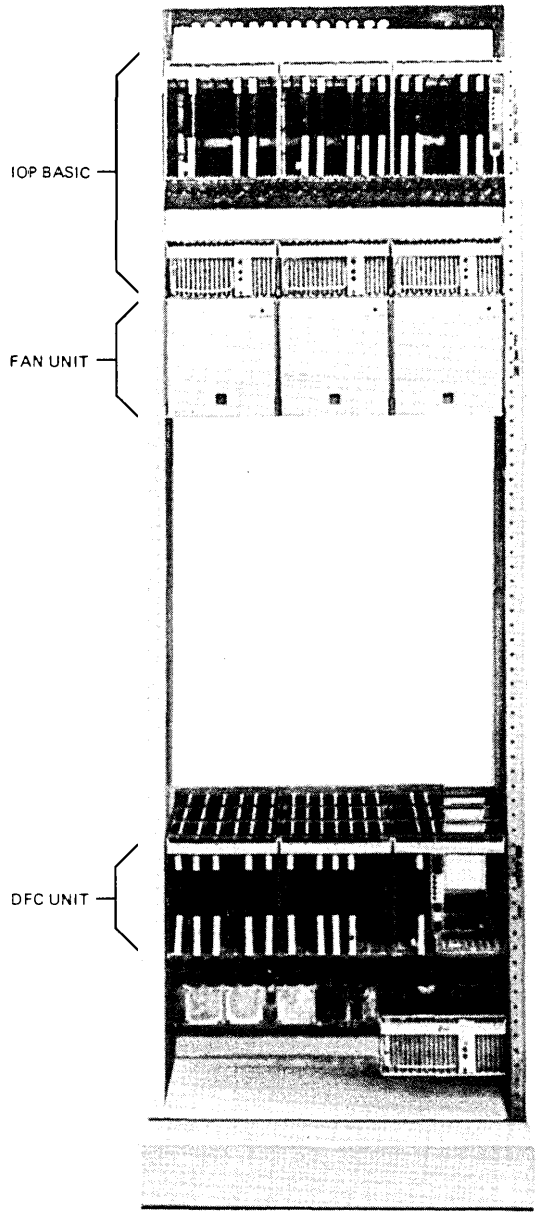


Fig. 6—Peripheral control frame.

This cable provides a communication path between the DFC and MHDs for command and status information. The transfer of data between the MHDs and the DFC occurs serially over a private path provided by the other cable. The data transferred is in a Non-Return

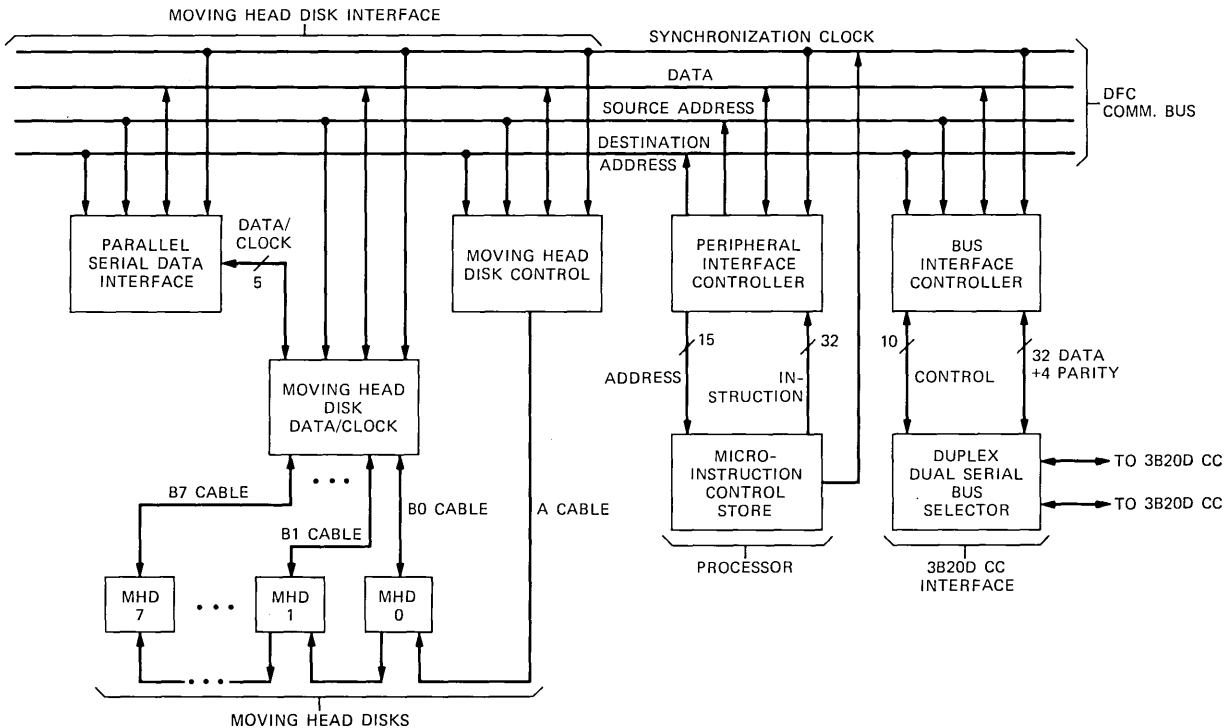


Fig. 7—Disk system block diagram.

to Zero (NRZ) format and a synchronous bit clock accompanies the data to simplify the circuit requirements within the DFC and MHD.

Because the MHD units may be located up to 100 cable feet from the DFC, signaling paths in both cables are implemented as twisted pairs, and both the DFC and MHDs interface with the cable using differential line drivers and receivers.

2.5.1.3 DFC/3B20D control unit interface. The DFC communicates with the 3B20D CU via a DSCH interface consisting of the Duplex Dual Serial Bus Selector(DDSBS) in the DFC and DSCH in the 3B20D CU. The communications path between the DDSBS and the DSCH consists of five twisted-pair dc-coupled signaling paths. The pairs are divided into two bidirectional data pairs, two unidirectional clock pairs, and a single pair used to transmit service requests to the DMAC and interrupt requests to the 3B20D CU.

Using the clock and data pairs, the minimum amount of information that is transferred is 32 data plus 4 parity bits. The data is transmitted serially on the two data pairs (16 data plus 2 parity on each pair) in an NRZ format. A synchronous bit clock is transmitted on the appropriate clock pair by either the DDSBS (DFC to CU transfer) or the DSCH (CU to DFC transfer). The interface does support a block transfer mode where 16 words of 32 bits may be transferred as a single block between the DFC and CU. This mode of operation is useful in minimizing the overhead associated with transferring large blocks of data between the DFC and CU.

The DFC's processor does not access the DDSBS directly, but rather has read/write access to several registers in the Bus Interface Controller (BIC). The BIC functions as a buffer between the DDSBS (32 bits) and the DFC's processor (16 bits). The BIC buffers data and commands for the processor and performs the necessary handshaking to communicate with the DDSBS.

2.5.2 DFC processor

The DFC's processor consists of the Peripheral Interface Controller (PIC) and its associated Microinstruction Control Store (MCS). The PIC is a 16-bit, bit-sliced microprocessor that is capable of performing all the common arithmetic, logic, and sequencer flow-control operations found in many 16-bit minicomputers. The PIC contains a microprogram data register, Arithmetic Logic Unit (ALU), 8K bytes of RAM that serve as a temporary data store, vectored interrupt control, and microprogram address generator.

The MCS stores instructions that the PIC executes in Programmable Read-Only Memory (PROM) devices. Each MCS circuit pack has the capability of storing a maximum of 4K microinstructions. A total of three MCS circuit packs are utilized. In addition to functioning as a

store, the MCS also generates the synchronizing clock for the PIC and DFC communication bus based on bits stored with each microinstruction.

On the rising edge of each synchronizing clock signal, the PIC generates a new microprogram address that is sent to the MCS. The MCS provides the PIC with the instruction stored at the requested address before the next synchronizing clock, at which time the PIC loads the instruction in its microprogram data register and sends a new address to the MCS. The PIC then executes the registered instruction while the next succeeding instruction is being fetched from the MCS. This pipelined operation that is provided by the microprogram data register allows a higher instruction execution rate than would otherwise be possible and contributes to a higher level of DFC performance.

Each PIC microinstruction is 40 bits wide and is divided into several fields as shown in Fig. 8. The PIC registers bits 0 through 31 in its microprogram data register while bits 32 through 35 are used by the MCS to verify parity on each instruction that is fetched, and bits 36 through 39 determine the period of the synchronizing clock generated by the MCS. Bits 0–4 and 5–9 are the source and destination address bus fields, and since these bit fields are dedicated, each PIC instruction specifies the movement of data from one register to another. Bit 12, when set, disables all hardware interrupts. The capability to disable interrupts on a per-instruction basis is required for the proper functioning of instructions that implement "JUMPS" and also allows the programmer to construct code segments that are critical from a real-time viewpoint and cannot be interrupted. Bits 10 and 11 determine how the overlay field, bits 13 through 31, are to be interpreted by the PIC.

2.5.3 MHD interface

The MHD interface consists of three circuit packs that allow the PIC to communicate with its MHD units. These circuit packs are the MHD Control (MHDC), MHD Data/Clock (MHDDC), and Parallel Serial Data Interface (PSDI).

2.5.3.1 MHD control. The MHDC consists of four registers, varying in width, whose outputs drive the control bus of an MHD with an SMD interface. These registers may be loaded and read by the PIC via the internal DFC communication bus. In addition to these registers, the

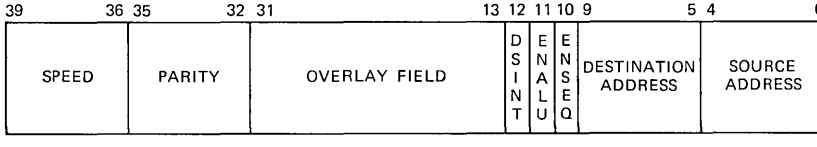


Fig. 8—Microinstruction of the peripheral interface controller.

circuit pack also provides a port that allows the PIC to sample 8 bits of status information from the MHD that is currently selected by the DFC. The output of each register drives the input of the differential line driver whose outputs drive the control cable to the MHD units. Likewise, the 8 bits of MHD status are driven onto the cable by the selected MHD unit and differential line receivers in the MHDC pack translate the differential voltage to a TTL level.

The device-enable register is a single bit register that is reset by the DFC during a power-up sequence, thus disabling the interface between the DFC and MHDs. The PIC writes this register to a one, enabling the interface, after executing its power-up microcode.

The unit-select register is a 4-bit register that contains the address of the MHD unit communicating with the PIC.

The data register is 11-bits wide and its outputs form the data portion of the control cable. This register is used with the disk tag register to transfer command information to the MHD unit consisting of cylinder number, read/write head number, and read/write/initialization commands.

The tag register's outputs form strobe signals that cause the MHD units to use information present on the data portion of the control cable. For example, bit 3 going active causes all MHDs on the control cable to compare the state of the unit-select bits with the MHD's address code. If a match occurs, then that MHD becomes selected and honors commands sent using bits 0 through 2 of the tag register. Since each MHD connected to a DFC has a unique address code, only one MHD may be actively communicating with the DFC at a given time. time.

2.5.3.2 MHD data/clock. The MHDDC interfaces the serial data and clock between the MHDs and the PSDI, and provides the PIC access to a select acknowledge signal generated by each MHD. The select acknowledge signal is made active by an MHD whenever it is selected, and by providing one acknowledge signal per MHD allows the PIC to verify that the correct disk volume has been selected for use. With respect to the serial data and clock signals, the MHDDC functions as a voltage level translator (differential signaling to TTL and vice versa) and signal multiplexer, routing the data/clock signals between the selected MHD and the PSDI.

2.5.3.3 Parallel serial data interface. The PSDI allows the PIC to transmit data to and receive data from the MHD. A block diagram of the PSDI is shown in Fig. 9.

Data exchange between a DFC and an MHD drive takes place over a 10-MHz serial data path. During a disk write, the PSDI performs parallel to serial data conversion, checks data parity as the data is being shifted out serially, and computes a 32-bit cyclic ECC that is

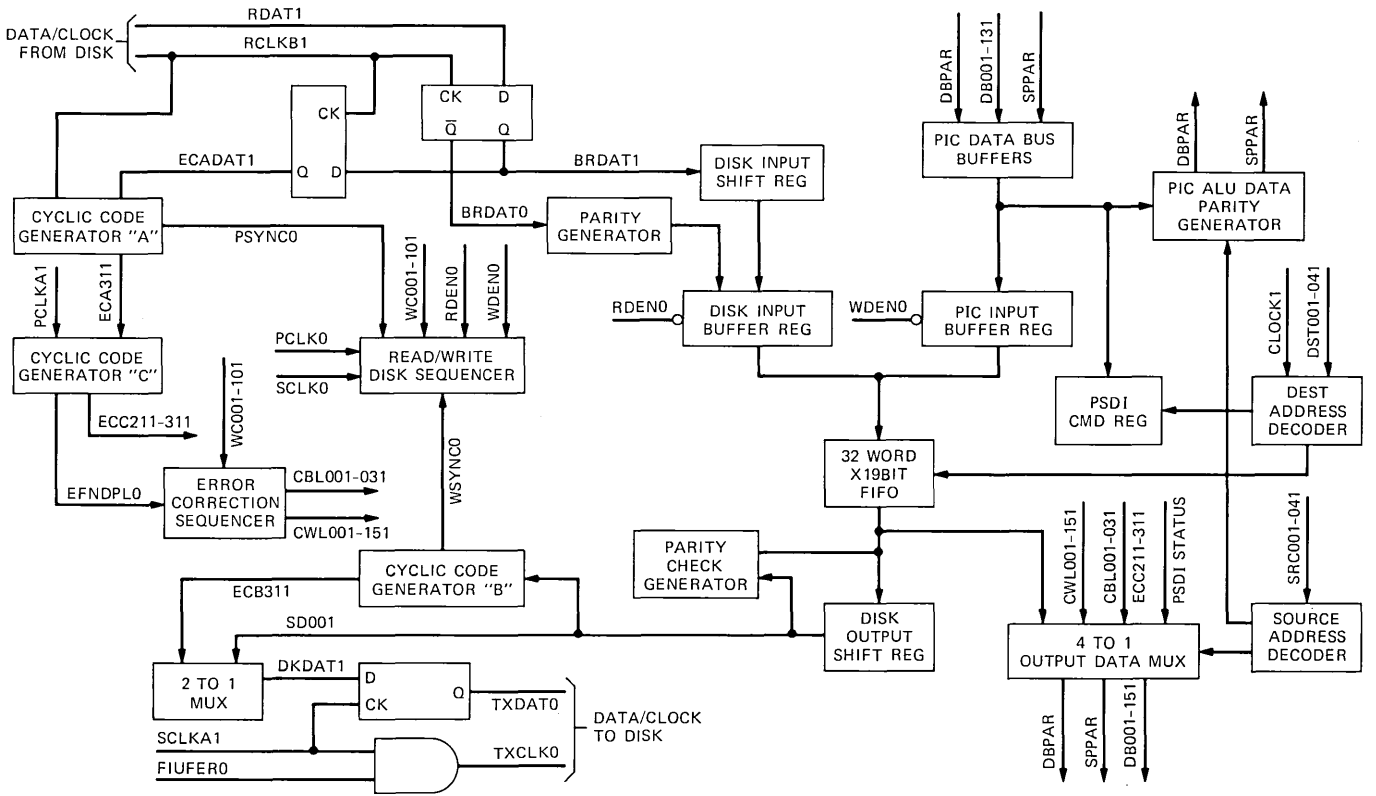


Fig. 9—Parallel/serial data interface.

also transmitted to the disk after a predefined block of data has been transmitted. During a disk read, the PSDI performs serial-to-parallel data conversion, computes data byte parity, and evaluates the ECC read from disk to determine if any errors are present in the data. While the PSDI itself does not perform the error-correction function, it does provide the PIC with the following information about each data block read from the disk: (*i*) The presence of errors, (*ii*) whether the error is correctable or uncorrectable, (*iii*) the location of the error relative to the start of the block, and (*iv*) an 11-bit correction mask that is to be exclusive ORed with the received data to affect the correction.

2.6 Disk read and write operations

2.6.1 Disk read operation

The PSDI functions as a serial data interface between the PIC and an MHD drive. During disk reads serial data and clock transmitted by the disk drive are assembled into 16-bit words in the disk input shift register, and stored in the data FIFO. Simultaneously, the serial data is also presented to the input of a parity generator where data byte parity is assigned and stored in the FIFO together with the data, and to cyclic code generator A where a 32-bit code word is computed to be utilized during the error correction search process. In addition, this cyclic code generator searches the incoming data stream for the presence of data synchronization bits. The generator searches the synchronization characters following the preamble to locate the beginning of the address/data fields in the serial data stream. Once sync is found, the read/write disk sequencer begins assembling 16-bit words and storing them in the data FIFO. The only information stored in the FIFO is the address and data bytes. Once the ECC field from disk has been input to the code generator, the contents are serially transmitted to a second code generator C. Then, to start error correction generation, the first code generator begins its search for synchronization characters again.

Cyclic code generator C and the error correction sequencer function together to determine the presence and location of an error in the received serial data stream. Code generator C is a 32-bit shift register which has several feedback paths around it. The error correction sequencer supplies shift pulses to generator C and if after each shift the contents of the low-order 21 bits of generator C are zero, the correction sequencer stops and indicates that a correctable error has been found. If they are nonzero and both the bit and word location counters have not reached their maximum counts, the sequencer will generate another shift pulse. If the 21 bits remain nonzero and both the bit and word location counters have reached their maximum

counts, the sequencer stops and indicates that an uncorrectable error is present in the data stream.

An uncorrectable data error is one where the first and last data bits in error are separated by more than nine consecutive bits. A correctable data error is one where the first and last data bits in error are separated by nine or less consecutive bits. The first case of no error is distinguished from the error case by the value of the upper-order 11 bits for code generator C. With no bits in error, these 11 bits will all be zero.

As mentioned earlier, the PSDI does not perform the error correction function, but does provide the PIC access to the information necessary to perform this task. The correction mask code generator C high-order 11 bits, the bit location counter, word location counter, and correction sequencer status (correctable or uncorrectable data error) are made available to the PIC through the output data multiplexer whose outputs are the PIC data bus. The error correction code contained in each section of the disk is utilized together with code generators A and C, and the error correction sequencer are designed to detect and locate errors in the address, data, and the error correction code fields recorded on disk.

2.6.2 Disk write operation

When writing the disk, information contained in the FIFO is loaded into the disk output shift register where it is serialized and transmitted to the disk drive under the control of the read/write disk sequencer. Simultaneously, the serial data stream is applied to the input of the parity check generator and cyclic code generator B. Although each data word in the FIFO has two parity bits associated with it, they are not transmitted to the disk, but are loaded into the parity check generator each time the output shift register is loaded from FIFO. In addition to the parity bits, each data word also has associated with it a parity check control bit, which is loaded into the parity check generator during the FIFO to output shift register transfer. The parity check control bit either enables or disables the parity generator on a per-word basis allowing words with known bad parity to be transmitted without causing parity errors.

Cyclic code generator B computes the 32-bit ECC field to be written on disk based on information transmitted in the address and data fields. Since writing a sector on disk requires all fields shown in Fig. 6 to be written, code generator B also searches for the synchronization characters. When sync is found, the read/write sequencer configures code generator B to operate as a feedback shift register to compute the ECC field for the address and data fields that follow the sync characters. In addition, at sync time, the read/write sequencer begins counting the words it loads into the output shift register from the

FIFO. After the last data word has been serially transmitted from the output shift register, the read/write sequencer disables all the code generator B feedback paths, enables the contents of the generator to be shifted out to the disk, and then redirects the serial stream back to the FIFO/output shift register to transmit the post-amble characters. Once the ECC field is transmitted, both the read/write sequencer and code generator B revert back to their initial states searching for synchronization characters.

For diagnostic purposes it is useful to be able to write an invalid ECC field when writing a sector on disk. Setting a bit in the PSDI command/status register prevents the read/write sequencer from transmitting the contents of code generator B to the disk. Instead, the read/write sequencer continues to transmit information to the disk contained in the FIFO. This scheme allows any two 16-bit words to be written in the ECC field in place of the normally computed value provided by code generator B.

2.6.3 Data FIFO

A 32-word FIFO memory is loaded by the read/write sequencer and read by the PIC during a disk read operation, with the reverse being true during a disk write operation. Each word in the FIFO is 19 bits wide: 16 data, 2 parity, and 1 parity check enable bit. As explained earlier the parity check enable bit is significant only during a disk write operation. FIFO write access is controlled by the mode bit in the PSDI status register. Depending on the state of this mode bit, data is written into the FIFO by the read/write sequencer or by the PIC. Because the FIFO requires that valid data be present for 70 ns after the write pulse makes its low-to-high transition, two input data buffer registers are provided for use by the read/write sequencer and PIC.

Associated with the FIFO are four status flags. The FIFO ready and FIFO half full/empty flags provide the PIC information to load or unload the FIFO properly and efficiently. When the PSDI is in the read mode, FIFO ready low/high controls whether the PIC may or may not read a word from the FIFO. Likewise, when FIFO half full/empty occurs, it allows or prevents the PIC from reading 16 words from the FIFO.

In write mode, these flags function similarly with FIFO ready low, indicating that the PIC may load a single word in the FIFO, and FIFO half empty low, indicating that the PIC may load 16 words in the FIFO. In addition to these FIFO status indicators, FIFO underflow and overflow error flags have been provided. A FIFO underflow error occurs whenever the read/write sequencer or PIC reads data from the FIFO when the FIFO is empty or valid data is not present at the bottom of the FIFO. FIFO overflow occurs when either the read/write

sequencer or PIC writes the FIFO when it is already full or data in the top memory location has not rippled down to a lower memory cell. Both of these errors are latched into individual flip-flops that can only be cleared when the PSDI is master cleared.

2.7 DFC firmware structure

2.7.1 Introduction

The disk file controller firmware provides the interface between the 3B20D Processor and the moving head disk for storage and retrieval of information. The firmware can make certain decisions of its own regarding the integrity of the MHDs or of itself, but will not take any actions on data content. It has a routine diagnostic exercise to check itself and a disk exerciser to check disk data format integrity. It also has sanity timers on data transfers to/from the processor and timers on disk accesses. Data is transferred between the DFC and the processor at an average rate of 1 megabyte per second. The DFC can handle up to 8 MHDs on a single controller. There are two job-processing options that may be invoked by the 3B20D. The DFC will process jobs on either a first-in/first-out basis or by a modified elevator algorithm. The elevator algorithm is used to keep the number of long head seeks to a minimum. There is also an autoread retry to help in recovering marginal data from the media.

The firmware will do as much as possible to guarantee the data integrity on a disk pack and will not allow data with bad parity to be written either to the disk or the 3B20D. The DFC will try, as much as possible, to perform 'overlapped' seeks. That is, if there are two or more requests, each to a different disk, the DFC will issue a seek to each disk before trying to do the job on any one of the disks. Thus, if a seek requires a long time to complete, and another disk has completed its seek, then the job for the disk which has completed its seek will be done.

To assure the operational capabilities of a disk, a disk exerciser is run periodically. The disk exerciser assures that the data within a given area of the disk is usable. By sequentially addressing various areas of the disk with the exerciser, eventually all areas of the disk will be verified for data integrity.

2.7.2 Timing considerations

The time it takes to get or put data on a disk file is determined primarily by how far away the data block is from the current position of the disk heads. The majority of disk accesses are less than 20 blocks in length, thus the rotational latency and seek times of the disk file become the determining factors in data transfer. In the optimal case, there is about 2 milliseconds of delay from the time the disk file

controller gets the command until it is through transferring the data. This is the sum of verifying the disk head location, doing error correction, and reading the requested block from a disk back to the processor. The worst case, assuming no DMA contention problems, is 74 milliseconds: the sum of the maximum seek time plus one rotation of the disk. The average time is 41 milliseconds.

2.7.3 Firmware tables

The DFC maintains an orderly record of data within the controller by using six tables: SYSGEN data table, circular queue table, Active Job Table (AJT), read data buffers, write data buffers, and the disk verify data table.

SYSGEN is a programmed input/output command received by the DFC. This command contains the address in 3B20D main memory of the SYSGEN table. The SYSGEN data table contains information for the DFC. The information includes the:

unload pointer
load pointer
disk driver queue address (virtual)
maximum cylinder allowed
optimization data
maximum number of jobs in disk driver's queue.

The job queue is an image of the data from the disk drivers queue. There are 16 bytes in each disk job description. There can be a maximum of 64 such jobs in the DFC at any one time.

The AJT has a series of entries to describe to the firmware the status of a particular drive. There is one such entry for each disk allowed on the controller. There are 32 words associated with each disk entry in the AJT. Some of the more important items in the list are: command, current disk maintenance state, current cylinder number, and where on the disk to start the job.

There are eight buffers set aside for the storage of data to be transferred to or from the disk. Each buffer is 263 words (16 bits each). There are seven status words and 256 data words in each buffer.

The disk verify routine checks the validity of data on the disk. This is done by reading a sector and determining if the header and data are correct. If they are not, then that disk block number is recorded in the disk verify buffer. Up to 64 such bad blocks are allowed before the verify routine will terminate.

III. SUMMARY

The 3B20D file memory system provides a maximum formatted storage capacity of 2,030 megabytes of storage on eight 300-megabyte

disk drives. The disks are managed by a disk file controller using a high-speed bit-sliced microprocessor, which utilizes 30,000 bytes of microcode to asynchronously process data transfers to and from the 3B20D main store via DMA. The disk controller interface to the 3B20D processor is via the high-speed dual-serial channel.

IV. ACKNOWLEDGMENTS

Acknowledgment is made to R. D. Adams, C. F. Ault, T. J. Clarey, R. A. Ehle, R. M. Gagliardi, T. S. Greenwood, D. H. Leonard, P. D. Olson, and E. M. Schaefer for their design contributions; and to R. C. Hatfield and N. C. Zielke for their supporting activities.

REFERENCES

1. M. W. Rolund, J. T. Beckett, and D. A. Harms, "3B20D Central Processing Unit," BSTJ, this issue.
2. J. L. Quinn and F. M. Goetz, "Deferrable Maintenance and Diagnostics," BSTJ, this issue.
3. M. E. Grzelakowski, J. H. Campbell, and M. R. Dubman "DMERT Operating System," BSTJ, this issue.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

3B20D Input/Output System

By A. H. BUDLONG and F. W. WENDLAND

(Manuscript received March 18, 1982)

The 3B20D Input/Output system supports a large number of peripheral devices of various types. The 3B20D Input/Output Processor was designed for maximum flexibility because of the diversity of the peripheral interface requirements. Because of this diversity the basic structure was developed around a high-speed bipolar multiplexor. This multiplexor interfaces with a standard microprocessor bus to numerous specialized one- or two-board computers, which interface to the peripheral devices. Power and unit designs of the input/output processor also provide flexibility in power control and ground isolation of the peripherals. The variety of peripheral units include slow- and medium-speed tape drives (including streamers), low- and high-speed data links, medium to very-high-speed printers, and scanner and signal distributor circuits.

I. INTRODUCTION

The 3B20D Input/Output system was planned from conception to be very flexible, and to support a large number of different peripheral devices. To accomplish this, several difficult system and design-level problems had to be solved.

(*i*) Processor real time—Previous experience in system input/output had shown that even low-speed input/output could be very real-time intensive. Terminal input/output, in particular, requires considerable per-character processing. The 3B20D Input/Output system, therefore, had to provide for front-end processing.

(*ii*) Lack of interface standards—Input/output standards at the physical, electrical, and protocol levels for certain peripheral devices, even those of the same functional class, i.e., tapes, floppy disks, and so

on, are relatively nonexistent. Where there are standards, such as Electronics Industries Association's RS 232 (electrical and physical standards for data link and terminal access), they often are interpreted differently by various vendors, cover only a portion of the market for a class of devices, or are subject to periodic revision. The input/output system, therefore, had to provide a standard interface that could easily support a wide range of nonstandard interfaces and power/ground systems.

(*iii*) Variations in operation and maintenance processing—Error handling, as well as operational processing, is significantly different for each peripheral device.

(*iv*) Noninterruptive growth and maintenance—The input/output system had to permit easy addition of new or replacement devices without interrupting service.

The sheer number of various devices available, their mismatch in performance to the central processing unit bus structures, and their often low cost and rapid obsolescence required an extremely flexible and adaptive interface structure.

II. THE 3B20D INPUT/OUTPUT PROCESSOR

The 3B20D input/output processor resolves the many problems stated above with five major hardware components, and a modular software structure in the host. The hardware components are: duplicated access to each 3B20D half over the two dual-serial channel interfaces; a common high-speed memory access multiplexor, the peripheral interface controller; communities of single-board computers, called peripheral controllers, which provide preprocessing intelligence and specialized device input/output interfaces; the input/output microprocessor interface bus, which provides a common Peripheral Controller interface to the Peripheral Interface Controller; and unit power design structured for flexibility and growth (see Fig. 1).

2.1 Circuit partitioning

The dual-serial channel is described in detail in this issue including the Duplex Serial Bus Selector.[1,2] The duplex serial bus selector acts in concert with the bus interface controller circuit pack to communicate with the peripheral interface controller. The bus interface controller buffers processor data and commands to the peripheral interface controller, as well as data and status information from the peripheral interface controller. It also performs the mandatory "handshaking" to communicate with the duplex serial bus selector. The bus interface controller allows the 16-bit peripheral interface controller to transmit and receive data from the higher-capacity 32-bit duplex serial bus selector at a rate the peripheral interface controller is able to accept.

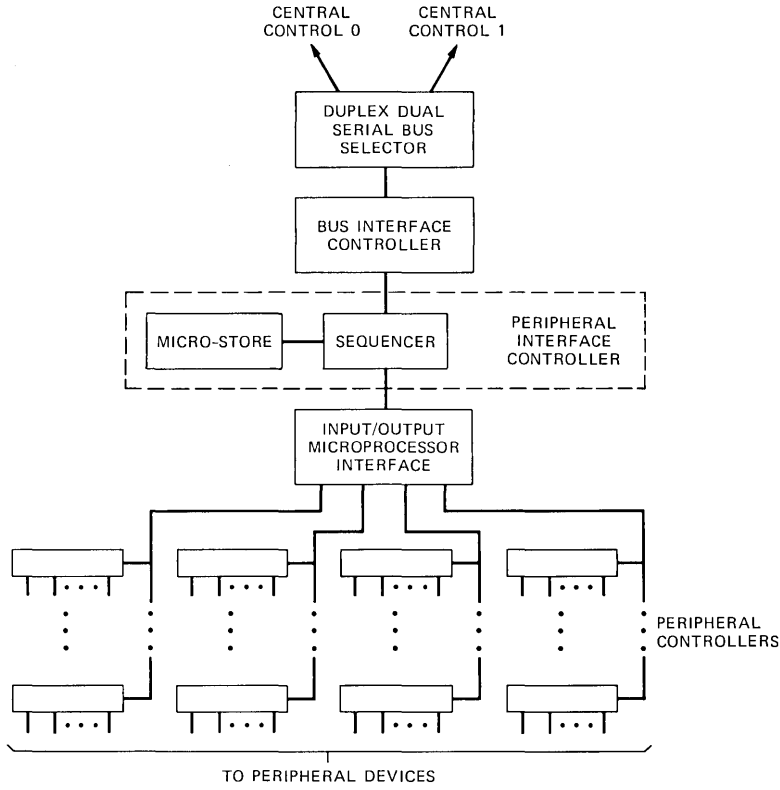


Fig. 1—The 3B20D peripheral interface.

The bus interface controller verifies the integrity of data transferred between the duplex serial bus selector and the peripheral interface controller via per-byte data-parity checks. The bus interface controller uses a 16-word by 32-bit data first in-first out register that can be alternately accessed by the peripheral interface controller and duplex serial bus selector for the transfer of data. The bus interface controller contains a 32-bit command register that records 3B20D processor commands to the peripheral interface controller, and a 16-bit status flag register and a 16-bit error flag register. The status flags are used to request and interrupt direct memory access service from the 3B20D Processor, and inform the peripheral interface controller of 3B20D commands or requests for data transfer. The error flags record the occurrence of errors and aid in fault resolution and recovery. Peripheral interface controller sanity and interval timing functions are also provided by the bus interface controller.

The peripheral interface controller is a simplex, high-speed, bipolar microprocessor. The peripheral interface controller, with its associated

bus interface circuits, handles all common maintenance and operational input/output functions between the various peripheral controllers, and the 3B20D direct memory access.

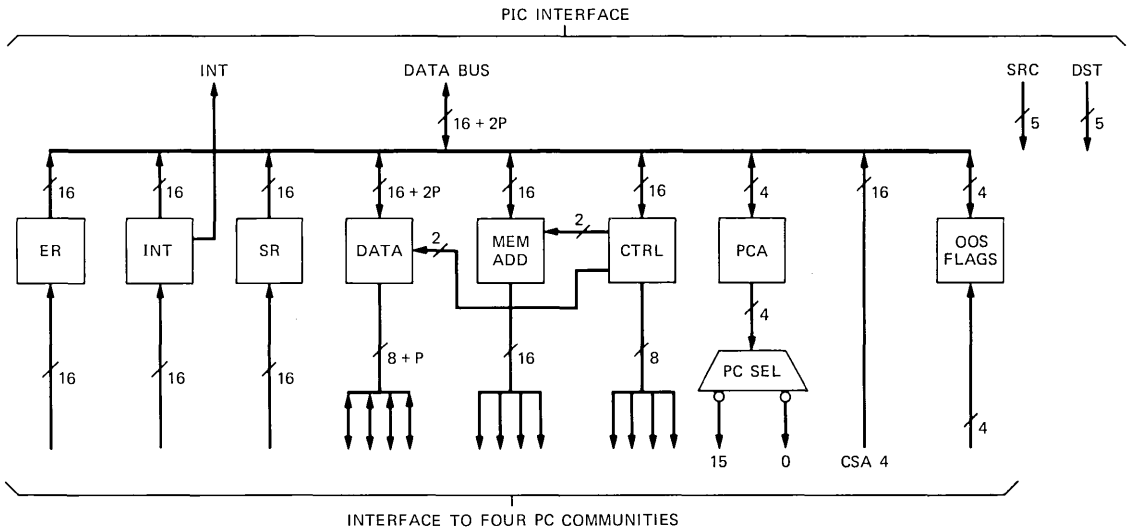
The peripheral interface controller consists of two circuit packs, a controller and one 8K-word micro control memory. The peripheral interface controller contains an arithmetic and logic unit, a 16-word general register file, a 4K-word data store, an 8-level interrupt controller, a microcontroller sequencer, and a pipe-line register. The 8K-by 40-bit micro control memory pack stores the peripheral interface controller operational and diagnostic firmware.

The input/output microprocessor interface circuit serves to interface the 16-bit peripheral interface controller with up to four communities of four peripheral controllers each. The interface between the input/output microprocessor interface and each community consists of a 16-bit memory address, an 8-data plus 1-parity-bit bidirectional data bus, and eight control pulses. Figure 2 details the input/output microprocessor interface bus structure. In addition, a private peripheral controller select signal is connected to each of the 16 peripheral controllers. The selected peripheral controller acknowledges the receipt of a control pulse by setting its control signal acknowledge. The 16 control signal acknowledge bits are compared to the 16 peripheral controller select signals to indicate to the peripheral interface controller when the acknowledgment has been received.

Each peripheral controller also has three request leads that are employed to report errors or to request service. Each peripheral controller request lead is assigned a bit in each of three 16-bit request registers: peripheral controller error, interrupt, and service request. A summary bit is provided over the interrupt register contents to trigger a peripheral interface controller interrupt whenever any peripheral controller interrupts are set. The peripheral interface controller can read each of the request registers to resolve requests to the peripheral controller level.

The four fanout branches of data, address, and control signals (one per community) are driven from three common registers. The 16-bit address register is implemented using binary up/down counters to allow autoincrement and autodecrement capability. The mode of operation is specified by bits in the control signal register. Special input/output microprocessor interface circuitry has been provided to guarantee address setup and hold times of the peripheral controller memory read and write control signals.

The peripheral interface controller data register is segmented by the input/output microprocessor interface into high and low bytes to be shipped a byte at a time to and from the 8-data and 1-parity-bit peripheral controller data buses. A flag in the control register selects



ER – ERROR REGISTER
INT – INTERRUPT REGISTER
MEM ADD – MEMORY ADDRESS
P – PARITY
PC – PERIPHERAL CONTROLLER
PIC – PERIPHERAL INTERFACE CONTROLLER
SR – SERVICE REQUEST REGISTER

Fig. 2—Microprocessor interface.

the high or low byte as the source or destination of the data transfer between the input/output microprocessor interface and the selected peripheral controller. An additional bit in the control register is used to toggle the data byte selector flag after a peripheral controller memory read or write cycle. The circuit incorporates a delay timing chain to guarantee data hold time during peripheral controller memory write operations.

The control register is partitioned into two fields; an 8-bit register, the outputs of which directly drive the control signal buses; and an 8-bit field that controls internal functions in the input/output microprocessor interface.

An out-of-service bit is provided for each of the four peripheral controller communities. This 4-bit register in the input/output microprocessor interface can be read and written by the peripheral interface controller. Additionally, a power-down condition within a community will automatically set its out-of-service flag. When out of service, the interrupt requests of the four peripheral controllers within that community are not ORed into the interrupt register summary.

2.2 Peripheral controllers

The peripheral controller is a microcomputer system that serves as an intelligent interface between the peripheral interface controller and the slow- to medium-speed peripheral units. The peripheral controller is responsible for all device-specific front-end processing and per-byte interrupt handling. This relieves the 3B20D central processing unit from low-level tasks and greatly increases the number of peripheral devices the 3B20D can support. Figure 3 shows a typical one-board peripheral controller design. Up to four subdevices can be separately addressed within a peripheral controller by the peripheral interface controller.

2.2.1 Peripheral interface controller-to-peripheral controller communications

Subdevices are capable of initiating several types of work requests. A peripheral controller may initiate a service request that results in the peripheral interface controller reading service-request status information to determine the type of work required. A subdevice activates a service request to transfer data to or from 3B20D main memory or to indicate a job is completed. In addition to service requests, a subdevice may at a high priority level, initiate an interrupt request that results in interrupt-request status information being read by the peripheral interface controller. The peripheral controller interrupt request is activated by a subdevice to transfer high-priority informa-

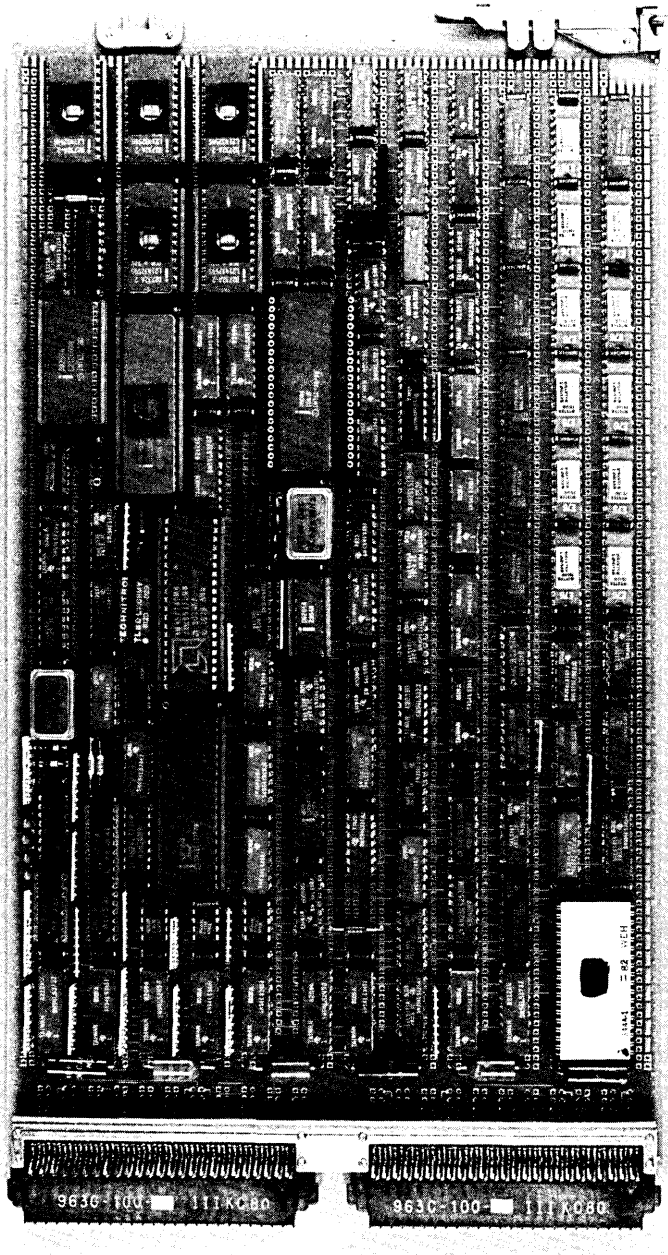


Fig. 3—Typical one-board peripheral controller design.

tion to or from 3B20D main memory or to report a high-priority job has completed.

All operational communication between the peripheral interface controller and the peripheral controller is done through a dual-access memory resident on each peripheral controller. Dedicated memory locations are reserved within the dual access memory that contain pointers used by the peripheral interface controller to locate each subdevice data buffers, data transfer parameters, and status and command areas. The dual access memory also contains memory locations that contain information at the peripheral controller level such as service and interrupt request data and peripheral controller command area pointers.

Each of the peripheral controllers is an independent single-board microcomputer. The hardware is specialized to handle the particular interface and is not necessarily the same as any other peripheral controller. However, all peripheral controllers have a functionally common front-end design and maintain the same communication protocol with the peripheral interface controller. The input/output processor has been designed to allow the peripheral controllers to exist as autonomously as possible. This capability will enable future peripheral controller designers to use state of the art microprocessors and microprocessor-controlled peripherals.

2.2.2 Peripheral controller hardware architecture

Although peripheral controllers designed for specific interfaces are different, all peripheral controllers have the following components: microprocessor, read-only memory, random access memory, service and interrupt sources, isolation circuitry, error detection logic, and scan back circuitry.

The peripheral controller employs a microprocessor to administer the transfer of data between its peripheral units and 3B20D main memory. There are no requirements dictating the use of a specific microprocessor type. The intent is to allow a peripheral controller design to take advantage of the microprocessor that most efficiently handles a particular peripheral unit.

The peripheral controller houses a bootstrap program stored in read-only memory that is entered when power is initially applied or when the peripheral controller reset function is activated by the peripheral interface controller. The bootstrap program causes the microprocessor to initialize its periphery and to recognize peripheral interface controller commands. In addition to the bootstrap program, the read-only memory may contain operational and diagnostic programs.

The peripheral controller uses random access memory to store the operational program, and buffer transient data, and to provide storage

for peripheral interface controller communication parameters. The portion of random access memory containing the data buffers and communication parameters is accessed by both the peripheral interface controller and peripheral controller microprocessor. Therefore, this portion of random access memory must provide dual-access capability.

To prevent an insane microprocessor condition from babbling on the common bus, the peripheral controller is provided with an isolation state. In this state, isolation circuitry removes all response signals generated by the associated peripheral controller from the bus. In some peripheral controller applications the isolation state also prevents data transmission to the peripheral units connected to the peripheral controller.

2.2.3 Service and interrupt request generation

The peripheral controller generates a service request or an interrupt request whenever a data transfer to or from 3B20D main memory is required, or to indicate that a command completion response is available. The peripheral controller provides a separate pulse source for the interrupt request and service request indication. The peripheral controller must refrain from sending a service request or interrupt request until the previous one has been recognized by the peripheral interface controller. The peripheral interface controller indicates recognition of a service request or interrupt request by clearing the associated request pending flag in the peripheral controller dual-access memory.

Each subdevice within the peripheral controller is assigned 4 bytes of dual-access memory, which contain the status information for the associated subdevice. The status information is comprised of generic and user-defined data, and is read as a result of subdevice initiated service requests and interrupt requests.

2.2.4 Hardware error detection

The peripheral controller provides parity checking and generation for data contained in the dual-access memory. Since most microprocessor and microprocessor peripherals available today do not carry parity within the device, routine diagnostics must take over the responsibility of error detection for these devices. Parity over read-only memory data is not a requirement. This enables peripheral controller designs to take advantage of the available large, 8-bit-wide, read-only memories. However, a program sanity check must be provided to detect a read-only memory data failure. A parity error flip/flop is used to store the indication of a dual-access memory parity failure. The peripheral controller also provides control logic to invert parity generation in order to check the parity circuits.

Peripheral controllers operate autonomously with regard to peripheral interface controller activity and, therefore, provide their own system clock. A system clock check circuit is provided by the peripheral controller to detect clock activity abnormalities. A clock error flip-flop is used to store the clock failure indication. The error lead is employed to report the detection of such failures.

Due to the complexity of the peripheral controller and the limited amount of parity checking provided, the peripheral controller uses routine maintenance diagnostics to enhance error-detection capabilities. Redundant software, rather than redundant hardware, is employed to detect fault conditions. The peripheral controller microprocessor periodically executes routine maintenance diagnostics during nonpeak data transmission time intervals. A routine maintenance diagnostics error flip-flop is provided to store the routine maintenance diagnostics failure.

To assure program execution sanity, the peripheral controller provides a sanity check mechanism. The sanity check complexity is directly proportional to the destructive powers of an insane peripheral controller microprocessor. A sanity failure flip-flop is used to store the sanity failure indication.

2.2.5 Scan back circuitry

The peripheral controller provides four directly addressable read-only functions: status and error information, address lower loop-around data, address upper loop-around data, and peripheral controller type-identity code.

The peripheral interface controller uses the scan back address data in a loop-around mode to check the integrity of the direct memory access address and data buses and to verify that a particular peripheral controller is receiving all the address information.

III. PERIPHERAL INTERFACE CONTROLLER AND PERIPHERAL CONTROLLER FIRMWARE AND SOFTWARE

3.1 Peripheral interface controller firmware

The functions performed by the peripheral interface controller are carried out at two levels: base level and interrupt level. Deferrable tasks are performed during base level processing and nondeferrable at interrupt level (see Fig. 4).

Peripheral controller service requests are handled during the peripheral interface controller base level loop. The peripheral interface controller interrogates the peripheral controller service request register to ascertain which peripheral controllers are requesting service. The peripheral interface controller then reads the peripheral controller status bytes in dual-access memory to determine the type of job to be

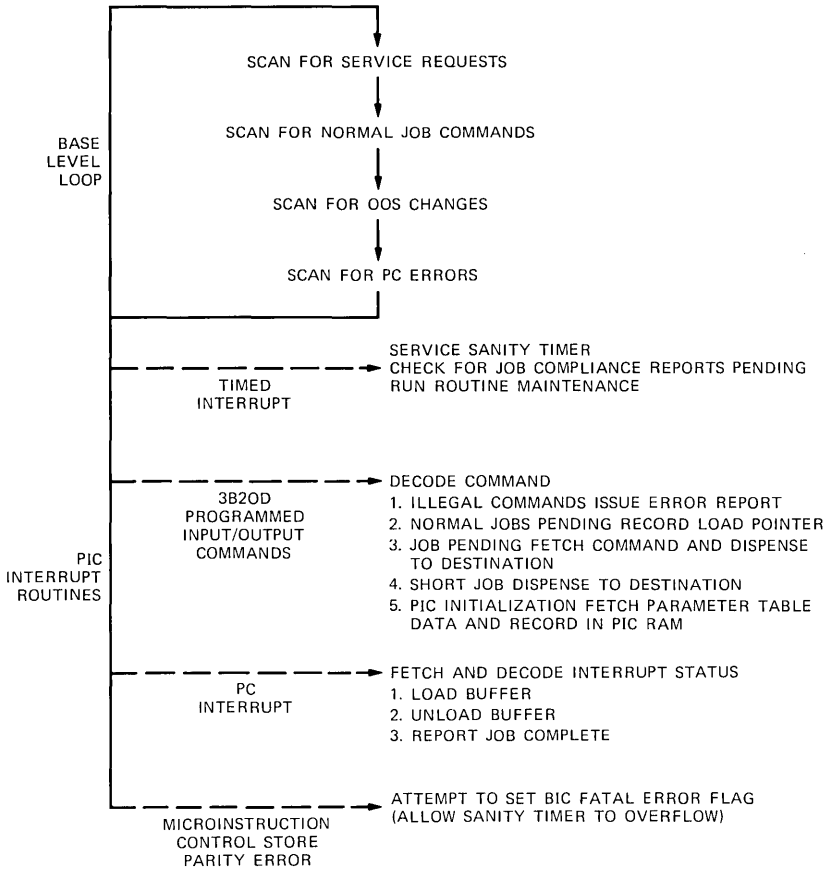


Fig. 4—Peripheral interface controller firmware structure.

performed and executes that job. Job sizes are restricted to a maximum transfer size of 256 bytes to guarantee the maximum time the peripheral controller is held out of dual-access memory. The peripheral controller breaks the transfer of a large block into small blocks to satisfy this requirement. Reports of completed jobs are buffered in an area of the peripheral interface controller data store.

After servicing peripheral controller requests, the peripheral interface controller compares the load and unload pointers in a job queue stored in 3B20D main memory. If the queue is not empty, the peripheral interface controller unloads up to eight jobs and issues them to the appropriate peripheral controllers or to the peripheral interface controller itself.

The peripheral interface controller next scans for changes in the peripheral controller community out-of-service status. A power con-

verter out of tolerance causes an out-of-service indicator to be automatically set in the input/output microprocessor interface. A high-priority error response is issued to report such a condition.

The peripheral interface controller next scans for peripheral controller error reports. Again, the peripheral interface controller reports this via a high-priority error response. The loop is complete at this point and the peripheral interface controller begins scanning for peripheral controller service requests once again.

Interrupts are employed to initiate time-critical tasks. A timed interrupt routine is entered at a fixed rate. The sanity timer is serviced at this time. Additionally, the job-completion queue is interrogated and, if not empty, is loaded into the response buffer in 3B20D main memory. The peripheral interface controller then interrupts the 3B20D.

High-priority jobs are handled using the interrupt mechanism. Upon registration of the message in the high-priority buffer in the peripheral controller dual-access memory, the peripheral controller interrupts the peripheral interface controller. The peripheral interface controller moves the buffer contents into 3B20D main memory. This process continues until the entire message has been transferred into 3B20D main memory. The peripheral controller then enters a job-completion report into the appropriate entry in the peripheral controller dual-access memory and again interrupts the peripheral interface controller. The peripheral interface controller then moves the report into the high-priority response buffer in 3B20D main memory and interrupts the 3B20D.

Programmed input/output commands arriving from the 3B20D also trigger a peripheral interface controller interrupt. The peripheral interface controller decodes the command to determine its type. New entries in the 3B20D main memory job queue are reported to the peripheral interface controller in this manner. A portion of the command contains the 3B20D load pointer for the job queue. The peripheral interface controller records this entry into the peripheral interface controller random access memory to be used during base level processing.

The 3B20D also uses programmed input/output commands to alert the peripheral interface controller of the presence of a high-priority job awaiting execution in the 3B20D main memory high-priority job register. The peripheral interface controller retrieves the command and immediately issues it to the specified peripheral controller.

Additionally, short jobs can be transferred to the peripheral interface controller via programmed input/output. Jobs of this sort are constrained to contain no more than 28 bits of command information. Only a single 32-bit programmed input/output command is used.

3.2 Peripheral controller software

3.2.1 Initialization

Peripheral controller initialization is initiated by a 3B20D Processor resident input/output processor fault recovery program. The fault recovery program can direct the peripheral interface controller, via a configuration command, to generate a clear signal for any one of the 16 addressable peripheral controllers within the input/output processor. The peripheral controller responds to the clear signal by transferring control to a bootstrap program resident in on-board read-only memory. Execution of the bootstrap program results in initialization of address pointers, flags, timers, interrupt control circuitry, and other parameters as required. In addition to the above mentioned items, the peripheral controller must initialize associated work pointers. On-board peripheral controller diagnostics are downloaded from the central processing unit during this stage.

Upon completion of initialization the peripheral controller microprocessor monitors sanity and responds to any commands placed in the peripheral controller work queues.

3.2.2 Peripheral interface controller/ peripheral controller microprocessor command execution

A peripheral controller microprocessor can be directed to perform certain tasks through the use of a command queue system. A peripheral controller can have as many as 15 queues, three of which are for generic type commands. The remaining queues are double ended (deques) and provide communication to peripheral controller subdevices. Each of the four peripheral controller subdevices has a transmit, receive, and asychronous report deque. Jobs are sent to the transmit and receive deques informing the peripheral controller subdevice to send and receive data respectively. Unlike the transmit and receive deques the asychronous report deque only sends subdevice reports, it never receives jobs for execution. The command queues and their associated pointers—load, unload, and current—reside in the peripheral controller dual-access memory. The peripheral interface controller, after entering a command in the queue, informs the peripheral controller by updating the queue load pointer. The peripheral controller detects the entry, processes the job, and inserts a completion report in the queue. The completion report includes a 2-byte completion code. The peripheral controller receives knowledge of the completion report via a service request. Retrieving the report, the peripheral interface controller sends it to 3B20D main memory and updates the unload pointer. The peripheral controller maintains a current job pointer, which allows the peripheral interface controller knowledge of which peripheral controller job is currently running.

A unique completion code has been reserved as an indication of an invalid command. Upon detection of this condition, the microprocessor immediately loads the invalid-command completion code byte into the appropriate field in the command/response entry and signals a job-completion report via a service request or interrupt request. At this point, the microprocessor is ready to accept the next entry in this command deque.

For high-priority command entries, the microprocessor may set an interrupt request indicator instead of setting a command completion service request indicator. The peripheral interface controller recognizes and responds to all peripheral controller interrupt request work before responding to peripheral controller service request work.

3.2.3 Peripheral controller microprocessor base level and interrupt level work

There are three modes of communication between the peripheral controller microprocessor, subdevices, and peripheral interface controller:

(*i*) Demand interrupt mode—The peripheral interface controller interrupts the peripheral controller microprocessor to indicate a high-priority command has been placed in the high-priority work deque. The peripheral controller microprocessor immediately responds to the command interrupt by accepting the command and executing the associated code as required. The subdevice may interrupt the peripheral controller microprocessor, indicating receive data is present. In this mode the microprocessor checks the data for errors and, if none are found, moves the data to an associated input buffer. When the input-buffer-full threshold is reached, the microprocessor uses a command address pointer to locate a receive command. The receive command contents are required to permit the peripheral controller microprocessor to initiate a direct memory access transfer of data.

(*ii*) Base level polling mode—In the base level program, all subdevices are polled in sequence for status and any required processing is performed. The status of all currently active direct memory access transfers is updated and new direct memory access jobs are initiated as required. Typically, all subdevices transmit data, and low-priority maintenance work is done during the base level loop.

(*iii*) Time interrupt mode—A real-time clock provides periodic interrupts to the peripheral controller microprocessor for the purpose of handling low-level periodic functions. Some examples of the functions processed in this mode are: checking subdevice status, maintenance of a sanity timer, and administration of routine maintenance diagnostics.

3.3 Peripheral interface controller firmware development tools

Available for development of peripheral interface controller firmware are (*i*) the Read-Only Memory Emulator and Trace Control Unit circuit packs and (*ii*) the Read-Only Memory Emulator and Trace System software system. These function jointly to support the peripheral interface controller as a test tool during hardware and firmware debugging. The read-only memory emulator and trace system has hardware control over the peripheral interface controller and provides program single-stepping and breakpointing capabilities. Program memory can be examined, modified, and downloaded, via a commercial microprocessor based system. A trace system is provided with three 16-bit by 256-word trace memories.

IV. INPUT/OUTPUT PROCESSOR UNIT AND POWER

The input/output processor unit and power is structured around the peripheral interface controller and the four communities of peripheral controllers as separate items. Each is located within its own housing and is separately powered. An overall power control switch is provided with the peripheral interface controller; and each peripheral controller community may be individually switched without affecting the others (see Fig. 5). Although several standard voltages are supplied in the backplane, other voltages can be supplied as part of the peripheral controller design, if required.

Since several of the problems of interfacing to various peripherals are related to the wide range of electrical and physical interfaces required, the input/output microprocessor interface bus connector field was separated from the peripheral device defined connector area for each peripheral controller slot. The entire upper half of each peripheral controller's backplane connector was reserved for definition by the particular application. Standard Bell System connectors are then used to generate a cable or harness that performs the translation to the specific connector of each physical device. The foundation peripheral controller for No. 5 ESS and the programmable link controller for the signal transfer point system used in the Common Channel Interoffice Signaling network are two examples of application peripheral controllers made possible by the flexibility of the input/output processor. Multiple board peripheral controllers have also been designed for special applications using connectorized straps in the upper connector area to interconnect the circuit packs that comprise the multiple board peripheral controller. Circuit pack slots are thus system defined, depending upon the peripherals needed by each application.

Because all four peripheral controller communities are on separate

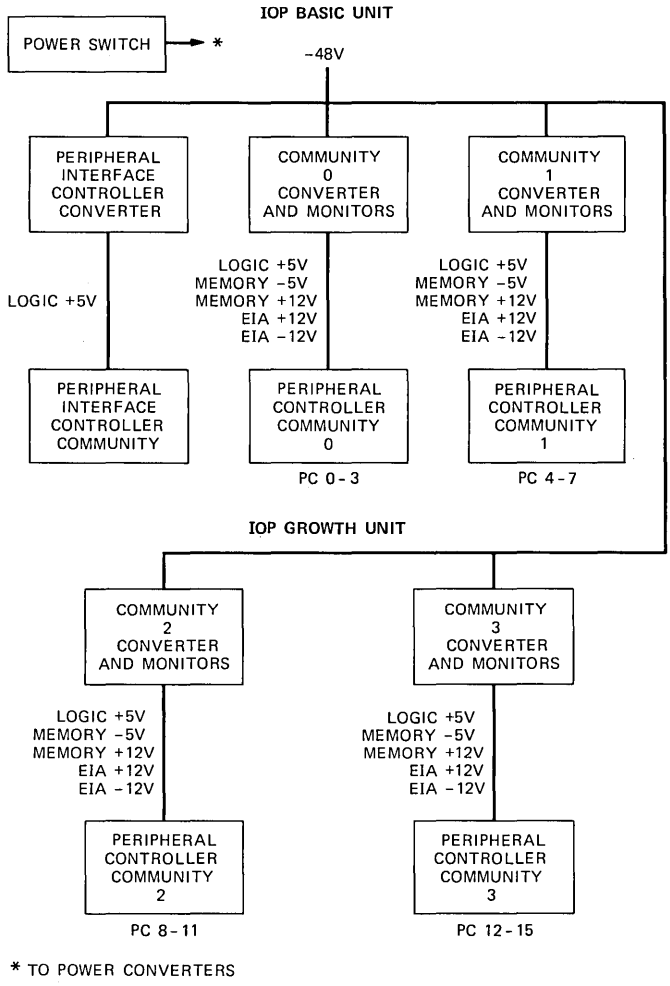


Fig. 5—Input/output processor power diagram.

power systems, peripheral controllers with special grounding or power requirements can be assigned to separate peripheral controller communities, effectively isolating them from the remaining communities, and providing separate power control.

V. 3B20D PERIPHERALS CURRENTLY SUPPORTED

The peripheral controllers and peripheral devices available on the 3B20D Processor are discussed in the following paragraphs.

5.1 Nine-track magnetic tape controller

Up to four 25-inches-per-second (IPS), 1600-bits-per-inch (BPI), phase-encoded (PE) nine-track tape transports can be accommodated per peripheral controller (only one transport may be accessed at a time via a shared formatter). The drive accepts up to 10.5-inch-diameter reels (2400 feet by 0.5-inch tapes). The interface is an RS422 differential bus up to 250 feet long using an industry standard data and control format. The controller can read and write any block size up to 2K bytes long, write file marks, seek forward or reverse, and can rewrite and reread blocks of data that exhibit errors. The instantaneous data rate is 40K bytes per second. The average data transfer rate for 2K data blocks is 26K bytes per second.

5.2 Scanner-signal distributor controller

Forty-eight scan points and 32 signal distributor points—which may be located up to 1000 feet away (assuming 26-gauge twisted pair)—are provided by this controller. Scanning is carried on autonomously or it is directed. A change of a scan state must remain in the new state for two consecutive scan cycles before it is recognized. The signal distributor can operate or release a point, or it can flash a point indefinitely or for a timed period. Both the scanner and the signal distributor are self protected to 140 volts, and can detect a foreign potential in the same range. A point scanned must be a floating contact or saturated transistor switch. Points are scanned once every 48 ms. The signal distributor can supply up to 5 mA into a floating load or loop. The scanner signal distributor was designed primarily to operate optical couplers and low-power relays. Scan or distribute points that are remote to the 3B20D Processor are interfaced through an optical coupler isolation circuit.

5.3 High speed nine-track magnetic tape controller

Up to four 12.5- to 125-IPS, 1600-BPI, phase encoded, start-stop transports, or up to four 25-IPS streaming transports with embedded formatter, in any mix, may be accommodated per peripheral controller. Access is via an RS 322 bus limited to 20 feet using an industry standard data and control format. Only one transport may be accessed at a time. The controller can write or read any block size up to 6K bytes as well as perform the same control functions as the peripheral controller described in 5.1. The instantaneous data transfer rate is 200K bytes per second. The average data transfer rate when reading 6K blocks of data is 150K bytes per second.

5.4 Synchronous data link peripheral controller

This circuit pack is a BX.25 level-2 synchronous link arranged for full duplex private line or dial backup operation, and it provides two independent channels at bit rates up to 9.6K b/s. The channels differ in that one supplies an automatic call unit port for dial-up and dial-in operations. The backup configuration is supported when equipped with Dataphone II generation data sets. The peripheral controller capacity is 9600 b/s full duplex. Each of the two BX.25 level-2 channels has an associated level-1 interface that is RS449/RS232C compatible.

5.5 Asynchronous data link peripheral controller

This peripheral controller provides two independently programmable full-duplex or half-duplex, asynchronous or isochronous channels at data rates up to 9.6K b/s. It can communicate with a local terminal directly or via a "null modem" connection, with a remote terminal via a modem and private line, or with a remote terminal via a modem and the public switched telephone facility. The user can specify the data rate, parity and stop code format in addition to input, output, and line discipline processing options. Each channel drives an optically isolated EIA RS232C standard port.

5.6 Maintenance teletypewriter peripheral controller

This peripheral controller is designed to permit manual intervention and control of the 3B20D control unit and peripherals through the maintenance terminal. It contains sufficient code in read-only memory to allow control of the system during routine or emergency maintenance procedures. Read-only memory code performs memory and device initialization, parsing of the 3B20D emergency action interface input and support for the maintenance terminal display and a remote maintenance center data link using BX.25 protocol. Down-loaded random access memory code provides additional functions such as support for teleterminal input/output syntax and other teleterminal display capabilities.

5.7 BX.25 data link interface

The data link controller provides a general-purpose high-speed synchronous BX.25 level-2 interface and a special high-capacity software interface to the 3B20D. The peripheral controller is equipped with an RS232C and a CCITT V.35 interface having data rates of up to 56K b/s.

VI. SUMMARY

The 3B20D Processor's input/output structure was designed around a high-speed direct memory access input/output channel, a high speed

multiplexor and direct memory access interface, and single- and multiple-board intelligent controllers, which provide device-specific front-end processing and interrupt handling. These components, associated with a common driver with device-specific handlers in the 3B20D Processor's operating system, successfully provide a flexible and effective interface to a wide range of peripheral devices.

VII. ACKNOWLEDGMENTS

The 3B20D Input/Output Processor and its peripherals are the result of a team effort of circuit, physical, and software design groups. The authors wish to acknowledge T. S. Greenwood, J. O. Becker, and R. T. Watters for their management support of the project, and the particular design efforts of T. T. Butler, D. J. Brahm, W. V. Lindquist, N. E. Stohs, J. L. Snapp, T. T. Towle, S. P. Michiels, M. G. Bradac, S. P. Davison, W. A. Hommowun, W. R. Hudgins, and R. M. Venzon, C. A. Mennitt, C. B. Kelley, W. G. Reichert, R. Huisman, all of whom helped to make the concepts become a reality.

REFERENCES

1. W. N. Toy and L. E. Gallaher, "Overview and Architecture of the 3B20D Processor," B.S.T.J., this issue.
2. M. W. Rolund, J. T. Beckett, and D. A. Harms, "3B20D Central Processing Unit," B.S.T.J., this issue.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

Software Development System

By B. R. ROWLAND and R. J. WELSCH

(Manuscript received March 22, 1982)

The 3B20D Processor software development system is an integrated collection of tools and procedures that is used in the development and administration of all 3B20D Processor software. This article describes the tools and procedures and their use in developing the 3B20D Processor software. These tools and procedures include compilers, assemblers, and loaders, as well as change-administration and load-building procedures. The most important characteristic of the development system is the balance between the enforcement of the project standards and the flexibility offered to developers.

I. INTRODUCTION

The software that comprises the operating system, diagnostics and fault recovery, configuration data base, field utilities, and craft interface for the 3B20D Processor has been undergoing development and change over several hundred developer years. Without a strict change-administration strategy and a productive development environment, the tight schedules and reliable deliveries characterizing this system would not have been possible.

This paper presents a description of the software-development environment used by the project's programmers and those administrating 3B20D Processor subsystems. Without the administrative control, these developers might otherwise be limited to simply compiling or assembling programs and attempting to test software with an ad hoc version of the total system software. The emphasis presented here is that of the roles of those involved in the software change process, the tools they each use, the flexibility offered by the tools and administration structure, and the standards and strategies enforced in the development environment.

II. THE ADMINISTRATION OF SOFTWARE DEVELOPMENT

The 3B20D Processor project started with relatively small teams of engineers, compilation tools, and a primitive testing environment. The project has grown to a size that now requires considerable machine support, sophisticated tools, and administrative control. Over 100 developers are responsible for nearly 8,000 source files resident across eight support computers. Compilation of all of these files into a full load takes over 24 hours of machine time on a commercial 16-bit minicomputer. This large amount of software has been partitioned into 23 logically manageable subsystems, the basis for administration and testing.

The 3B20D Processor is the target for output from a software development and administrative environment that resides on the host support computers. All object code for the target is archived, edited, and cross-compiled on the host and then transported by tape or data link for testing on target 3B20D Processors. This separation of host and target ensures a stable development environment as the target machine evolves with new features and performance improvements that may consist of changes in hardware, software, or firmware.

The software administration and development strategy can best be described as one of well-defined and closely tracked data movement between nodes on development host machines and it requires a specific scenario dictating when the data movement takes place. The data are requests for software changes and program files, and the nodes are instances of the software structure reflecting some portion of the total software in one of several development states. Development activity is spread across the host machines by partitioning the software into nearly independent subsystems.

In the development environment, a file system structure containing copies of controlled software source and generated object is referred to as a *node*. When a node is populated with source, object, and target products—all in a like state of development or approval—the node is called a *view* of the software. For each major release, the development system supports three views of the software.

(*i*) The *official* view, or node, contains all of the official source, object, and target products released to customers.

(*ii*) The *approved* view contains those source, object, and target products that have been released as emergency corrections to the last major release. The approved view is an incremental addition to the official view.

(*iii*) The *under-test* view contains those source, object, and target products that will be sent out with the next major release. The under-test view contains everything from the approved view and is otherwise an increment from the combination of the official and approved views.

A final view, that of the developer, may include a mixture of the above views in combination with privately modified versions of various files undergoing a development change.

The approval of a software change submitted by a developer involves the successful integration of that change into a known stable software base. The data movement and approval strategies have a common thread: the incremental modification of a known good software base to produce the next iteration of that base.

An important decision made in the design of the 3B20D Processor development environment was the centralization of the activities used to construct the collection of libraries and objects for installation (i.e., the *load*) on a machine isolated from development activity. This allowed efficient load-building techniques to be developed that do not interfere with developer activity. The implication of centralized load building is that developer changes must be collected and moved to a common point where official load building takes place. In fact, it is only source changes that are moved for load-building purposes. All changes to products are then reconstructed from changed source. Results are then redistributed back to the appropriate development machines.

III. MAJOR SOFTWARE DEVELOPMENT SYSTEM COMPONENTS

It is the interaction of four distinct software administration and generation systems that creates the 3B20D Processor Software Development System (SDS). This section presents an overview of these systems; Section IV describes their use; and Section V describes the major standards that contribute to the environment and are enforced by the project's tools.

3.1 The Modification Request System

The Modification Request (MR) data base system is a general-purpose hierarchical system tailored to 3B20D Processor change-tracking requirements. The MR is the entity that identifies all requests for 3B20D Processor software changes and tags all source changes made by developers and administrators to official 3B20D Processor software. The MR system serves as a central master data base from which administrators control and track all software changes and generate appropriate tracking and status reports.

As implemented for this project, the MR data base is actually a three-level data base comprised of an MR level, one or more release levels per MR, and one or more subsystem levels per release. The higher the level (the MR level is highest), the more global the scope of the information maintained.

The MR level information includes an identification number, origin-

ator information, problem description, priority, severity, and due date. At the release level, the data base includes due date, feature engineer and developer, and priority. Subsystem level data includes responsible developer, modified source files, status, solution description, and load-building instructions. The majority of the information is automatically generated and stored in response to developer and administrator actions. For the most part, direct data entry is not necessary.

The MR system includes a high-level query language and report-formatting facilities for producing reports from the data base. The query language is very powerful in that it allows for selection specification and sorting levels. The reporting facilities process the output of the query language. Additional capabilities exist that cause brief reports, called synopsis reports, to be generated automatically and sent to appropriate administrators, supervisors, and/or developers in response to particular status changes. For example, when an MR requires a change in a particular subsystem, the responsible developer receives a synopsis report. When the responsible developer submits changes in response to an MR for system test, appropriate administrative personnel receive synopsis reports.

3.2 The Change Management System

The Change Management System (CMS) is a facility compatible with the *UNIX** operating system aimed at controlling the activities of both developers and administrators related to software change and change approval. The CMS uses the *UNIX* operating system Source Code Control System (SCCS) and its own relational data base to track and relate each change made in a source file to an MR.[1]

All official 3B20D Processor source files are maintained via CMS on a per-subsystem basis. A CMS instance is defined to be a set of SCCS-controlled source files and the relational data base tracking the MR-based changes. Each subsystem is a unique instance of CMS. There are, therefore, one or more CMS instances on each development machine (see Fig. 1). This strategy was chosen over a strategy having only one CMS instance per machine for the following reasons:

(*i*) Corruption of a CMS data base would only affect one subsystem.

(*ii*) The majority of the developers need access to the official source files for only one subsystem.

(*iii*) Separate CMS instances per subsystem allow load balancing on the development machines to be more easily implemented. Instead of having to extract a portion of a large data base and move it with developers to another machine, an entire CMS data base can be moved.

---

* Trademark of Bell Laboratories.

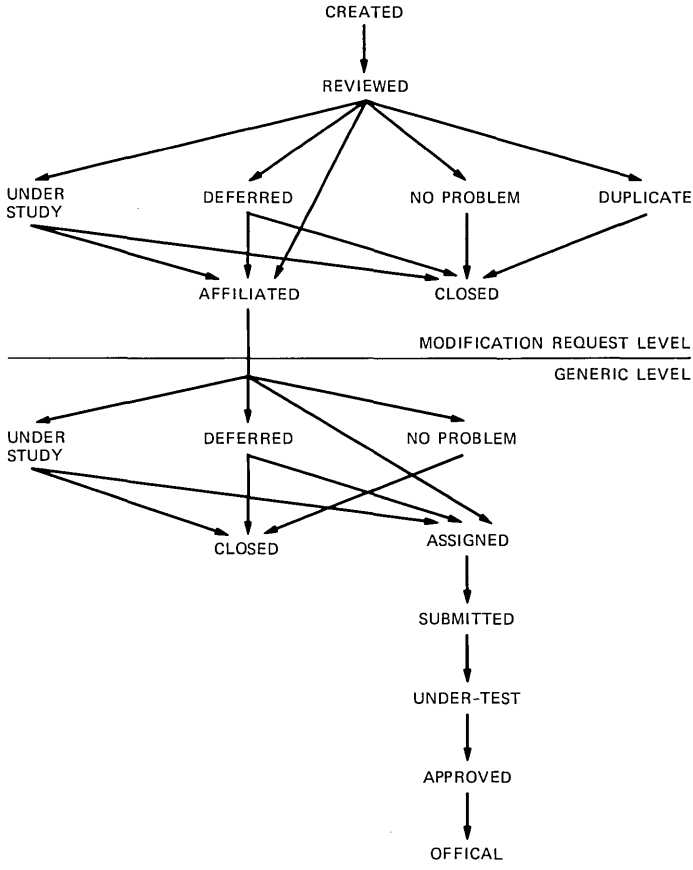


Fig. 1—Diagram of the state changes in the Modification Request System.

3.3 The Software Generation System

The software development system is geared to construct executable software for the following three target environments:

(*i*) A 3B20D Processor running the DMERT operating system.[2] This is the primary target for official system software construction.

(*ii*) A PDP 11/70 running a real-time version of the *UNIX* operating system. This is the recognized 3B20D test laboratory support processor.

(*iii*) A PDP 11/70 running the *UNIX* operating system. This is the primary machine and operating system for development and official software construction.

The primary target for which the SDS generates code is the 3B20D Processor. A Software Generation System (SGS) containing a compiler, assembler, and link editor is provided for generating and sup-

porting 3B20D Processor binary executable object modules on the development and load-building machines.

3.4 Building software views

The major tool used by both developers and administrators in conjunction with the SGS for generating 3B20D Processor loads is the build tool. This tool invokes the minimum number of commands necessary to create a requested object incorporating selectable states of related software from its components. It determines, according to a specified set of dependency information, exactly which objects must be rebuilt (because they may be out of date) and which objects are current with respect to the changes desired in the requested object to avoid unnecessary compilation steps. This dependency information is automatically generated by a utility from the software components. This ability to create an object with particular versions of software that reflect its state or submitted MR software changes is known as *building a software view.*

IV. THE SOFTWARE DEVELOPMENT PROCESS

All software in the DMERT environment is created, corrected, or enhanced in conjunction with an MR. With this association of an MR to every software introduction or change, all development on the project can be adequately monitored. The assignment of an MR allows a developer to make modifications to official project source and to ultimately submit the changes for approval by project administrators. The development activities that occur and the tools used in the process between the assignment of an MR and the submission of changes are described in this section.

4.1 MR handling

The MR system maintains modification requests in a variety of states (see Fig. 1) that reflect the activity being taken on the request. An MR is *created* by other developers or applications and then initially *reviewed* by administrators. It may be found that an MR is a *duplicate* of a previously resolved MR; it may be required to be placed *under study*; it can be *deferred* for later consideration; or it may be found to be *no problem* in which case the MR is ultimately *closed.* The under study or deferred MRs eventually will be either *closed* or *affiliated* with generics for action. Once the MR level state is affiliated, then generic level action is allowed.

MR assignment is made to one or more subsystems, as many as are required to implement a solution. When an MR is assigned, a developer then creates a development node in which to maintain copies of source to be created or modified in response to the modification request and

to build a view of the subsystem affected by the MR. This private development node gives the developer the freedom to make any changes with any set of tools on local copies of source files. The developer is now free to experiment with solutions to satisfy the MR, while official source remains protected by CMS. Source files intended to be modified by the developer for the MR are requested via CMS commands that associate the changes with the MR and the release to which the MR is being applied. The association of MR to source files changed is kept in the CMS data base to allow administrators or other developers to request versions of software containing solutions to particular problems. While a developer has copies of official source to be worked on in a private node, all other developers are prevented by CMS from obtaining copies of the same source to avoid creating unsynchronized changes. Using private source copies, the developer is ready to make changes and create new subsystem objects for testing.

4.2 Languages and tools

Software for the 3B20D Processor target is written primarily in the high-level programming language C with occasional use of the 3B20D Processor assembly language. The C language contains many modern control and data structures found in languages such as PASCAL. It is characterized by its brevity of expression, direct access to data type representation, and operations and declarations available to the programmer to enhance the generation of efficient assembly level code. The 3B20D Processor assembly language is a member of IS25, a 3B20D family standard specifying:

(*i*) Activation stack format
(*ii*) Data type representations
(*iii*) Registers
(*iv*) Operations and addressing modes for accessing and manipulating data objects ranging in size from a single bit to 32-bit words.

The tools used to compile C programs and assemble assembly code into user level objects are modeled after the tools used with the *UNIX* operating system and for the development of the *BELLMAC**-8 microprocessor.[3] An additional complement of tools is used to create and modify special process files and prepare the developer with information that will be very valuable in the testing environment.

A single command can be used to control the C program compilation process. This command invokes a source code preprocessor, the C compiler and optimizer, assembler, and link editor. These four tools in turn convert a collection of C programs into a single object with addresses that are either relocatable or absolute (Fig. 2).

---

* Trademark of Western Electric.

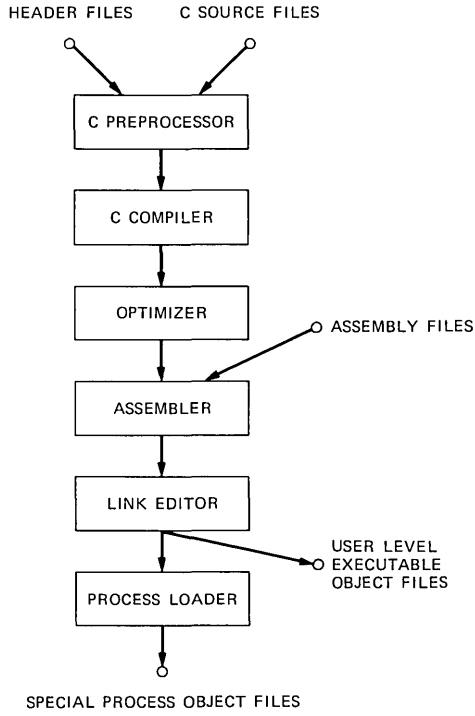


Fig. 2—Compilation tools.

The preprocessor provides a macro expansion facility and directives for sharing common source (or *header*) files. Header files typically contain data and macro declarations shared by many programs. This mechanism ensures that common definitions are consistent across all programs using them because they are accessed from the same source.

The C compiler and optimizer are based on the portable C compiler[4] and a similarly structured portable code improver. The compiler generates assembly code from preprocessed C source using syntax-directed parsing (from *YACC*[5]) and modified Sethi-Ullman[6] and Aho-Johnson[7] tree-matching and expression-optimization techniques. The generated assembly code can then optionally be passed through an optimizer (or, more correctly, code improver) that eliminates unnecessary branching, removes redundant register loads, and converts certain instructions or sequences of instructions into simpler or more efficient, yet semantically equivalent ones. Both the compiler and optimizer leave their result in the form of an assembly language file.

From assembly language source, the assembler creates an object file containing object code text and data (with optional relocation information), symbol and source line number tables to communicate with

testing tools, and a section layout dictionary that provides information on the structure of the file.

The link editor is capable of combining a collection of object files into a single object by resolving interprogram symbol references and binding symbols to virtual or absolute addresses as specified. The object file for the 3B20D Processor may contain multiple text and data sections. This facility is used in the creation of special process files and libraries.

Process files for the DMERT operating system are generated using a special tool that uses the link editor to create special data sections used in process communication and entry. All kernel, supervisor, and special DMERT processes require use of this tool as a final construction step. User level processes require only the normal link edit step for completion.

4.3 Building a view of a subsystem

The object construction tool, build, (based on the *UNIX* operating system make command), exists to create objects according to a previously specified set of object construction commands and a list of dependencies. This specification is used to identify which software components are needed to construct others.

Using modification time-stamps provided by the file system, build determines which objects are current and which need to be rebuilt. When an object needs to be rebuilt, its specification must be examined, and so on down the line until all necessary objects are rebuilt or current. These object construction instructions constitute a makefile for the particular DMERT subsystem.

The advantage that build offers over make is that the developer in a private development node need only have copies of the source being changed and can access the remainder of the objects, header files, or C or assembler source files from other nodes as specified in the developer's *viewpath.* A viewpath is an ordered set of directory names indicating the nodes to be searched for missing software components. The viewpath is an extension of the *UNIX* operating system *searchpath* variable that contains a list of directories containing commands. By specifying a viewpath, a developer can create a desired view of a subsystem product to be tested. The view may be, for example, that of the developer's changes integrated with only officially approved software or with any other software currently submitted to the test team.

4.4 Preparation for testing

Once a subsystem has been changed in response to an MR within a developer's node and is ready to be taken into the laboratory for

testing, there are listings that can be prepared to aid in the software testing process. A set of SGS utilities exist to generate these listings. A C program breakpoint source listing contains a source listing of all the functions in a file augmented by line numbers indicating C source lines at which a breakpoint can be set during testing. This listing is important due to the fact that object code may have been significantly rearranged during the optimization step of the compiler rendering certain C source lines shuffled in a semantically equivalent, but non-obvious, fashion. The listing helps a developer in a test laboratory who needs to correlate high-level C source to 3B20D Processor assembly code.

A namelist utility creates a listing of all the C language symbols residing in the object file's symbol table with their addresses and types. Not all symbols associated with an object need reside in the symbol table since they can be removed during object creation by another SGS utility.

A developer can obtain assembly source listings associated with an object file in either of two ways. The listing can be generated by the compiler during compilation or it can be created by a *disassembler* from the object file. This latter listing will contain C program source information in terms of symbols, labels, and line numbers if they have not been stripped from the object. This listing can be of particular interest after software has executed on the machine to determine where text and data may have been accidently altered by a process out of control in a development laboratory.

With these listings augmenting the original source files, the developer is prepared to enter the testing environment.

4.5 Submitting changed files for approval

When all desired changes have been made and unit or subsystem testing has convinced the developer that the modification request has been satisfied, the files extracted from the official CMS data base (and any files newly created as a result of the MR) are submitted to the test team through the use of CMS commands (refer again to Fig. 1). The activity of the developer on the files in question can be temporarily or permanently suspended by other CMS commands that place the changes made into the CMS data base and allow other developers access to the same files for editing if necessary.

The software approval process in the SDS is initiated when the developer submits an MR-related software change for inclusion in the next release. The developer submittal begins a three-step approval process:

(*i*) Independent Certification—The source change is used by administrators to reconstruct the changed products. This is done in such

a way as to not affect anything currently approved or under test. The changed products are then independently tested (certified) by a member of the project's system test group. Certification failure implies rejection of the MR and further work for the developer.

(*ii*) Integration—Those changes passing certification are incorporated into the under-test view where they are integrated with the rest of the under-test changes. This integration is performed by members of the project's integration group. Failure during integration also implies MR rejection, as well as recertification following additional development.

(*iii*) Approval—When all certified changes have been integrated and soak tested, the under-test view will be approved. This implies releasing a software update to 3B20D customers, updating all the nodes, and approving MRs.

V. DEVELOPMENT ENVIRONMENT AND STANDARDS

The software development system must establish a balance between the flexibility given to developers and administrators to enhance their productivity and restrictive standards so that administration is manageable and effective. Considerable development flexibilities already have been identified. Among these are:

(*i*) The capability to select a particular version of a source file by specifying one or more MR numbers whose related source changes are applied to the last released version of the source file.

(*ii*) The use of any editing facilities once a source file has been obtained from CMS.

(*iii*) The C language, which encourages programmer optimization and functional modularization.

(*iv*) The protection from concurrent source changes by independent developers ensured by CMS.

(*v*) The independent, private development node allowing experimentation that does not interfere with other development activity.

(*vi*) The flexible capabilities for view construction provided by build.

(*vii*) The ability administrators have through CMS to back out changes associated with particular MRs and return to previous software states.

The software environment and its SDS tools also impose an important set of standards on development activities that guarantee safe, productive, and orderly administration of all change activity.

5.1 Host machine configuration

The host machines used in software development and administration are each standardized with respect to the activities that occur on the

machines. This standard ensures that all development activity can be monitored adequately and that the proper tools can be made available to those needing them. With a fixed set of machines, updates to the software tools can be synchronized making sure that all development is compatible.

The 3B20D Processor software development system resides on a network of eight general-purpose, 16-bit minicomputers, all of which run the *UNIX* operating system. The functional configuration of these eight machines is shown in Fig. 3.

(*i*) Developer activity is confined to the six machines labeled SD (software development). Here developers utilize CMS, the SGS, and the build facility, as well as the standard tools available with the *UNIX* operating system.

(*ii*) The machine labeled SC (software control) is for administrative use only. This machine contains the modification request data base system from which all change activity is controlled.

(*iii*) The SP (software production) machine is an administrative machine. All official product construction takes place on this machine.

The test environment combines a 3B20D Processor and a support machine running a real-time version of the *UNIX* operating system as its support processor.

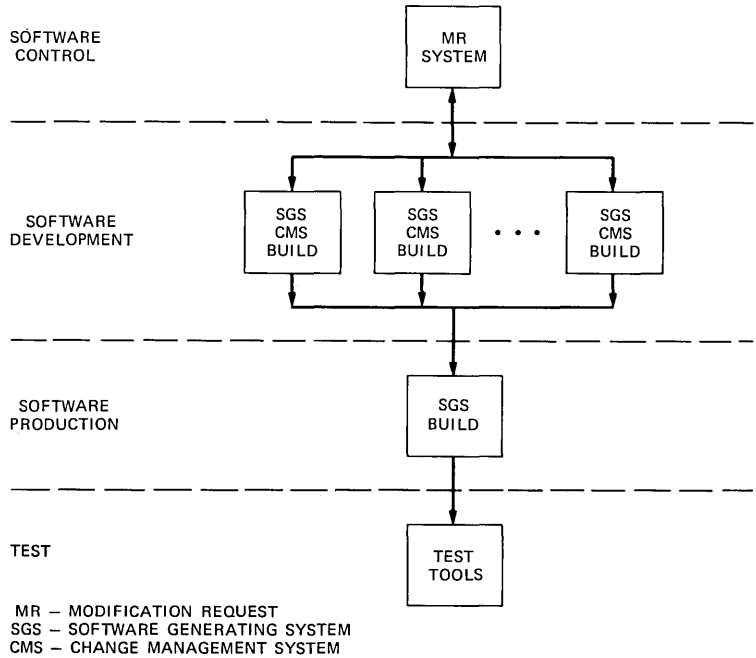


Fig. 3—Host machine configuration.

5.2 Software configuration

A standard *UNIX* operating system directory structure configuration has been established for the 3B20D Processor software subsystems. This configuration is repeated in each CMS node so that file access and object construction techniques are fully repeatable no matter where the view building activity takes place.

The 3B20D software is logically partitioned into distinct subsystems. Typical subsystems include the operating system, diagnostics, and peripheral unit drivers. Software development for the subsystems is evenly spread over the six SD machines described in the previous section. A single subsystem is fully contained on one of the SD machines. Each SD contains the software development activities for one or more subsystems.

The only machine on which the full directory structure combining all of the subsystems exists is the SP machine where official load construction takes place. Subsystem development can take place on the individual SD machines, independent of other subsystems, because the following software construction standard is enforced: the only source files shared between subsystems during load construction are global header files (common definitions required by more more than one subsystem) or global libraries (collections of common object modules).

Global header files and global libraries have fixed, standard positions in the SD machines' directory structure. These shared resources are distributed from the SP machine to the various SD machines after validation by administrators. In this way, all SD machines are kept synchronized with respect to this critical data.

Other top level directories or directory structures are used during the load-construction process as installation points for revised tools, tool usage descriptions in manual page format, and 3B20D Processor core software components.

5.3 Interactions between CMS and the MR system

The major strength of CMS and the MR system lies in the standardization of source-change administration. The following major standards are enforced with these tools:

(*i*) Every incremental change is tagged with an MR number.

(*ii*) The developer cannot modify official source until an MR has been assigned to him or her by an administrator. This assigning capability can be used by project administrators to funnel developer activity.

(*iii*) Once a developer is satisfied with a change, it is submitted against the corresponding MR for independent verification. Once submitted, no further source changes for the MR are allowed.

The MR data base system on the SC machine and the various CMS instances, each identified via a subsystem name, on the six SD machines are connected in the SDS via a high-speed network and a remote job-execution facility. Specific actions on the SC machine cause remote jobs to be sent to one of the SD machines and executed for a particular CMS instance. The reverse is also true. The major triggers interconnecting the MR system and the CMS instances are:

(*i*) When an MR is assigned to a particular subsystem for a particular generic, a remote job is sent to the SD machine on which that subsystem's development is based. When the job is executed on the SD machine, a sequence of CMS commands is executed that cause the MR to be assigned to the particular generic in the correct CMS instance. The developer can then officially edit the source files to effect the change.

(*ii*) When the developer is satisfied with changes, they are submitted for verification. The CMS status is set to *submitted* to prohibit any further editing for this MR, and a remote job is sent to the MR system on the SC machine. This job will cause the MR status for the particular generic and subsystem to change to *submitted.*

(*iii*) A remote job to reject an MR can be sent to CMS on an MR status change of under-test to assigned or submitted to assigned. In CMS, the MR would again be available for editing with a subsystem status of being-fixed.

These are the only MR/CMS interconnections necessary. All other control functions require only MR system actions.

5.4 Product construction standardization

A major ingredient of the 3B20D Processor SDS is the standardization of the format and content of the makefiles used by build. The standards are enforced by allowing developers to create only a skeleton makefile and providing a makefile generator to produce the full makefile. The advantages accrued by standardized makefiles are the guaranteed correctness and completeness of object dependency lists and the ability to create cross-reference listings of dependencies to determine changed file impacts.

VI. SUMMARY

Software development for the 3B20D Processor is administered with an important set of standards and a powerful set of tools to track development efforts and enforce these standards. The developers have at their disposal a flexible system for the generation of C programs and utilities to adequately prepare for debugging and efficiently construct object programs. The resulting 3B20D Processor software development system strikes a critical balance between offering freedom

and flexibility to programmers while managing and monitoring orderly software change procedures.

REFERENCES

1. M. J. Rochkind, "The Source Code Control System," IEEE Trans. Software Eng., *SE-1* (December 1975), pp. 364–370.
2. J. R. Kane, R. E. Anderson, and P. S. McCabe, "3B20D Processor & DMERT Operating System: Overview, Architecture and Performance of DMERT," B.S.T.J., this issue.
3. H. D. Rovegno, "A Support Environment for MAC-8 Systems," B.S.T.J., *57,* No. 6 (July–August 1978), pp. 2251–64.
4. S. C. Johnson, "A Portable Compiler: Theory and Practice," Conf. Rec. Fifth Annual ACM Conference on Principles of Programming Languages, Tucson, Arizona, January 23, 1978, pp. 97–104.
5. S. C. Johnson, and M. E. Lesk, "Language Development Tools," B.S.T.J., *57,* No. 6 (July–August 1978), pp. 2155–77.
6. R. Sethi and J. D. Ullman, "The Generation of Optimal Code for Arithmetic Expressions," J. ACM, *17,* No. 4 (October 1970), pp. 715–28.
7. A. V. Aho and J. C. Johnson, "Optimal Code Generation for Expression Trees" J. ACM, *23,* No. 3 (July 1976), pp. 488–501.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

Overview, Architecture and Performance of DMERT

By J. R. KANE, R. E. ANDERSON, and P. S. McCABE

(Manuscript received March 22, 1982)

A process-oriented operating system, the Duplex Multiple Environment Real Time (DMERT) operating system, was designed for the 3B20D Processor and offers both real-time and time-shared operation, with emphasis on high availability. The design objectives and architecture of the DMERT operating system and an explanation of how the system performance is characterized are presented. A companion article describes in depth the DMERT operating system design.

I. INTRODUCTION

The direct predecessor of the 3B20D, the 3A Processor, included a real-time monitor known as the Extended Operating System (EOS).[1] Experience with EOS demonstrated that applications could develop their software with less effort and that synergy resulted between the hardware and software developers. Hence, the hardware is optimized to support the software and the software, in turn, uses the hardware in the most effective manner. The success of EOS led to the decision to support 3B20D applications with a more extensive operating system than EOS. The operating system that resulted is known as the *D*uplex *M*ultiple *E*nvironment *R*eal *T*ime (DMERT) operating system.

The basic architecture of DMERT is based on an earlier system named MERT,[2] which was derived from the *UNIX** operating system.[3] Both the *UNIX* and the MERT operating systems were origi-

* Trademark of Bell Laboratories.

nally developed to execute on commercial equipment; today *UNIX* operating systems are used widely for time-sharing on a variety of computers. The "D" in "DMERT" reflects one of the characteristics that distinguishes it from the previous two operating systems, namely that DMERT is designed to execute on a *duplex* 3B20 Processor. Thus, the DMERT architecture draws upon concepts from EOS, MERT, and *UNIX* operating systems.

The applications using the 3B20D Processor have been described in detail in a previous article.[4] Notice that while different, they have several common characteristics. First, a major component of the application is software. Second, the major mission of this software is real-time oriented with response times as short as several milliseconds. Third, each application has a need for continuous operation 24 hours a day, 7 days a week and hence stringent processor availability requirements. Fourth and finally, each application is to be operated over a long period of time, which requires extensive software for monitoring and reporting on system status as well as changing and upgrading the system while it is in operation.

This paper describes the development objectives of DMERT, which were chosen to satisfy the above application characteristics. The operating system architecture used to achieve these objectives is next described. Finally, this article describes the performance characterization of DMERT. The design details of DMERT are presented in depth in the next article[5] in this Journal.

II. DMERT DESIGN OBJECTIVES

2.1 Support multiple real-time applications

It is necessary for the DMERT operating system to support many applications, each with different real-time demands. Some applications include data bases that need many disk jobs serviced quickly. Others control telecommunication equipment requiring rapid response to an event such as an interrupt and dedicated processing capacity for an interval thereafter. To satisfy these diverse needs, a design objective was established to provide modularity in the operating system to allow a high degree of application tailoring.

2.2 Improve application development productivity

The real-time applications of the 3B20D Processor often have major software components that are not time critical. The "rule of thumb" stating that 80 percent of the time is spent on 20 percent of the software generally applies to these applications. Hence, a design objective of DMERT was to support a feature-rich operating system environment for the non-time-critical software while retaining real-time responsiveness for the rest.

To increase productivity of the developers, an objective of efficiently supporting a programming language at a level substantially higher than assembly language was established. (See the previous article[6] for further details.)

To allow technology upgrading of the 3B20D Processor without impacting the application software, an objective of shielding application programmers from hardware implementation details, such as memory protection[7] and I/O devices,[8,9] was established.

2.3 Error tolerant design

To meet the reliability objectives of the 3B20D Processor, it is necessary to support software packages for error checking and recovery. Some of these are described in subsequent articles.[10–12] To reduce the complexity of both the operational and recovery components of the system, a design objective was established to separate recovery software from the core of the system.

An objective of incorporating extensive internal consistency and integrity checks within all software components was established to ensure that critical software modules protected themselves from errors in other parts of the system.

III. DMERT ARCHITECTURE

3.1 Processes

One of the basic design goals for DMERT was to build modular and independent processes, each having localized data known only to itself. Hence, the notion of a process is fundamental to the DMERT architecture, which is essentially composed of a kernel and a collection of cooperating, concurrent processes. The following sections define what a process is, how processes communicate with each other, and how they are used by DMERT applications.

3.2 Definition of a process

A process is a collection of related, logical segments (programs and data) that can be brought into memory to form an executable entity. A segment is the basic memory entity in DMERT. A segment is composed of 1 to 64 pages, each 512 32-bit words in length. Segments can grow dynamically in increments of a page. A process consists of at least three segments: code or text, a stack used for temporary data, and a special type of data segment called a process control block (pcb). The pcb segment contains unique information that identifies the process to the operating system. This information includes the process number, type of process, priority, and address space qualifiers that define the virtual address space for a process. Each process has its own virtual address space of up to 128 segments. These virtual addresses

are mapped to physical addresses by 3B20D hardware under the control of the DMERT operating system.

A process can be dynamically created to perform a set of functions and then to gracefully terminate itself when its task is finished. Processes that continuously perform work remain "alive" at all times; however, they may sleep or be inactive until an interrupt fires. These features allow main memory to hold only the processes necessary, at a given point in time, to support the application.

3.3 Process types

DMERT has four basic types of processes: kernel, kernel process, supervisor, and user. DMERT may be viewed as a hierarchy of virtual machines, where successive levels put additional restrictions on access rights and further remove the programmer from the details of the physical machine. However, the higher levels may take advantage of services provided by the lower levels. In general, the higher the level, the more services available to the application programmer; the lower the level, the more real-time-efficient the program execution. This level structuring of virtual machines permits DMERT to manage real-time applications, while at the same time providing the flexibility of a time-sharing system for background tasks. This approach avoids contention for system resources with the high-priority tasks and simplifies the implementation effort for lower priority tasks.

3.3.1 Kernel

The DMERT kernel provides the most primitive virtual machine. The kernel handles hardware interrupts, timer interrupts, and operating system traps. In all cases, the kernel saves the state of the interrupted process, provides whatever service is requested, and then restores the state of the interrupted process. The kernel services are fairly primitive but they execute efficiently.

Also part of the DMERT kernel are special processes that provide scheduling, memory management, and other services. For example, the memory manager and the scheduler are two of the special processes in DMERT. The memory manager loads processes into main memory, selects segments to be swapped out to disk when additional main memory is required, and provides routines that may be called by the kernel. The scheduler controls the execution of time-shared processes, that is, supervisor and supervisor-user processes. Special processes behave as kernel processes, except that they do not have their own virtual address space, but rather reside in the kernel's address space. These special processes communicate with the kernel through function calls instead of operating system traps, and they have access to global system data.

3.3.2 Kernel processes

Kernel processes comprise the next virtual machine layer in DMERT. They are completely interrupt-driven and are designed to provide time-critical processing in a real-time environment. Kernel processes have their own virtual address space. However, they share the kernel's stack and the kernel's message buffer segment. Swapping is not necessary with kernel processes because their segments are locked in memory to ensure rapid response to events such as interrupts. The various peripheral device drivers and the file manager, which implements a hierarchical file system, are examples of kernel processes.

At process build time, kernel processes are set up to share the operating system's stack and message buffers. This design was chosen for quick access to arguments of operating system traps and fast message communications between processes. Since kernel processes use the kernel's stack, they must run until they complete their task and then return control to the kernel.

3.3.3 Supervisor and user processes

Supervisor processes form the third layer of virtual machine. These processes can use all the services provided by the kernel and kernel processes. Supervisor processes provide time-sharing services that can be considered background tasks. They share the real time of the processor with each other according to priorities administered by the scheduler, which is a special process. In general, supervisor segments are not locked in memory, but can be swapped out. Thus, supervisor processes take much longer to dispatch than either special or kernel processes.

Supervisor processes can be designed to run "stand-alone" or they may be used to implement a fourth virtual machine layer called user processes. The DMERT process manager is a supervisor process that does not support a user level. However, the *UNIX* supervisor provides a user environment identical to that seen by a *UNIX* program. This is done through code at the supervisor level that calls upon the services of lower virtual machine layers. DMERT can simultaneously support multiple supervisors, each supporting its own user processes. It should be noted that while DMERT treats a supervisor/user combination as a single process with a dual address space, both levels exist conceptually. Also, supervisor-level code executes more efficiently than user-level code because a supervisor has direct access to the lower level primitives, while the user interface to these primitives is coordinated by the user's supervisor.

3.3.4 Interprocess communication

DMERT provides a rich set of interprocess communication and

synchronization mechanisms including messages, events, interprocess traps, and shared memory. These interprocess communication primitives are fundamental to the DMERT structure. Most of the system services are requested by an exchange of events and messages between a requesting process and either a system process or the kernel.

3.3.4.1 Messages. Processes are in general separate and distinct entities. Two processes working together on a task must be able to exchange information. To satisfy this need messages may be sent from any level process to any other level process. These messages can be up to several hundred bytes long. The sender need only know the target process number and a pre-agreed format of the message. An optional acknowledgment capability is provided so the sender can synchronize actions with the receiver.

3.3.4.2 Events. Communications between processes may occur using an event mechanism. An event is a single bit that is set by DMERT or a process and can be interrogated by the receiving process. Presently, 32 bits are available of which the DMERT operating system reserves 16 bits for its use. Thus, two or more processes can communicate internal states using events.

3.3.4.3 Interprocess traps. A mechanism exists in DMERT to allow a lower-level process to support a higher-level process. A user-level process may trap to a supporting supervisor and a supervisor may trap to a kernel process. Trapping implies a transfer of control from one process to another with the passing of input parameters to the target process. The lower-level process returns status and control back to the trapping process after it has completed the required support work.

3.3.4.4 Shared memory. Processes are built with a view of their own virtual address space and in general cannot access any other process's process's address space. This affords protection; however, sharing large amounts of data is difficult. Cooperating processes that must exchange information at rates higher than those supported by messages or events can share segments. A shared segment is a part of the virtual address space of several processes simultaneously. The application must control access to the shared data.

3.4 Multiple environment support of applications

DMERT simultaneously supports both a real-time and a time-sharing philosophy. Kernel and kernel processes operate in a real-time environment and have first call on the available real time of the 3B20D Processor. The remaining time is shared among supervisor and user processes.

Most telecommunication applications build their own virtual machine or "operating system" as a kernel process that can respond to

the time-critical stimuli characteristic of telecommunications. This kernel-process approach also allows fine tuning of the telecommunications operating system, independent of the DMERT operating system. In at least one case, the application specific virtual machine is a kernel process that runs in emulation mode. Being a microcoded machine, the 3B20D Processor can efficiently execute another machine's instruction set. By using the emulation mode, existing debugged application code, including operating systems, can be carried forward to the 3B20D Processor and DMERT with little additional software development effort.

Some applications spread their functions over the existing DMERT virtual machines. For example, the time-critical functions related to disk and data link usage are implemented as kernel processes, and the administrative and postprocessing functions are implemented as supervisor and user processes. An application of this type is normally used as a data base management system and/or a data communications system. Generally, an application fine tunes its system by moving processes to different execution levels within the virtual machines.

IV. PERFORMANCE

The 3B20D/DMERT system is capable of providing computing services in a stand-alone mode; however, usually it is utilized by surrounding it with application hardware, firmware, and software. The application hardware may include additional units identical to those already a part of the processor complex (e.g., additional memory, disk, data links), or it may include hardware unique to the application system (e.g., bus controllers, or time- and space-division switches). The application software frequently includes drivers, schedulers, and fault-recovery facilities, as well as the more usual "application programs." As a result of this diversity of software interfaces to DMERT, the performance modeling and measuring of the application system requires an extensive performance characterization of DMERT, rather than the more traditional benchmark approach used in general-purpose, computer-performance evaluation. The following sections describe the approach taken and the type of performance data made available to DMERT applications.

4.1 Performance characterization

Since the application software may interface DMERT at all levels of the virtual machine (hardware, firmware, and the software levels of kernel, kernel process, supervisor, and user process), the performance characterization of the system must include data for all of these levels. In addition, the application can make use of a variety of DMERT system resources such as memory, peripheral devices, message buffers,

etc, and hence these resources must be accounted for in the performance characterization as well.

The 3B20D performance characterization addresses four areas: the operating system, input/output, DMERT services, and DMERT overhead. Each of these areas will be discussed in more detail, and the sum of these areas covers all significant aspects of DMERT performance, as well as providing some models for the application developers to use in the analysis of the application system performance.

4.1.1 Operating system characterization

The goal of the performance characterization of the operating system was to identify the cost in resources for every service available at every level. The predominant service interface in DMERT is the Operating System Trap (OST), and hence every significant OST in DMERT was characterized with respect to its central processing unit (CPU)-time at the various modes and execution levels available in DMERT.

The kernel, supervisor, and user OSTs cover a broad range of DMERT services:

(*i*) *Timing:* clock reading and setting, single and repetitive timeouts, and process sleeping requests.

(*ii*) *Memory management:* locking and unlocking of memory segments, growing and shrinking of memory segments, and swapping of memory segments.

(*iii*) *Scheduling and interrupting:* protecting against interrupts (critical region), priority changing, and fielding of interrupts.

(*iv*) *Interprocess communication:* interprocess message sending and receiving, and sending and fielding of interprocess events.

(*v*) *Process switching and other functions:* switching from one process to another, changing the execution level of a kernel process, creating another process, faulting another process, routing and rerouting of standard inputs and outputs.

4.1.2 Input/output system

The input/output (I/O) system of DMERT is characterized from an *internal* point of view; that is, each of the kinds of I/O services are characterized with respect to their primary resource consumption. The various I/O services are all measured with respect to the CPU-time consumed for each transaction and the maximum throughput rate based upon the elapsed time for each transaction.

A basic set of operations can be performed on most I/O devices: open, close, read, and write. Seeking is a disk-only service, and rewinding is a tape-only service. Except for data links, read and write operations can be invoked in several ways, depending on the physical organization of the data on the device. Most file operations can be

invoked directly or through the file manager. The devices supported by DMERT include disks, tapes, terminals, and data links. Finally, the disk I/O services can be invoked in several modes: normal (synchronous, buffered I/O), asynchronous (which allows the invoking process to continue running while the I/O request is being serviced), physical (no buffering constraints, or services) and synchronous writes (which guarantee an immediate write to the disk, rather than the potential delayed write possible in the normal mode).

Most of these I/O services are available to kernel, supervisor, and user processes via separate OSTs. Each OST is characterized with respect to CPU-time and maximum device throughput for each of the applicable cases.

4.1.3 DMERT applications services

Application software may utilize high-level services from DMERT as well as the more primitive OST-invoked services. The Craft Interface[12] is an example of this general category covered by the term DMERT services. The Craft Interface system provides an extensive and sophisticated set of terminal-oriented facilities that are used by both DMERT and the application software. Additional examples of DMERT services are the diagnostic and audit facilities that are a part of DMERT, but also may be invoked by application software.

Of these DMERT services, the Craft Interface has been characterized for performance owing to its importance to early DMERT applications. The performance characterization chosen was to measure the CPU-time usage of the three most important application services: the Program Documentation Standard (PDS) Shell, the Control and Display Administrator, and the Output Spooler. Each of these services was measured with a range of appropriate job sizes.

4.1.4 DMERT overhead

The final category in the performance characterization of DMERT is the system overhead. While system overhead is not invoked explicitly as a service, it provides the essential services of the operating system. There are several types of system overhead:

(*i*) *Functional:* provides for the capability of a multi-environment, real-time operating system by handling the timing-based facilities for interrupt servicing, scheduling, craft terminal polling, and data link polling.

(*ii*) *Sanity:* provides for sanity and overload monitoring by resetting hardware sanity timers, monitoring DMERT and application sanity timers, and checking for process lock-out conditions.

(*iii*) *Preventive maintenance:* provides for routine preventive maintenance exercising and running routine software audits.

(*iv*) *Fault maintenance:* provides for fault detection, location and recovery, including removing faulty units from service, and testing and restoring replacement units. This overhead is incurred only when a fault occurs.

(*v*) *Services:* provides for other non-electable services not covered by the previously described types, such as Craft Interface services and Plant Measurements services invoked in providing the processor system.

The DMERT operating system overhead is characterized in terms of CPU time and is expressed as a percentage. The first two types of overhead (functional and sanity) constitute the "continuous" overhead seen by the application, and cannot be controlled or throttled by the application. This component of the overhead is less than 5 percent for DMERT. The preventive maintenance overhead can be controlled in two ways: routine software audits can be throttled so that their peak resource usage is limited to a desired value, and the routine diagnostic exercising can be scheduled during times of light load. The fault maintenance overhead is measured as a single-fault, worst-case scenario for the fault detection, isolation, and recovery, as well as the testing and restoral, of the repaired unit. The total resource usage is averaged over the specified two-hour repair interval. The services overhead includes the normal administrative activities necessary to maintain and administer the processor complex. The total system overhead for functional, sanity, preventive, and fault maintenance and services is less than 15 percent for DMERT.

4.2 Performance measurement techniques

The key resource in the 3B20D/DMERT system, and in the applications systems built upon it, is CPU time. It is shared among many processes, both DMERT and application, in four execution modes, at 16 execution levels and 256 priority levels. The sharing is interrupt driven, with preemption from processes at higher priorities and at higher execution levels. While it is relatively simple to measure CPU time in an overall sense, it is a very complex job to measure the CPU time used by a particular process, function, or service. To solve this problem, the DMERT kernel was instrumented. Finally, hardware monitors were used to measure the performance of the processor complex, and also to verify the correctness and accuracy of the software performance measurement instrumentation.

4.2.1 Kernel instrumentation

The DMERT kernel was extended to provide detailed accounting of the CPU time usage (by execution mode and level) for the system as a whole, and also for each process. To keep the overhead low enough

to allow the instrumentation to be a permanent part of DMERT, a statistical sampling approach was used. Each 10 milliseconds the kernel records the process currently executing, together with the execution mode and level, assigning the previous 10 milliseconds to that process. Over a reasonable period of time (seconds or more) these statistical sampling results will converge arbitrarily close to the actual CPU time usage.

V. SUMMARY

An overview has been given for the objectives of DMERT based on its goal of providing a high-reliability, real-time processor system for telecommunications applications. This overview has indicated some of the development objectives of DMERT and has given the approach for the performance characterization of the whole system. Also included is a description of the various kinds of operating system overhead, including measured values for DMERT, and a description of the performance measurement instrumentation within the DMERT kernel.

VI. ACKNOWLEDGMENTS

Many members of Bell Laboratories' staff contributed to the successful introduction of DMERT. Key contributors to the work described in this article and not mentioned as authors or references were: S. F. Ho, P. C. Kao, T. K. Liu, T. M. Raleigh, E. P. Schan, D. S. Trushin, and S. M. Udupa.

REFERENCES

1. C. H. Elmendorf, "Meeting High Standards with the Extended Operation System," *Bell Lab Rec.,* (March 1980), pp. 97–103.
2. H. Lycklama and D. L. Bayer, "The MERT Operating System," B.S.T.J., *57,* No. 6., Part 2 (July 1978), pp. 2049–86.
3. D. Ritchie and K. Thompson, "The *UNIX* Time Sharing System," B.S.T.J., *57,* No. 6, Part 2 (July 1978), pp. 1905–29.
4. R. W. Mitze and W. C. Schwartz, "The 3B20D Processor & DMERT Operating System: 3B20D Processor and DMERT As a Base for ESS Applications," B.S.T.J., this issue.
5. M. E. Grzelakowksi, J. H. Campbell, and M. R. Dubman, "The 3B20D Processor & DMERT Operating System," B.S.T.J., this issue.
6. B. R. Rowland and R. J. Welsch, "The 3B20D Processor & DMERT Operating System: Software Development System," B.S.T.J., this issue.
7. M. W. Rolund, J. T. Beckett, and D. A. Harms, "The 3B20D Processor & DMERT Operating System: 3B20D Central Processing Unit," B.S.T.J., this issue.
8. R. E. Haglund and L. D. Peterson, "The 3B20D Processor & DMERT Operating System: 3B20D File Memory Systems," B.S.T.J., this issue.
9. A. H. Budlong and F. W. Wendland, "The 3B20D Processor & DMERT Operating System: 3B20D Input/Output System," B.S.T.J., this issue.
10. R. C. Hansen, R. W. Peterson, and N. O. Whittington, "The 3B20D Processor & DMERT Operating System: Fault Detection and Recovery," B.S.T.J., this issue.
11. J. L. Quinn and F. M. Goetz, "The 3B20D Processor & DMERT Operating System: Diagnostics," B.S.T.J., this issue.
12. M. E. Barton and D. A. Schmitt, "The 3B20D Processor & DMERT Operating System: Craft Interface," B.S.T.J., this issue.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

DMERT Operating System

By M. E. GRZELAKOWSKI, J. H. CAMPBELL, and M. R. DUBMAN

(Manuscript received March 17, 1982)

General operating system services in Duplex Multiple Environment Real Time (DMERT) are provided by a kernel and a set of cooperating processes. These services support a multitude of process-oriented features that include a rich set of interprocess communication mechanisms and a sophisticated collection of memory manipulation primitives. The operating system provides processes with two scheduling alternatives and various creation and termination options. The cooperating processes include input/output drivers and device handlers that enable processes to communicate with a variety of peripheral devices. Another set of processes simulates a UNIX™ *operating system and file system.*

I. INTRODUCTION

This article describes the design of the DMERT operating system nucleus. The reader should be familiar with the basic architecture of Duplex Multiple Environment Real Time (DMERT), which is described in a companion paper.[1] The operating system nucleus is composed of: the kernel, which supports interprocess communication mechanisms, the system clock, and interrupts; the special processes, which perform memory management and scheduling; two supervisor processes, which handle process management and the *UNIX** operating system environment; and three kernel processes that control communication with peripherals and the file system.

In general, DMERT applications view the operating system nucleus

* Trademark of Bell Laboratories.

as one entity rather than a set of cooperating processes. Thus, the interrelationships between these processes are hidden from them. This article describes the design of DMERT's nucleus from the user's perspective; that is, it is feature oriented rather than process oriented. After finishing this paper, however, the reader should be familiar with both the feature set and the internal design of the cooperating processes.

In order for a process to use any of the operating system's features, it must interface to the operating system with Operating System Traps (OSTs) or Interprocess Communication (IPC) mechanisms. OSTs were discussed in Ref. 1, and Section II describes IPCs. Sections III, IV, and V define the image and life cycle of processes by describing how memory is handled, the scheduling options, and how processes are created and terminated. Section VI summarizes the characteristics of a special type of process, namely a user process. Section VII discusses the input/output (I/O) and disk interfaces, and Section VIII describes DMERT's file systems.

II. INTERPROCESS COMMUNICATION

As stated in the previous paper, DMERT provides an extensive set of interprocess communication and synchronization mechanisms, including messages, events, process ports, faults, interprocess traps, first in-first outs (FIFOs), and shared memory. These interprocess communication mechanisms are described below.

2.1 Events

An event is a one-bit piece of information that may be sent, received, and recognized by a process. DMERT has 32 event bits, several of which are reserved for special meanings by DMERT. Those not reserved may be used in any manner agreed upon by a set of cooperating processes. DMERT provides several event-type OSTs that enable one process to send an event to another process or to a class of processes.

Processes have multiple entry points for start, event, OST, and fault entries. The purpose of the event entry is to allow the system to direct the process's attention to an event that has just occurred. Thus, when any of the process's event bits are set, control is passed to its event entry.

Additional control over events is given to a kernel process. It has the ability to mask and reenable the bits that comprise its event flag word. This gives a kernel process the capability to more precisely control its flow by only allowing certain communications to occur.

To provide timely response to events, the process receiving the event is interrupted and the state of its activities is saved by the kernel. After

the event is acted on, the process is restored to its original state and continues from the point where it was interrupted.

2.2 Messages

Messages are the primary method of communication between cooperating processes in the operating system. The DMERT kernel provides a variety of OSTs for allocating, sending, receiving, canceling, auditing, and freeing messages for individual processes.

A message consists of a fixed-size header followed by a variable-size block of data, called the message body. The header contains routing directions, status indicators, and other administrative information. The message body's contents and structure are determined by the cooperating processes using the message.

A reserved event is used to notify a process that it has received a message. Hence, when a process receives a message, control is passed to its event entry. Waiting messages are queued in a message buffer segment in the kernel's address space and are directly accessible by kernel processes. However, kernel processes do not manipulate messages directly without first dequeueing them by using one of the message dequeueing OSTs. Supervisor processes do not have direct access to the message buffer segment. Messages must be copied into and out of this segment from and to the supervisor's own space.

2.3 Ports

Communications by messages require the knowledge of the target process's Process Identifier (PID). Since processes can be created and terminated dynamically, PIDs also are generated dynamically and a process typically knows only its own PID. Process ports permit processes to communicate with each other without knowing each other's PID. A process port is a globally known "device" to which a process may attach itself for receiving messages. Other processes may communicate with a process connected to a port by sending a message to that port rather than to a specific PID. Thus, process ports permit unrelated processes to communicate with each other.

2.4 Faults

Faults are very similar to events. A fault is a one-bit piece of information that may be sent, received, and recognized by a process. DMERT has 32 fault bits, some of which are reserved for special meanings by DMERT. When a process's fault bits are set, control is passed to its fault entry.

Faults are used to inform processes of error and overload conditions and take precedence over events. If a process has both events and faults pending, control will be passed to its fault entry first.

2.5 Interprocess traps

As mentioned in Ref. 1, an OST is generally used to transfer control to the kernel to perform some service on behalf of the requesting process. As such, an OST does not constitute a form of interprocess communication. An interprocess trap is a generalized notion of the OST, where a process can trap to some other process rather than the kernel to have some service performed. Because this interprocess trap can communicate service requests or other data between two processes, it is a form of IPC.

Interprocess traps are more restrictive than other IPC mechanisms. A process can trap to another process only if the former process's execution level is lower than the latter's. Control is passed to the trapped process's OST entry.

2.6 FIFOs

Unrelated processes can use FIFOs for character stream communication. Conceptually, a FIFO is a queue of characters; characters flow through in a first in-first out order (hence, the name). The queue is fixed in length and may become filled if writers write faster than readers read. When such a condition occurs, further writes are prevented. Similarly, reads are prevented when the queue is empty.

Each FIFO has a name; that is, each is a special file in the file system (described in Section VIII). This naming characteristic distinguishes it from a pipe,[2] because only processes spawned by a common ancestor can communicate using a pipe. Naming allows any process that knows a FIFO's name to use it, regardless of its ancestry.

Any process level may use this mechanism. The FIFO is a suitable replacement for messages where communication is a character stream.

2.7 Shared memory

A segment is the basic unit of shared memory. It can be shared between different processes, or between multiple instances of the same process. A segment also can be shared among any number of arbitrary processes by assigning a global name to it and allowing any process to access it that has been given the global name.

Since it is completely managed by the cooperating processes, shared memory is the form of interprocess communication most susceptible to error, but it is also the most efficient. Within the operating system, shared segments are used wherever reliability will not be sacrificed and where real-time response is paramount.

III. MEMORY MANAGEMENT

DMERT memory management centers around the segment. A segment is a set of logically related pages. A page is 2048 bytes of

contiguous main memory that always begins on addresses that are multiples of 2048. A segment is a collection of pages that do not have to be physically contiguous. All segments in the system are unique, that is, no page can belong to more than one segment. (Reference 3 contains a detailed description of pages, segments, and memory translation.)

3.1 Segment attributes

Segments are created in main memory by the operating system on demand and disappear when they are no longer needed. When the initial image of a process is created, its segments are loaded into main memory by the memory manager. Although a segment of a user or supervisor process can be swapped out to a swap area reserved on the disk if additional main memory is needed for a process of higher priority, the segment remains known to the system until it is not required by any process.

A segment is identified internally by its Segment Identifier (SID), which is the virtual address of its entry in the kernel's segment description table. A segment of a supervisor process also can be referenced by a segment number that indexes into the process's segment list. The segment list specifies the segments that are known to the process, including those that are not necessarily in the current virtual address space of the process.

A process has segments that may be private or shared by other processes. Text segments are shared when possible to avoid duplication of commonly used functions. Data segments are usually private since most processes want sole access to their own data. However, a data segment can be shared with another process for implementing interprocess communication (see Section 2.7) and can have a global name. Segment sharing increases the efficiency of memory management since a process needing to use a shareable segment already in main memory need only attach the segment to its virtual address space, thus reducing segment swapping.

A segment can be defined as executable, readable, writable, or some combination thereof. Two different processes sharing a segment can have different access rights to it. So, for example, an "owner" of the segment could have read and write access, while another "user" of the segment would only be allowed to read it.

A segment also can be designated as swappable or nonswappable. If nonswappable, the segment will not be swapped from the main memory to disk memory. If it is imperative that the segment not be moved around in main memory as, for example, when I/O is being done in the segment, it can be locked in main memory. In addition, a segment can be blocked (that is, made unavailable) to other processes while it is being initialized.

Finally, segments can be either active or inactive. If inactive, the segment is not presently a part of the address space for each process that owns it.

3.2 Process images

The executable image of a process consists of a set of segments. Originally, a kernel process had its segments completely specified at link time. Later enhancements allow kernel processes to dynamically add or drop segments using messages to the memory manager. A supervisor process can use OST calls to create or destroy segments, make copies of segments, request named segments, attach or detach segments to its address space, grow or shrink a segment, lock a segment in main memory for I/O, or make a segment swappable or nonswappable.

A kernel process starts with at least four segments: the Kernel Process Control Block (KPCB) segment, the stack segment, the system message buffer segment, and the process's text segment. The KPCB segment contains the process-dependent information. The stack and system message buffer segments are shared with all other kernel processes and the DMERT kernel while the process's text segment is shared only with other invocations of the same kernel process. (Most kernel processes have only one invocation active at a time.)

A supervisor process starts with at least three segments: the Process Control Block (PCB) segment, which contains the system's tables for the process; the stack segment; and the text segment. The process's data may be combined with its stack into a single segment or it may be placed in a separate segment. The access modes for the PCB and text segment are read-only and read-execute, respectively. The process may never write its PCB. The stack segment of a supervisor process is never shared, while a text segment is usually shareable between multiple invocations of the same process.

IV. SCHEDULING

DMERT simultaneously supports both a real-time and a time-sharing environment. The kernel and kernel processes operate in a real-time environment and have first call on the available time of the 3B20D processor. The remaining time is shared among special, supervisor, and user processes.

4.1 Real time

Any process that must satisfy a real-time requirement should be a kernel process. DMERT maintains a process hierarchy based on 16 execution levels. A kernel process can belong to levels 3 through 15. (Levels 0 through 2 are reserved for time-sharing environment.)

DMERT bases its real-time allocation strategy on three concepts: execution levels, round robin scheduling, and preemption. DMERT dispatches processes at the highest execution level first. For example, a process belonging to execution level 15 is dispatched before a process belonging to level 14, which is dispatched before a process at level 13, and so on. For each execution level, DMERT maintains a list of waiting processes. When a process requests servicing, it is added to the appropriate list in a round robin fashion. As the operating system descends the hierarchy, it dispatches all the waiting processes at each level.

Generally, a kernel process executes until it exits. However, if another kernel process at a higher execution level is awakened, DMERT preempts the executing process. Upon completion of the preempting process, if no other higher level processes are awakened, the operating system resumes the suspended process.

DMERT's management of real time is straightforward and adds only a minimal amount of overhead. In fact, preempting one kernel process and dispatching another takes only about 320 microseconds. At the same time, applications are allowed to assign their own process's execution levels, which customizes their control and distribution of real time. This approach has proved to be quite flexible and permits a variety of applications.

4.2 Time sharing

As stated previously, the portion of real time not utilized by the kernel or kernel processes is time shared among supervisor and user processes. Processes supporting the time-sharing environment, such as the scheduler and the memory manager, are special kernel processes that reside at execution level 2. These processes are at the bottom of the real-time hierarchy and gain control of the processor only after all other real-time work is completed.

Supervisor and user processes normally execute at level 0. Level 1 is reserved for when they need to lock each other out. Within execution level 0, the scheduling hierarchy of supervisor and user processes is based on priority. The major difference between priority in the time-sharing environment and execution levels in the real-time environment is that DMERT adjusts priorities dynamically, whereas execution levels are fixed.

Priority adjustment is based on two factors: state and age. A time-sharing process can be in one of three states: sleeping, waiting, or executing. As a process enters a new state, it begins to age (from zero) until it changes state. Processes in different states age at different rates.

DMERT uses age and state to adjust the process's scheduling

priority. In particular, the priority of a sleeping or executing process is reduced as it ages and the priority of a waiting process is increased as it ages. The result is that a low-priority process waiting to run will eventually move above higher priority processes that have been executing.

DMERT selects the time-sharing process to execute by searching down a priority list until it finds a process waiting to run. The selected process continues to run until it changes state or is preempted by a kernel process. It cannot be preempted by a higher priority time-sharing process. A time-sharing process can leave the executing state for one of the following reasons: (*i*) it roadblocks (enters the sleeping state); or (*ii*) it uses up its allotted time slice and is returned to the waiting state; or (*iii*) it terminates. When a process times out, it is chained to the end of the list of waiting processes at the appropriate priority. Then, DMERT searches the priority list to find the next time-sharing process that is waiting to execute.

V. PROCESS CREATION AND TERMINATION

The DMERT operating system supports the dynamic creation and termination of kernel, supervisor, and user processes. A process can be created on demand from a process load file maintained in secondary memory and can be terminated and recreated at will. Process creating and termination can be requested via messages, OSTs, or commands executed at a terminal.

5.1 Process creation

The load file of a process is generated by means of the process loader, which is part of the software development system.[4] A process load file contains the text and data segments that comprise the initial image of the process. It also specifies a variety of basic process attributes, such as the execution level, stack size, instruction privileges, entry points, and, for time-shared scheduling, priority and time slice.

The creation of a process involves the identification of each of its segments in the kernel memory management tables, the formation of its initial image in main memory, and the identification of the process itself in the operating system scheduling tables. Start-up of a created process is generally accomplished by sending it a special initialization event, in which case control is passed to the process's event entry. In the case of supervisor and user processes, control is passed to the process's start entry.

5.1.1 Process creation mechanisms

There are three distinct mechanisms for creating a process. The system bootstrap mechanism creates a set of processes that constitute

the nucleus of the operating system and are essential to its operation. These include a process, called the process manager, whose principal function is the creation of other processes from process load files. The process manager can be instructed to create processes via the mechanism, and of sending it a message containing the name of the process load file and other relevant information. The third mechanism is the execution of a "fork" system call by a user process that results in the creation of a copy of the process executing the fork. The copy, called a child process, can subsequently execute another file via the "exec" system call, i.e., exchange its text and data for that of another process load file.

The creation of processes during system bootstrap is carried out by the kernel in a very efficient manner utilizing full access to the memory management tables and functions. The process manager is implemented as a swappable supervisor process that calls upon the services of the memory manager through OSTs and the file manager through messages. The fork mechanism is carried out within the process executing the fork and avoids the overhead associated with loading in the process manager.

5.1.2 Process creation features

Certain features are provided by the process creation mechanism. These include a distinction between single-copy and multiple-copy processes, the sharing of segments, and the use of shared libraries.

Single-copy processes are not replicated in main memory by successive creations. Instead, the operating system increments a usage count, which is decremented whenever the process is terminated. The process remains in the system so long as its usage count is at least 1. Device handlers that are repeatedly opened and closed are typically single-copy processes.

Successive creation requests for a multiple-copy process results in processes with distinct PIDs that have independent existences. As mentioned earlier, the text segment is usually shared among the invocations of a multiple-copy process. Data segments are typically private.

Processes that have a parent-child relationship may conveniently share segments as part of the creation process. The process load files of the parent and child processes define the segments that are shareable. When the parent process requests the creation of the child via a message to the process manager, it identifies the segments to be shared and specifies the access permissions for the child process. A process can share segments with any number of different child processes.

Shared libraries consist of segments of text and data that can be incorporated into any number of processes. A shared library is formed

by the process loader, and its process load file is maintained in secondary memory. If a process wants to use a shared library, the load file of the process need only contain the name of the library. The segments of the library are loaded into the process when the process is created unless they have already been loaded to satisfy the creation of another process. Symbolic references are resolved when the process load file is formed. The use of shared libraries decreases the size of the text segments in the process load file.

5.2 Process termination

Any process running under DMERT may elect to terminate itself or may be terminated by request of another process or the kernel. Exceptional cases are allowed; that is, the operating system has the capability of declaring processes to be essential or nonterminable. In particular, the processes that provide the basic operating system services are essential and cannot be terminated short of another system initialization. A kernel or supervisor process may be declared to be nonterminable as an option of the process loader. For such processes termination requests are always denied.

The operating system supports a variety of mechanisms for requesting process termination, as described below. However, regardless of the specific request mechanism used, the operating system takes the same basic actions.

5.2.1 Termination actions

All requests to terminate a process eventually result in a message being sent to the system scheduler specifying the PID of the process to be terminated. The termination request will result in the removal of the process from the system, unless it is a single-copy process with a usage count greater than 1, in which case the usage count is decremented. Removal of the process entails deleting its entry in the scheduling and segment tables. Shared segments that are needed by other processes for execution are retained in the system. The other segments are removed from the memory management tables and any associated secondary memory uses for swapping are freed up.

The operating system also takes some actions on behalf of the terminating process to free up other system resources that may have been allocated to it. Messages still queued up for the process are deleted or returned to the sender with a special acknowledgment. The process is detached from system ports and interrupts. For a supervisor process, the operating system issues close requests for any files it may have in the open state.

The message sent to the scheduler to terminate a process may optionally request a memory dump of the process. In this case, the

operating system produces a file in secondary memory similar to a process load file, which includes a copy of each of the process's segments at the time of termination. Such dumps are often useful for debugging purposes. Another option frequently exercised to coordinate the interactions of processes is to have the parent process notified when one of its child processes is terminated.

5.2.2. Process termination request mechanisms

A process may send a message to the system scheduler to terminate itself or another process. In certain situations, the operating system will terminate a process automatically. For example, a process that is faulted but that does not have a fault entry is terminated by the kernel.

A user process typically executes an exit system call to terminate itself. The exit system call is converted into a terminate message to the scheduler.

The operating system provides various OSTs to terminate a group of processes. One OST can be used by supervisor or kernel processes to terminate all processes having the same Utility Identifier (UID). (The UID is an identifier of a process that is selected at process load time and is administered to be unique to each process load file). This OST terminates all invocations of a multiple-copy process, since multiple invocations have the same UID. Another OST can be used by a kernel process to terminate all processes belonging to a specified process class. DMERT supports the optional grouping of processes into one of several classes, with each class chosen at load time. The capability of terminating all processes in a given class is useful in overload control schemes. Another termination OST available to kernel processes is specifically designed to handle system message buffer overload. This OST terminates processes that are using more than a specified percentage of the system message buffer resources. The actions of this OST also can be restricted to processes with a specified execution level or process class.

Several commands are provided to permit process termination from a terminal. One command will force the termination of any process with a specified PID and also will cause a memory dump of the process to be created. Another command is provided to terminate all processes with a specified UID. It also has the special feature of clearing the names of all shared segments used by the processes. This feature is used in conjunction with the updating of a process.[5]

VI. SIMULATED *UNIX* OPERATING SYSTEM

The time-sharing environment supported by DMERT simulates a *UNIX* operating system[2] implemented via a supervisor process called the *UNIX* Supervisor Process (USP). The USP supports standard

UNIX software including system calls from C programs, file operations, process communication through pipes, and interpretation of terminal commands through the "shell" process.

Processes controlled by the USP are called user processes. The USP partitions its address space into a user area and a supervisor area and appears as a single process to the DMERT kernel. Thus, the user and supervisor are physically combined into the same process, having the same PID, scheduling priority, PCB, etc. Each time a user process forks, another USP is formed.

The role of the USP is to supply services to its user portion. It accomplishes this through supervisor OST calls and through communication with other DMERT processes. For example, file system capabilities are provided by the USP sending the appropriate messages to the DMERT file manager process.

The availability of a simulated *UNIX* operating system in DMERT allows *UNIX* programs from other processors to execute on the 3B20D Processor. DMERT provides some capabilities to user processes not currently supported by the standard *UNIX* operating system. These include asynchronous I/O directly to or from the user's address space and memory management of user process segments. In addition, there are a number of file system capabilities, such as contiguous files, that are provided through the DMERT file management facilities discussed in Section VIII. User processes also have access to the DMERT IPCs such as messages and events.

DMERT's memory management capabilities allow a user process to manipulate and share portions of its address space on a segment basis. In particular, a user process can create a new segment in its address space and can specify the virtual address of the segment. It can acquire an existing and named segment into its address space and also remove segments from its address space. Segments listed in a user process's PCB can be activated or deactivated through OSTs to the USP. OSTs permit it to share up to three segments with a process it creates via the DMERT process creation functions described earlier.

VII. I/O FACILITIES

The operating system supports communication with peripheral devices through a set of drivers and device handlers. These drivers isolate most processes from the details of the peripheral system, and they ensure efficient use of the peripheral devices by scheduling access to them on an equitable basis.

The architecture of the I/O software closely resembles the I/O hardware architecture.[5] The I/O Processor (IOP) driver and the device handlers manage the IOPs, the Peripheral Controllers (PC), and the

Peripheral Controller Subdevices (PCSDs). The disk driver manages and controls the disk file controllers and the disks.

7.1 Input/output processor driver

The IOP driver is a kernel process that administers all IOP transactions in the 3B20D DMERT system. The driver is responsible for normal I/O activities fault recognition and recovery, configuration management, and diagnostic access.

The IOP, from a software standpoint, can be visualized as a three-level structure (see Fig. 1). The "front-end" Peripheral Interface Controller (PIC) controls up to sixteen peripheral controllers (PCs), and

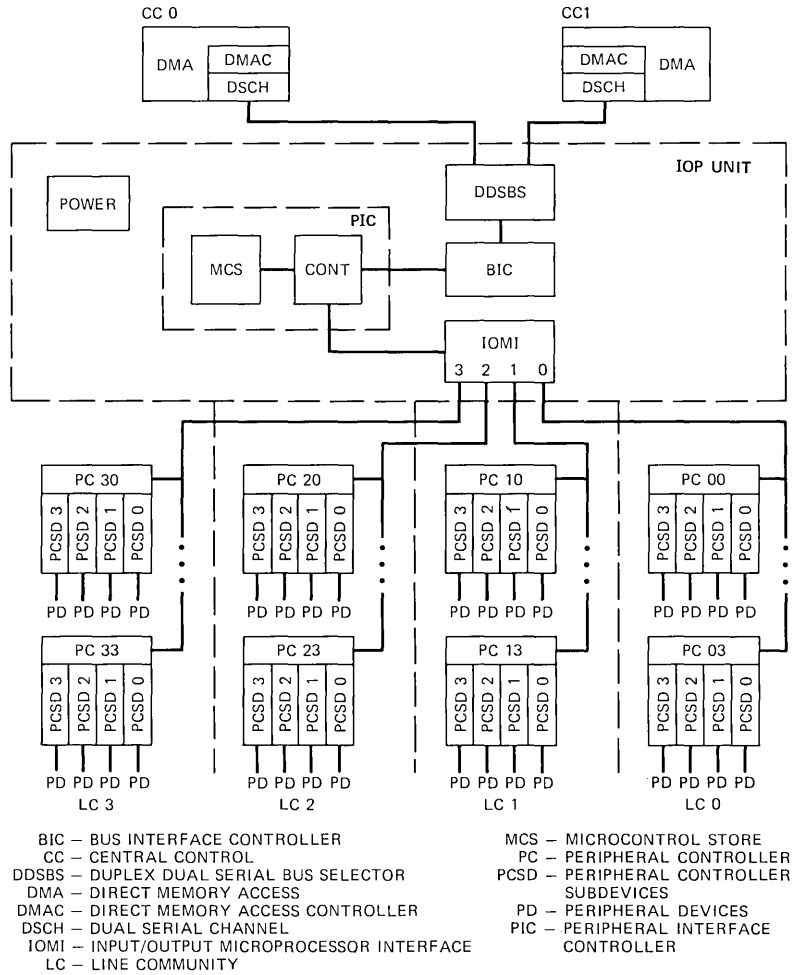


Fig. 1—Input/output processor.

each PC controls up to four subdevices (PCSDs). The subdevice provides the interface to the end device, such as magnetic tape unit, teletypewriter (TTY), data link, etc. Four PCs are combined to form a PC "community" with each PC community having a separate power supply.

Each element in the IOP (PIC, PC, and PCSD) has a corresponding element in the driver called a handler and corresponding unit control and option blocks in the Equipment Configuration Data (ECD) data base.[6] The handler for the PCSD is referred to as the device handler. A handler for a PC is called the generic PC handler or application PC handler. The handler for the PIC is called the generic PIC handler. The term generic implies that the handler is capable of performing all required handler functions for more than one PC type (that is, TTY, magnetic tape, data link, etc.).

Handlers are collections of C-language functions that have well-defined interfaces with the driver and are responsible for carrying out all maintenance (excluding diagnostics), recovery, and normal mode operations for their respective elements in the IOP. Handlers contain the necessary specialized logic to deal with a given unit type.

The handlers' service routines, input routines, control routines, and associated libraries form a single IOP driver process (IODRV).

IODRV can be subdivided into the following functional areas:

(*i*) Common service routines: routines that are frequently called from numerous points within IODRV and the handlers.

(*ii*) Configuration control: routines that maintain proper configuration of IOP units (inverted tree structure).

(*iii*) Input routines: routines that process primary inputs from the DMERT operating system and pass them off to IODRV configuration control or handlers.

(*iv*) Application and generic handlers: the operational interface between the user and physical device (magnetic tape, terminals, etc.).

(*v*) Maintenance handler: the diagnostic interface to the IOP units.

(*vi*) Archive libraries: system routines used by IODRV and other kernel-level processes.

Normal mode activities are carried out through the input routines, common service routines, and the operational handlers. All communications within the driver are through function calls.

I/O messages enter IODRV as message events and pass through the message input routines. Similarly, operating system traps enter IODRV at the OST entry point and pass through the OST input routines. Typically, I/O messages and OSTs contain a Logical Device Identification Number (LDIN) that identifies a logical (or virtual) device with which a user wishes to do I/O. IODRV maps the LDIN to

one or more physical devices. Once a physical device is identified, the IODRV can identify the corresponding handler via the ECD and pass control to it.

Completion reports, or responses, are deposited in the IODRV response queue by the IOP. If responses have been added to the response queue within a certain batching interval, the IOP will interrupt the 3B20D, causing IODRV to be entered. IODRV pops each queued response and, based on the PC and PCSD identifier in the response, maps to a physical unit, and passes control to the handler.

Maintenance and recovery activities are coordinated through IODRV configuration control routines, which, in turn, call on the handlers at appropriate points in time to carry out specialized maintenance operations at the subdevice level. Diagnostics for the PIC and PC are handled exclusively by the maintenance handler.

7.1.1 Handler applications

Peripheral devices supported by the IOP include TTY terminals, Maintenance TTY (MTTY) terminals, magnetic tape drives, data links, and the Scanner and Signal Distributor (SCSD). Interfaces to these devices are provided by IODRV and device handlers. IODRV is responsible for initializing units upon bootstrap and removing and restoring units upon manual requests or faults. Each handler-peripheral device combination determines the interface mechanism and the set of features to be supported. The following sections give a brief description of the facilities supported by each peripheral device type.

7.1.2 Terminal devices

The Craft Interface Handler (CIH) provides access to terminal devices. The CIH communicates with two types of controllers: Maintenance Terminal Controllers (MTTYCs) and Terminal Controllers (TTYCs). MTTYCs support four subdevices: a Maintenance Terminal (MTTY); a Receive-Only Printer (ROP); a Switching Control Center (SCC) interface; and an Emergency Action Interface (EAI).[7] TTYCs support terminals (TTY).

The MTTY, TTY, and ROP are known as terminal devices. The SCC and EAI devices are not terminal devices and are accessed via other handlers (see below). All standard terminal operations supported by the *UNIX* operating system are available to TTY devices, including read, write, open, and close requests, which are supported through a message interface. The ROP does not support reads.

7.1.3 Data links

The Communication Protocol Handler (CPH) provides access to synchronous data links. The CPH is designed to communicate with

two types of peripheral controllers: (*i*) the MTTY controller, and (*ii*) the synchronous data-link controller. In the case of the MTTY, access is available only to the SCC peripheral controller subdevice, which supports synchronous data link communication between the 3B20D and an SCC office using the BX.25 protocol.

The synchronous data link controller supports the BX.25 link layer (level 2) communication protocol and the Digital Data Communication Message Protocol (DDCMP) through the use of different versions of peripheral controller software. The CPH software supports two access methods: link-layer protocol access (level-two-only access) and BX.25 packet-level (level 3) protocol access. The use of a link-layer protocol (BX.25 and DDCMP) assures the integrity of data transmissions on a physical link. The use of a packet-layer protocol (BX.25) allows the added capability of multiplexing multiple users on a physical link. Flow control procedures also are used on both protocol layers.

In addition to the the two different access methods, the handler supports both a simplex and a duplex link configuration. In the duplex configuration, two physical links make up a logical communication path between the 3B20D and another system. The CPH automatically routes data through the currently active physical link. Link switching is done automatically when the active link fails.

7.1.4 Magnetic tape drives

The magnetic tape peripheral controller handles up to four 9-track 800 or 1600 bits per inch (bpi) tape drives. The magnetic tape handler provides the interface to this controller and supports open, read, write, seek, and close requests through a message interface. Seeks are not supported for write operations.

7.1.5 Scanner and signal distributor

Administration and control of the Scan and Signal Distributor (SCSD) points currently involves two DMERT kernel processes: the SCSD administrator and the SCSD handler. The latter is an integral part of the I/O driver process. The primary function of the SCSD software is to provide an interface enabling client processes to manipulate distribution points and receive information about the state of the scan points (i.e., autonomous scan state transition and directed scan reports). The SCSD administrator allows a client process to identify SCSD points by logical or physical addresses; logical addressing allows applications to code software independently of physical cabling.

The SCSD handler translates messages from the administrator into SCSD controller commands and receives responses from the controller

and forwards these responses to the administrator through a message interface.

7.1.6 Direct user interface

For some applications the current method of communication with the peripheral controller subdevices through IODRV is not efficient enough to meet their needs. Therefore, the Direct User Interface (DUI) exists to expedite data transfers between an application process and a specialized 56-KB BX.25 data-link controller.

The DUI handler is an integral part of IODRV. In the normal mode of operation, the only functions of the DUI handler are to set up and clean up the DUI table, which is in a common area of memory and is used for passing commands and status information between the application process and the peripheral controller. Using the DUI table, jobs are passed directly to the peripheral controller by the application process without any intervention from IODRV.

A secondary function of the handler is to administer the fault recovery strategy for the peripheral controller subdevice. If the subdevice has to be removed or restarted, the handler will tear down the DUI table and send a message to the application process.

7.2 Disk driver

The disk driver is a kernel process that handles all normal disk I/O and all maintenance disk I/O. Only system initialization I/O bypasses the disk driver and transfers information directly from the system boot device to main memory. The disk subsystem consists of the disk driver, the Disk File Controller (DFC), and the Moving Head Disk (MHD) drives. The 3B20D supports a maximum of eight DFCs, each having up to eight MHDs. The DFC and MHDs are described in Ref. 8.

MHDs may be used in a simplexed or duplexed configuration. In simplex mode the MHD stands alone. Should a file become damaged it will be irretrievably lost. In duplex mode two MHDs are maintained such that each is an exact copy of the other. Should one disk fail the other can be used in simplex mode.

7.2.1 Operational characteristics

The disk driver handles open, close, read, and write message requests. Open and close messages are passed from the file manager (see Section VII) to the disk driver, while read and write requests may be sent directly from any process or routed through the file manager. Before the kernel attaches the read or write message to the disk driver's message queue, it verifies that the segment is locked in main

memory (see Section 3.1). It also verifies that the I/O transfer is within the bounds of the segment.

When the disk driver processes the I/O message, it translates the LDIN contained in the message to one or more physical devices. The request is then placed in one of three circular job submit queues in main store associated with the DFC for each specified physical device.

The three types of disk job queues are high-priority, base-priority, and special. Special commands sent by maintenance processes or originated in the disk driver are immediately executed by the DFC from the special job queue. High-priority jobs can be sent by any client process and will be processed by the DFC before base-priority jobs. All other jobs are placed in the base-priority queue.

Whenever the DFC completes a job requested by the driver, it returns a response indicating the outcome of the job. All job responses are placed in a single main store response queue, regardless of the priority of the original job. The DFC generates an interrupt to the driver after each response is added to the queue. The driver only clears the interrupt after processing the last entry in the response queue.

The disk driver handles job responses each time it is entered at its interrupt entry. The job response indicates the status and identity of the job being reported. If the job was successful, the driver sends a successful job completion acknowledgment to the client using the same message buffer that requested the I/O. In writing to duplexed disks, the driver guarantees that both disks were written successfully before acknowledging the job. In reading from duplexed disks, the driver reads from a single disk, alternating disks between requests.

If a job failed, the driver determines whether the device should be removed from service or if it should retry the job.

When the driver wishes to retry a failed job, it sets up a retry request in the main store retry queue. The format of an entry in the retry queue is the same as that of a job in any of the other queues. After writing the entry in the queue the driver wakes up the DFC with a programmed I/O command. The DFC then takes this job, even if the other submit queues contain work. When the driver handles the response from the retry request, it knows the queue is available for reuse.

7.2.2 Reliability characteristics

The disk driver also has a message interface for maintenance commands. Once the device (MHD or DFC) is taken out of service by the disk driver, the device can be reserved for maintenance access and the disk driver provides the maintenance client processes unlimited access to the device. During maintenance, specific areas of a MHD can be read or written by bypassing many of the operational checks performed

on normal I/O requests. This allows the creation of a disk and the system update of a disk with a new software generic.[6]

VIII. FILE SYSTEM

All accesses to the file system are done through the file manager, a DMERT kernel process. In addition to maintaining file system security and integrity, the file manager translates read and write requests within the file system to physical I/O requests on the disk.

The DMERT file system is similar to the file system provided by the *UNIX* operating system and features a hierarchical structure, byte-oriented files, and uniform access to files, directories, and periphery. In addition to regular files, which are scattered throughout the disk and can grow dynamically, DMERT also provides contiguous and extent files, which are contiguous on disk but have limits on their growth. Contiguous and extent files are optimum for data base and object files, where large, fast I/O transfers are needed. For field update, DMERT provides a "windowless move" facility, which automatically moves an updated object file over the old one, thus eliminating any possibility that the file be used or the system initialized while the file is in an inconsistent state.

To meet DMERT's reliability requirements, DMERT file systems are crash resistant. In particular, a crash does not jeopardize file system integrity, the file systems do not need manual repair, and they are available within seconds after a crash.

The file manager uses two techniques to ensure crash resistance. First, it orders all writes to disk to maintain a consistent file system state. To create, link, or write a file, the ordering is:

(*i*) Write the data blocks
(*ii*) Write the indirect i-node blocks
(*iii*) Write the i-node*
(*iv*) Write the directory entry, if necessary.

To unlink or truncate a file, the ordering is:

(*i*) Write the cleared directory entry, if necessary.
(*ii*) Write the cleared i-node.
(*iii*) Free the blocks.

Second, to ensure that no block is allocated to more than one file, the file manager rebuilds a file system's free-block list before it is used following a crash. Doing this for the 50-000 block, 2048-i-node *root* file system adds about 10 seconds to DMERT's boot procedure.

These two techniques are sufficient to ensure crash resistance, and we have found no problems with these in the field.

---

* An i-node describes a file and contains its block addresses. An indirect i-node block extends the i-node and contains more block addresses.

IX. SUMMARY

This article has described the DMERT nucleus, which consists of the kernel, the special processes, the I/O drivers and file manager, the process manager, and the *UNIX* supervisor. The major services provided by this nucleus include a multitude of interprocess communication mechanisms, a sophisticated set of memory allocation features, both real-time and time-shared scheduling, dynamic process creation and termination, a simulated *UNIX* environment's communication with terminals, magnetic tape drives, data links and disks, and powerful real-time and time-shared file system capabilities. The operating system has been continually evolving since DMERT was conceived, and is expected to continue to evolve over the next few years. This article has described the first official version of DMERT, which entered service in the Bell System during September 1981.

X. ACKNOWLEDGMENTS

We thank C. J. Antonelli, J. Q. Arnold, T. P. Bishop, R. W. Fish, N. A. Martellotto, R. R. Snead, P. J. Stankus, R. M. Venzon, and R. E. Yuknavech for their contributions to this document. Special thanks go to J. J. Wallace for his contributions to the file manager section.

REFERENCES

1. J. R. Kane, R. E. Anderson, and P. S. McCabe, "The 3B20D Processor & DMERT Operating System: Overview, Architecture, and Performance of DMERT," B.S.T.J., this issue.
2. D. Ritchie and K. Thompson, "The *UNIX* Time-Sharing System," B.S.T.J., *57*, No. 6, Part 2 (July–August 1978), pp. 1905–29.
3. I. K. Hetherington and P. Kusulas, "The 3B20D Processor & DMERT Operating System: 3B20D Memory Systems," B.S.T.J., this issue.
4. B. R. Rowland and R. J. Welsch, "The 3B20D Processor & DMERT Operating System: Software Development System," B.S.T.J., this issue.
5. A. H. Budlong and F. W. Wendland, "The 3B20D Processor & DMERT Operating System: 3B20D Input/Output System," B.S.T.J., this issue.
6. R. H. Yacobellis, J. H. Miller, B. G. Niedfeldt, and S. S. Weber, "The 3B20D Processor & DMERT Operating System: Field Administration Subsystems," B.S.T.J., this issue.
7. M. E. Barton and D. A. Schmitt, "The 3B20D Processor & DMERT Operating System: Craft Interface," B.S.T.J., this issue.
8. R. E. Haglund and L. D. Peterson, "The 3B20D Processor & DMERT Operating System: 3B20D File Memory Systems," B.S.T.J., this issue.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

Field Administration Subsystems

By R. H. YACOBELLIS, J. H. MILLER, B. G. NIEDFELDT, and S. S. WEBER

(Manuscript received March 10, 1982)

This article describes the field administration facilities of the Duplex Multiple Environment Real Time (DMERT) operating system, as provided on the 3B20D Processor. These facilities are: Recent Change/Verify, the subsystem that allows manipulation of office-dependent configuration information; Field Update, the software change mechanism; and System Update, the component used to install a new generic program in an office. The article also includes information on how these capabilities fit into the overall scheme of field support in an in-service office environment.

I. INTRODUCTION

An integral part of high-reliability applications of the Duplex Multiple Environment Real Time (DMERT) operating system is the administration of system hardware information and of software. This includes both the initial delivery of the system as well as subsequent upgrades. In DMERT[1] there are three commonly used capabilities to apply, track, and administer such changes. These are Recent Change/Verify, Field Update, and System Update. They are listed in this order according to decreasing frequency of field use and increasing impact (typically) on the overall system. Each of these capabilities is designed to permit display of some aspect of the current status of the system, to change that status in a simplified and highly reliable way, and to either reverse such changes or make them permanently a part of the system. This article discusses each in turn, and provides examples of their use. Each capability may form the base for an application-dependent version of its function. These functions are discussed briefly in the rest of this introduction.

The 3B20D Recent Change/Verify (RC/V) system provides the ability to change and manipulate various aspects of office-dependent information. This capability is focused on the system hardware and software configuration and is based on the Low-Level Access (LLA) Data Base System, whose operation is normally hidden from field-site administrators. RC/V is used manually or automatically to verify and change the hardware and software components known to the system, and the ways in which they are interconnected.

Field Update is used to correct problems in the operation or functionality of the system. Field Update is the official fix mechanism for DMERT. Rapidly installed emergency fixes, as well as more routine trouble corrections, may be installed into the software or other files in DMERT via Field Update.

Finally, System Update, also known as Generic Update, changes a major portion of the entire DMERT or application generic program. In doing so, System Update may write over old generic information or provide a completely restructured generic program image. Typically, a new generic release will involve a new structure for RC/V information as well, so RC/V may be involved with such an update. The following sections provide more details on these fundamental administrative capabilities of DMERT.

II. RECENT CHANGE/VERIFY—LOW-LEVEL ACCESS DATA BASE SYSTEMS

The 3B20D/DMERT System has provided a data base management capability as part of the DMERT operating system. Built upon a Low Level Access (LLA) data base system are the Equipment Configuration Data Base (ECD), System Generation Data Base (SG), and the 3B Recent Change/Verify (RC/V) and Data Base Evolution Systems. This section describes these systems and their relationship to the field administration environment.

2.1 Low-Level Access Data Base System

The Low-Level Access Data Base System organizes and manipulates data in a C-language environment. The name low level implies that the system places minimal restrictions on its users: decisions about data organization and retrieval are left to the application. LLA trades user convenience for greater flexibility in data base design and performance tuning.

LLA gives the user latitude in defining both data units and data models and provides a powerful set of primitives to access the data. System characteristics include:

(i) Data definition via a hierarchy of abstract types

(ii) Specification of data mapping from the data base to the user's buffers

(iii) Ability to select various access methods, i.e., logical organization of subsets of data

(iv) Data access through a library of functions

(v) Isolation of operating system dependencies in a small number of program modules.

Figure 1 gives a simplified schematic of the operation of an LLA application.

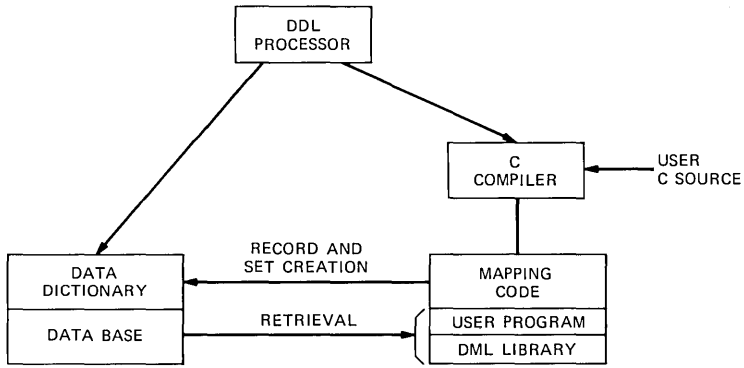
2.1.1 Data definition

The Data Definition Language (DDL) is used to define the "shapes" of records, the LLA data type for retrieval and storage. It also allows user-defined "views" of the data base via data mapping, and the specification of data models by associating records with access methods. The recognizer for the DDL, the Data Definition Language Processor (DDLP), has many C-compatible features, such as common syntax for preprocessor lines, comments, identifiers, constants, and type definitions. The DDLP generates C code to implement data mapping and C definitions, and a data dictionary to describe data types.

2.1.2 Data manipulation

The Data Manipulation Language (DML) is a library of functions that perform actions on instances of the data types defined by the DDL. The DML provides the following facilities:

(i) Creation and deletion of instances of data types



DDL – DATA DEFINITION LANGUAGE
DML – DATA MANIPULATION LANGUAGE

Fig. 1—Low-level access application.

(ii) Retrieval and update of existing instances of data types
(iii) Gathering of information about existing data instances.

These categories exist for instances of data bases, sets, and records. Generally, the lifetime of an instance of a data type starts with creation, proceeds through several retrievals and updates, and ends with deletion.

LLA is not used directly by a field administrator. Instead, the creators of various LLA data bases, be they 3B20D/DMERT system programmers or 3B20D application designers, provide appropriate higher-level access to their particular LLA data base application.

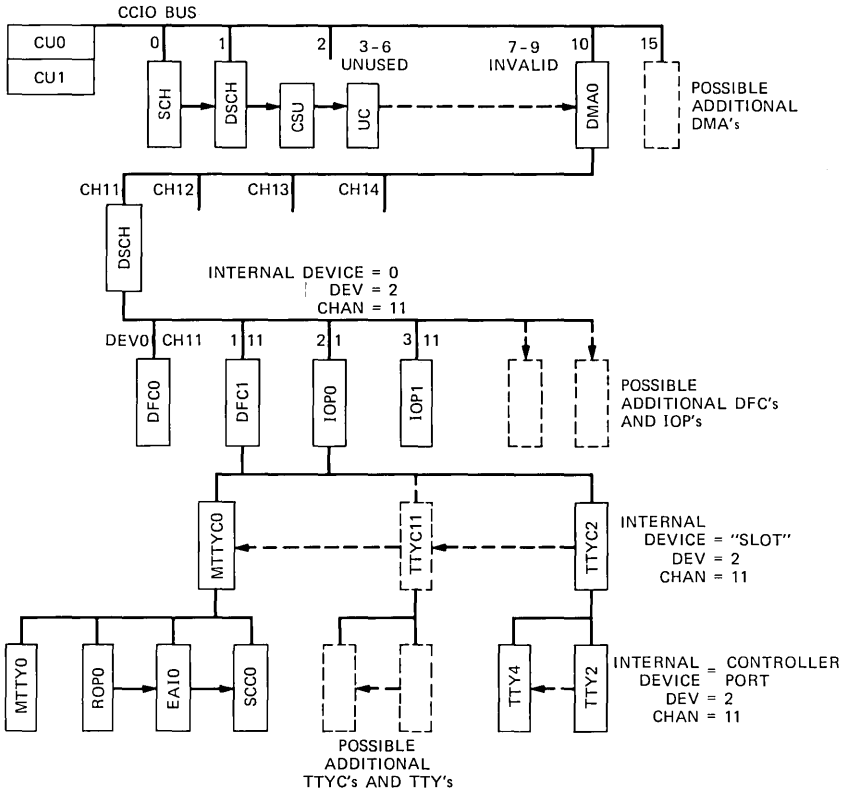
2.2 3B20D Data Base Recent Change and Data Base Evolution Systems

2.2.1 3B20D data bases

The 3B20D/DMERT operating system has two major LLA databases. The Equipment Configuration Data Base (ECD) describes the processor and peripheral hardware configuration, while the System Generation (SG) Data Base describes the system parameters, boot processes, and disk image and ECD administration information. The concept of a data base was adopted to eliminate redundant device information, provide a unified approach to handling and accessing that information, and provide easy methods for generating and changing it.

Records in the ECD data base represent the hardware devices in the 3B20D Processor system, such as the Control Unit (CU) and Input/Output Processor (IOP), and are logically linked in a manner analogous to the physical linkages (see Fig. 2). In addition, records are provided to organize physical devices as logical devices and to maintain error counts for each physical device. To provide rapid access, the ECD is always kept in main memory.

The information in the ECD and SG data bases is used by several classes of users. The DMERT operating system, itself, forms one set of using processes and includes the device drivers, processor and peripheral diagnostics, and processor and peripheral fault-recovery programs. The second class of users of these data bases is the human user, whether that person be a Bell Laboratories' application designer adding new peripherals to the ECD or an operating company craft preparing to add more memory to an on-line 3B20D in the field. Two types of access have been provided for these two classes of users: The DMERT operating system processes access the ECD through a collection of LLA primitives that provide rapid access to those specific items required, for example, by the device drivers. Human users utilize the Recent Change/Verify system, which provides a forms-oriented input, via a cathode ray tube (CRT) terminal. The user may create, change, delete, or merely review the forms. Error and consistency checking is provided at the time of initial entry and before storage into the data base.



CCIO – CENTRAL CONTROL I/O BUS
CH – CHANNEL
CSU – CACHE STORE UNIT
CU – CONTROL UNIT
DEV – DEVICE
DFC – DISK FILE CONTROLLER
DMA – DIRECT MEMORY ACCESS
DSCH – DUAL SERIAL CHANNEL
EAI – EMERGENCY ACTION INTERFACE
IOP – INPUT/OUTPUT PROCESSOR
MTTY – MAINTENANCE TERMINAL
MTTYC – MAINTENANCE TERMINAL CONTROLLER
ROP – RECEIVE ONLY PRINTER
SCC – SWITCHING CONTROL CENTER
SCH – SERIAL CHANNEL
TTY – TERMINAL
TTYC – TERMINAL CONTROLLER
UC – UTILITY CIRCUIT

Fig. 2—Prototype 3B20D configuration.

2.2.2 3B20D Recent Change/Verify

The Recent Change/Verify system is built upon the LLA data base management system and utilizes the LLA primitives for accessing and managing its two DMERT data bases. There are three basic components of 3B20D RC/V (see Fig. 3). The first is the front-end form processing system. This component is known as the On-line Data Integrity (ODIN*) subsystem. ODIN allows the various forms to be specified through a series of CRT screen mask definitions and for each

* ODIN is a product of Western Electric Company.

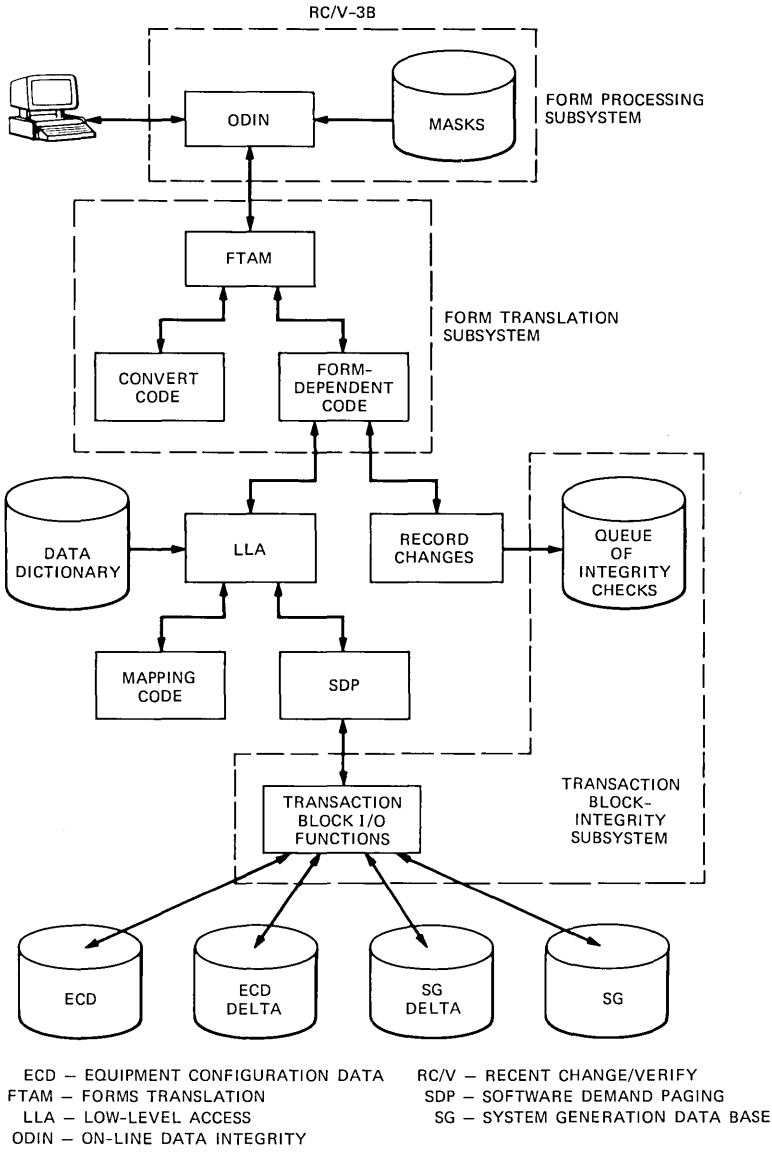


Fig. 3—Components of Recent Change/Verify.

of these definitions to contain certain syntactic information to be checked upon entry. For the ECD/SG data bases there are 36 different form types, each of which has an associated mask definition. Most forms are either ECD or SG forms, but there are a few that are directives for the RC/V or Evolution systems. For each form type some error checking is provided. The second fundamental component

of RC/V is the Form Translation and Mapping subsystem. This takes the output of ODIN and transforms it into LLA record definitions and access functions. Then the LLA functions are used to actually manipulate the data in the ECD and SG data bases. The third component is the transaction block-integrity check subsystem. This provides a mechanism for checking consistency between forms. RC/V has implemented the concept of a "transaction." Two special forms delimit a transaction. Upon processing a transaction-end form, RC/V invokes the integrity checks as well as linking the new information into the data base.

As we stated earlier, the ECD that describes the running 3B20D is always in main memory; however, there is also a copy on the disk. In order for a change to be made permanent it must be applied to the disk as well as the memory version. To maintain the integrity of the ECD, changes are soaked on the memory version (test state) before they are applied to the disk version (active). A special form has been provided to perform this final step of activating changes to the disk copy of the data base. Upon processing of this form, RC/V copies the main memory copy of the ECD to the disk. To facilitate error checking and correction, a journal file of all transactions is kept on-line and can be printed on the Receive-Only Printer (ROP) at the request of the office craft. Also, an error log file is maintained and a periodic audit of the ECD structures is performed.

2.2.3 Data Base Evolution System

Because the release of a new 3B20D/DMERT generic is anticipated to be associated with changes to the ECD or SG forms or the LLA primitives, a system for transforming these data bases has been provided. The Data Base Evolution system (DBEVOL) allows this transformation to occur in a regular and uniform manner without special programs needing to be written. DBEVOL allows old data to be restructured, new data fields to be added to existing forms, and old data to be deleted or changed. DBEVOL also provides semantic hook functions that allow applications to tailor some specific information before completing the data base evolution.

DBEVOL has two types of steps. The first set is characterized as pre-processing. Here a translation data base (also an LLA data base) is built on a host support processor. The inputs are the old and new form specifications (as used by RC/V) and a specification of the changes in Form Translation Language. These inputs are supplied with the new DMERT generic program. If semantic hook functions are required by the application they are also an input to the final translation data base. A translation data base matching the required changes in the standard DMERT ECD is also released with new DMERT generics.

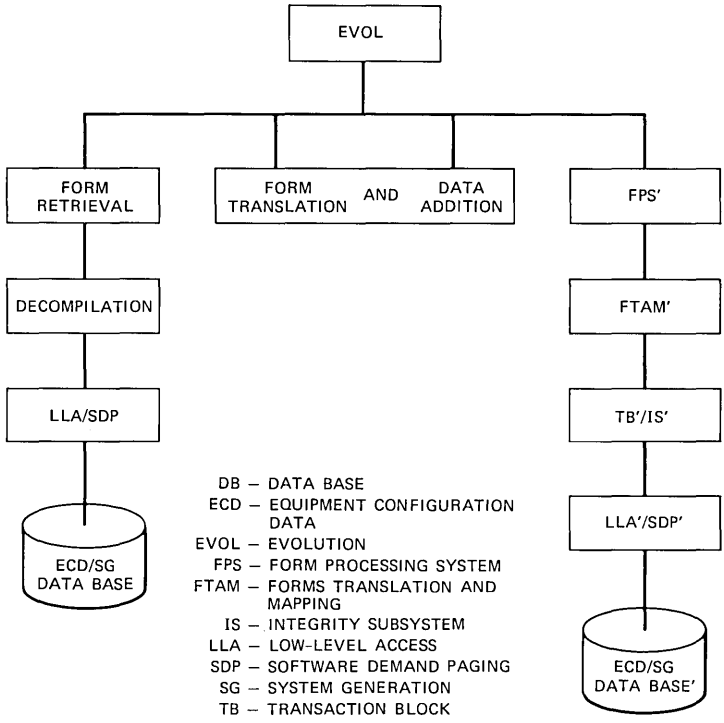


Fig. 4—Evolution of 3B20D/DMERT Data Base Management System.

The second set of actions are run-time steps that produce a new, evolved ECD/SG data base pair (see Fig. 4). The first step is a dump of the old ECD using the "old" existing generic RC/V. This is produced using one of the special forms provided by the RC/V system. Then this snapshot of the old data base is translated into a snapshot of the new data base. The "new" RC/V is then used to load the new data bases into the proper LLA format for the 3B20D.

DBEVOL runs on both the support processor and the 3B20D giving the using applications considerable flexibility in choosing a strategy for performing data base evolution. The evolved data base is actually put in place on the running 3B20D during the generic update scenario described below.

III. FIELD UPDATE

Field Update, which is typically called "overwriting" in traditional Electronic Switching Systems (ESSs), is the problem correction mechanism for DMERT. While overwriting usually applies specifically to program bugs, Field Update may be used to correct any file on the 3B20D disk. Such files may contain human-readable text or binary

tables, for example. (In DMERT, files are structured like a *UNIX** operating system file system.[2]) Field Update must perform this updating without disturbing call processing or other critical system functions. Since operating systems do not normally support this style of updating, some difficult technological problems had to be overcome in designing and implementing Field Update. Some of these problems and their solutions are described below, followed by a more general discussion of the overall structure and use of Field Update.

3.1 Problems and solutions

Like most modern operating systems, DMERT supports the concept of a process, which is a collection of tightly coupled executable programs. Programs are in turn broken down into units that perform specific activities, called functions. Processes can communicate with each other, generally at "arms-length," and are normally protected from each other by DMERT software and the 3B20D hardware and microcode. Since Field Update runs as a cooperating set of processes within DMERT, some highly specialized operating system interfaces were required to break through this protection. Furthermore, the real-time critical processes in DMERT or its applications must run continuously [they are termed "non-killable" (NK)], so that they are always available to process events quickly. The running process images of such processes must be accessible and changeable in main memory, again via special operating system functions.

Since a process is a collection of functions, the C-language[3] function was chosen as the unit of update. The implementation of field update specified that there be a single reference point for each changed function, so as not to require changes everywhere such a function was involved. To solve this, the concept of a Transfer Vector (TV) used in ESSs was implemented within a process image. Figure 5 is an example of a simplified process image showing this. In Fig. 5, the TV area contains a list of the addresses of the process's functions. When a change is made to function f, the new version f′ is written into a special "patch" area provided with the process, and the particular address in the TV area is switched to point to f′ (see Fig. 6). This solution also allows the fix to be backed out by changing the address in the TV back to its original value. When the fix has been tested and is ready to apply permanently, the space occupied by f can be made available for future fixes (Fig. 7). While this concept is simple, introducing TVs to DMERT had operating system implications down to the microcode level. With TVs, the impact of introducing a new or changed function

* Trademark of Bell Laboratories.

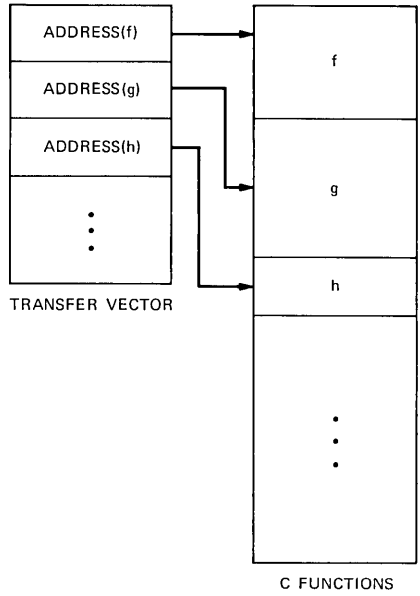


Fig. 5—Simplified DMERT process image.

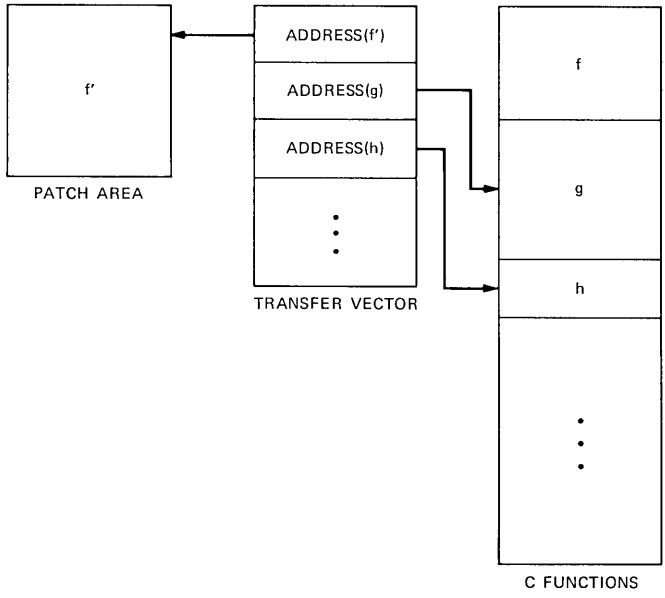


Fig. 6—Function f replaced by function f'.

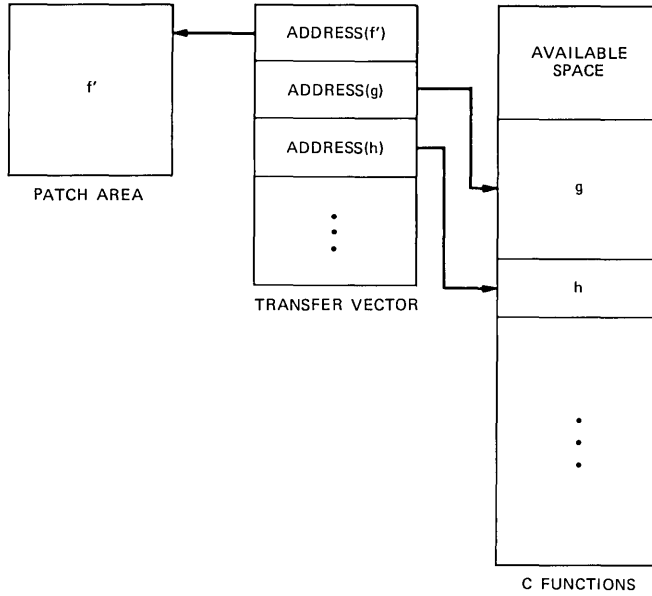


Fig. 7—Reclaiming the space occupied by function f.

has been restricted to a small, well-defined area of the process, making this activity inherently more reliable.

Traditional operating systems do not have the ability to change a critical function or process while the system is running. Since DMERT is derived from such an operating system, many challenges were encountered in providing the field update capability. Some specific areas included:

(i) The ability to change a file both instantaneously and in a temporary way. This is used in updating both non-killable processes and more routine processes that can be terminated and restarted;

(ii) Retention of sufficient symbolic information to properly update the 3B20D disk-resident versions of processes ("pfiles");

(iii) The ability to update C functions even though the old versions of the functions had been suspended while field update was running;

(iv) The ability to change data contents or the structure of data used by a continually running process;

(v) The ability to coordinate changes to functions within a process.

3.2 The use of field update

Field Update is an end-to-end concept within DMERT; that is, it is involved with the development, distribution, installation, and tracking of changes. When a process is first introduced into DMERT, or when its subsystem architecture changes, the process developer must com-

municate its characteristics to personnel who administer the DMERT source programs. The developer also must create a script of commands to be executed at a field site, which will be used to install, back out of, or make permanent a fix to the process. Generally, this will be simple to do because there are categories of existing process scripts, and new processes will fit into an existing category (or a simple modification to one will suffice). Once these steps are taken, the developer can depend upon the DMERT administrative system[4] and specific Field Update change development commands to remember these details. This approach standardizes the development of fixes so that each is handled the same way, as opposed to being a unique activity. The primary advantage comes when an emergency fix must be created quickly without the extra burden of collecting procedural information.

When a developer has created a fix and tested it, the standard change development mechanisms produce a package called a Broadcast Warning Message (BWM), which is used to transmit and install the fix (see Fig. 8). System Test personnel use this package to test the

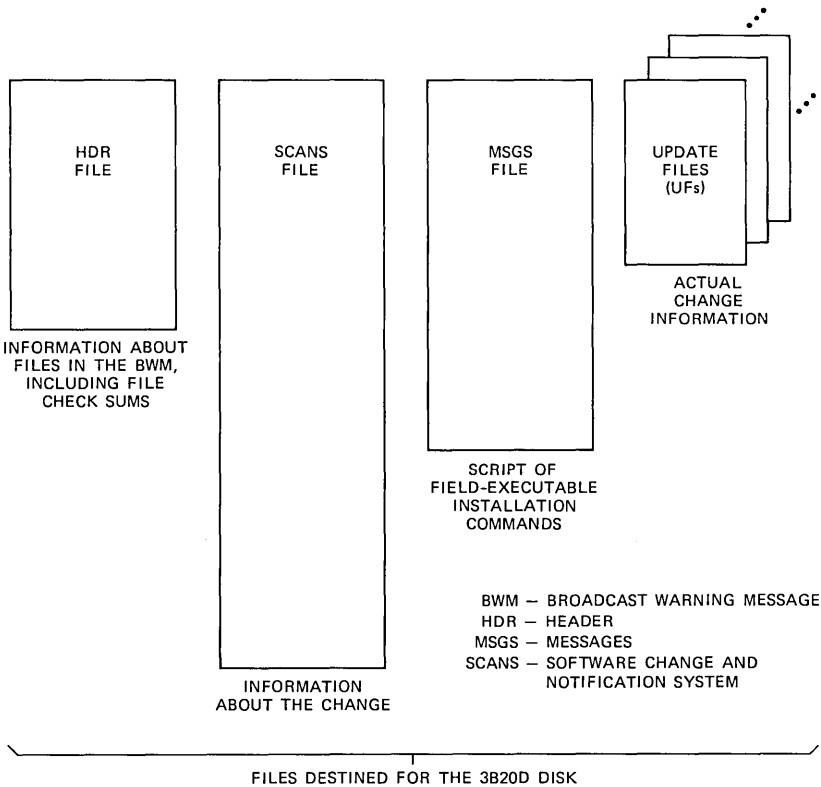


Fig. 8—Structure of a broadcast warning message.

field updatability of the fix as well as its impact on the system in the same way it will be installed at a field site (see the article on "System Integration and Test" in this issue of the Journal). When testing is completed, the fix can be packaged together with other fixes via automated tools into an official BWM for delivery to application project personnel, who will intermix it with application-specific BWMs and send it on. During this packaging, the particular order of installation of specific fixes is indicated both within and across BWMs.

A BWM consists of a set of files in a *UNIX* operating system directory, and can be transmitted via magnetic tape to a site. The Bell System is standardizing on the Software Change and Notification System (SCANS-II) as the official change distribution network, and the files in a DMERT BWM are also compatible with SCANS-II. DMERT also provides file reception software for use with SCANS-II. Typically, personnel at a Switching Control Center (SCC) will interrogate SCANS-II, recognize that a change is pending for one of their associated field sites, and initiate transmission of the change to the field site.

Once a change reaches a field site, it is stored in a staging area on disk until it is manually installed. The developer-produced script of commands is sent as part of the BWM (see Fig. 8), and is used by office personnel to install the change. With a short sequence of DMERT Field Update commands, the fixes can be:

(i) Installed
(ii) Tested
(iii) Backed out or made permanently a part of the system.

While a fix is being installed, an internal system error will result in automatically backing it out; once it is soaking in a temporary state, it may be backed out manually, or automatically if the system undergoes a major recovery action.

Each field site maintains an on-line log of all Field Update activity since the last System Update (see Section IV). This may be used to verify the current state of the office as far as installed BWMs are concerned, and is used each time a new change is installed to guarantee proper sequencing of changes. Other Field Update-related utility programs in DMERT can be used to print out a C function-to-process address map, and to verify that the main memory (executing) copy of a process matches its image on the 3B20D disk (see Section 3.1).

By the facilities mentioned above, Field Update allows fix creation in a style compatible with normal program development, prepackaging of developer-approved installation scripts, fix coordination both within and across BWMs, automated delivery and installation mechanisms, and detailed change tracking. These capabilities make Field Update a truly end-to-end DMERT change mechanism.

3.3 Field update example

Let us presume for this example that a problem has been found in the DMERT disk driver program, whose pfile is called dkdrv.o in directory /bootfiles. The developer has constructed a fix and tested it, and further system impact testing has verified it. The fix is given a DMERT official BWM name of BWM82-0028 (the first two digits are the year, and the last four a sequence number), and is passed to personnel in an application of DMERT, who approve it and send it out as application BWM, BWM82-0037. Once the fix has arrived at a field site, it is installed via the commands shown in Fig. 9. The descriptions below explain the commands:

(i) Request a printout of change information that field update has logged against process dkdrv.o.

(ii) Prepare the site to receive the BWM. After SCANS-II receives a command to send the BWM (not shown), it is transmitted automatically to the site with data error detection and positive reporting.

(iii) Install the fix into the system.

(iv) Test the fix (coupled, perhaps, with manual actions).

(v) Make the change permanent and remove the BWM files from the system. In this particular case the DMERT boot image is rebuilt as part of making the fix permanent, because the changed process is one of the system boot processes.

(vi) Once again display the change status of dkdrv.o.

(vii) Print a map of C functions and their addresses for drdrv.o.

(viii) Reclaim the space occupied by old versions of C functions in dkdrv.o.

The installation command mentioned above causes an entire set of commands to be executed, those in the "install" section of the script originally provided by the developer. An example of that script is shown in Fig. 10, which shows the Messages (MSGS) file for BWM 82-0037.

```
(i)     UPD:DISPLAY; FN "/bootfiles/dkdrv.o"!
(ii)    IN:REMOTE:START!
        VFY:BWM: 82-0037!
(iii)   UPD:BWMNO 82-0037!
        UPD:EXEC 82-0037: CMD APPLY!
(iv)    UPD:EXEC 82-0037; CMD SOAK!
(v)     UPD:EXEC 82-0037; CMD OFFICIAL!
        CLR:BWM:ALL!
(vi)    UPD:DISPLAY; FN "/bootfiles/dkdrv.o"!
(vii)   UPD:TRC; FN "/bootfiles/dkdrv.o" : ALL!
(viii)  UPD:AUD!
```

Fig. 9—Commands to Receive and Incorporate BWM 82-0037.

```
  APPLY.
                MRs: d8200002; DMERT BWM82-0028
UPD:UPNM BWM82-0037;FN''/bootfiles/dkdrv.o'':UF''/etc/bwm/82-0037/one.m''!
  SOAK.
                The fix(es) should soak for at least 1 days 00 hours 00 minutes.
                It will be apparent that the fix(es) have been applied:
                When no disk restore failures occur,
                commands to soak the fix appear here.
  BKOUT.
                If the fix results in the need to reboot the system, the fix will
                have been backed out automatically. If the fix does not result
                in a reboot but otherwise does not work correctly, it can be backed
                out by entering the command [s]:
UPD:BKOUT;UPNM BWM82-0037!
  OFFICIAL.
UPD:UPNM BWM82-0037;OFC!
                This will update the bootfile APPDMRT.
```

Fig. 10—MSGS file for BWM 82-0037.

IV. SYSTEM UPDATE

DMERT System Update provides a safe, reliable mechanism for field personnel to introduce new versions of DMERT and application software into 3B20D/DMERT systems, while minimizing service disruption. System Update differs from Field Update in the magnitude of the program and data changes being installed. Normally, a system update will replace all the software in the system with the release of a new generic program, which is a complete reissue of DMERT and/or application software and/or data. For this reason, system updates always include a memory reinitialization with a full bootstrap (reinitialization of all processes and data from disk). Only the contents of protected application segments, special memory areas where application systems may retain critical information, are retained across the boot. Since a system update includes a reinitialization, only the version of the software on the 3B20D disk is updated. The main memory images of system processes will then be re-read from the disk during the bootstrap. This section describes how this disk updating is done within DMERT, and gives an overview of the overall System Update process.

4.1 System Update concepts

The DMERT System Update Program (SUPR) provides a way to replace the entire contents of the 3B20D disk with a new version of those contents from a magnetic tape. SUPR deals with masses of data, and changes the disk contents section by section rather than file by file or logical data base updates. These sections are called *partitions.* To do this, SUPR takes advantage of the fact that the 3B20D disks are duplexed for reliability, writing the new system information onto

only one of a pair of disks. This is the *off-line disk method* of system updating. It derives its name from the fact that one of a pair of disks must first be removed from active service (taken off-line) before writing the new system onto it. With the off-line disk method the amount of redundant disk information is kept to a minimum during the update, and the disk structure may be completely changed. There is some increase in system vulnerability during the time that the disks are not running in duplex mode.

Certain aspects of the system update procedure have caused unique requirements and changes within DMERT. The key to the off-line disk method is protecting both generic programs from being overwritten during the update procedure. Since these generics reside on duplex disk mates, an off-line disk must never be restored to service. (The restore process includes a copy from the on-line to off-line disk.) The attributes of the "off-line" device state in the ECD were expanded to provide this capability. After a bootstrap on a new generic disk image, the disk copy of the old generic must similarly be marked off-line, and hence protected from restorals. This was accomplished by having each generic's ECD record the disks containing the *other* generic as off-line.

It was also necessary to be able to access partitions on an off-line disk, in order to read or write partitions on an off-line disk, to transfer files from the old generic to the new generic, and to perform recent changes on the new generic ECD (for example, in marking old generic disks as off-line). This was done by having the disk driver program access the Volume Table of Contents (VTOC)—the directory of the disk's contents—on the off-line disk during the update process. This is a special case, since the VTOC on an off-line disk may be different from that of its mate disk, or may not even be sane. When updating multiple disks, SUPR uses a special disk identifier added to the VTOC to ensure that the disk image being written corresponds to the information on that disk. As another safeguard, System Update uses checksums (special numbers computed from the data in a file) on the generic tape to check the new generic data for damage before writing it to the disk.

4.2 System update scenario

SUPR provides a complete update scenario, including a means to reverse the update and re-establish the original system. Because of the major impact on the application during a system update, the complete update procedure is broken down into several distinct steps, and allows the craft to choose the best time to begin each successive step of the update. The update may be canceled at any step of the procedure. Application-dependent processing may be introduced at any step.

Under favorable conditions only the forward steps of SUPR would be used, resulting in a successful update. These steps are:

(i) Enter new generic—Read all the new generic data onto the off-line system disk.

(ii) Proceed with new generic—Make final preparations prior to booting the system from the new generic.

(iii) Boot from new generic—Manually boot the system using the new generic.

(iv) Commit to new generic—Complete propagation of the new generic into the system after the soak period by removing all aspects of the old generic.

If the new generic does not work as expected, the craft would not commit to it, but would start a backout procedure to return to the original system.

SUPR also provides a convenient mechanism to allow application-dependent processing at each step of the update procedure. This is accomplished by transferring control to an application process that can perform whatever actions are appropriate. The types of actions most likely to be done as part of the application processing would be to transfer data (files, data bases, office-dependent information) from the old generic to the new generic or to save call registers and billing information in protected application segments prior to suspending call processing and booting from the new generic.

V. SUMMARY

This article has dealt with the subsystems of DMERT that administer changes to system data. Recent Change/Verify is used to change system configuration data and its underlying data base, Field Update allows "bug fixes" and logical file changes, and System Update will install an entirely new version of the operating system. These subsystems were described and examples given of their use. In each case DMERT provides change application, testing, and rejection or acceptance capabilities in a context very similar to that of typical operating systems, but in a highly reliable way.

REFERENCES

1. M. E. Grzelakowski, J. H. Campbell, and M. R. Dubman, "The 3B20D Processor & DMERT Operating System: DMERT Operating System," B.S.T.J., this issue.
2. D. M. Richie and K. Thompson, "The UNIX Time-Sharing System," B.S.T.J., *57*, No. 6, Part 2 (July-August 1978), pp. 1905-29.
3. B. W. Kerninghan and D. M. Ritchie, *The C Programming Language*, Englewood Cliffs, N. J. : Prentice-Hall, 1978.
4. B. R. Rowland and R. J. Welsch, "The 3B20D Processor & DMERT Operating System: Software Development System," B.S.T.J., this issue.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

3B20 Field Utilities

By G. P. ELDREDGE and J. G. CHEVALIER

(Manuscript received March 10, 1982)

The term "field utilities" describes a number of tools used by telephone company craft and support staff as well as Western Electric and Bell Laboratories field support personnel for trouble-clearing and routine maintenance activities on the 3B20D/DMERT system. This complementary set of tools provides debugging coverage for the system regardless of load or system functionality. In addition, it deals with the challenges and complexities posed by the concepts of parallel processing, virtual addressing, and swapping. This article describes the various field utilities and discusses their capabilities.

I. INTRODUCTION

The term "field utilities" includes a number of tools used by telephone company, Western Electric, and Bell Laboratories support personnel to perform trouble-clearing and routine maintenance activities. Currently, software debugging and investigation tools include the Field Test Set (FTS), the Generic Access Package (GRASP), and IBROWSE, an interactive tool used to "browse" through the contents of main memory. In unusual cases, a Micro-Level Test Set (MLTS) may be used in a troubleshooting mode. The Program Documentation Standard (PDS) Field Maintenance Commands are a collection of tools used to perform more routine operational maintenance on the operating system. Each of these capabilities will be described in this article.

II. TROUBLESHOOTING AIDS

The nature of large, evolving software projects is such that, despite multiple levels of testing by developers, integration teams, system test groups, and field site acceptance teams, some software "bugs" escape

Table I—Comparison of 3B20D/DMERT debugging tools

Attribute/Tool	FTS	GRASP	IBROWSE	MLTS
Interference	None	Small, self-regulated	Small, not regulated	Extreme
Scope of capabilities	Medium	High	Low	Medium
Debugging level	Assembly	Assembly	Assembly, source	Microcode, assembly
Limitations	Limited on kernel	No special processes or kernel	No breakpoints, no trace	Difficult with supervisor or user processes, no data breakpoints
Language	C-like	PDS, MML	ADB-Like	Terse
Target users	Bell Labs, WE	Operating Co., Bell Labs, WE	Bell Labs, WE	Bell Labs, WE
Target software needed	None	DMERT	DMERT	Microcode
Support processor software needed	*UNIX* Operating System (FTS)	None	None	None
Hardware needed	FTS, DUC, terminal	UC or DUC (optional)	Terminal	MLTS, terminal
Theater of use	Limping or loaded field site	Running, nonoverloaded field site	Running, nonoverloaded field site, off-line	Lab, dead field site

detection and are included in field releases of software. In the real-time systems used in switching, the bug may be so subtle that it may surface only under equipment configurations, telephone user actions, and/or traffic loads not easily reproduced in a system laboratory environment. System debugging tools must be available in a field site carrying live traffic to solve these problems when they arise.

The 3B20D/Duplex Multiple Environment Real Time (DMERT) operating system employs advanced computer technologies that require equally sophisticated tools to isolate errors. Parallel, time-sliced execution of processes, virtual addressing, and swapping all contribute to the need for a variety and diversity of system debugging tools. Table I is a comparative summary of these various troubleshooting tools available to field sites.

2.1 Field test set

In rare cases, a system problem could occur that leaves the system

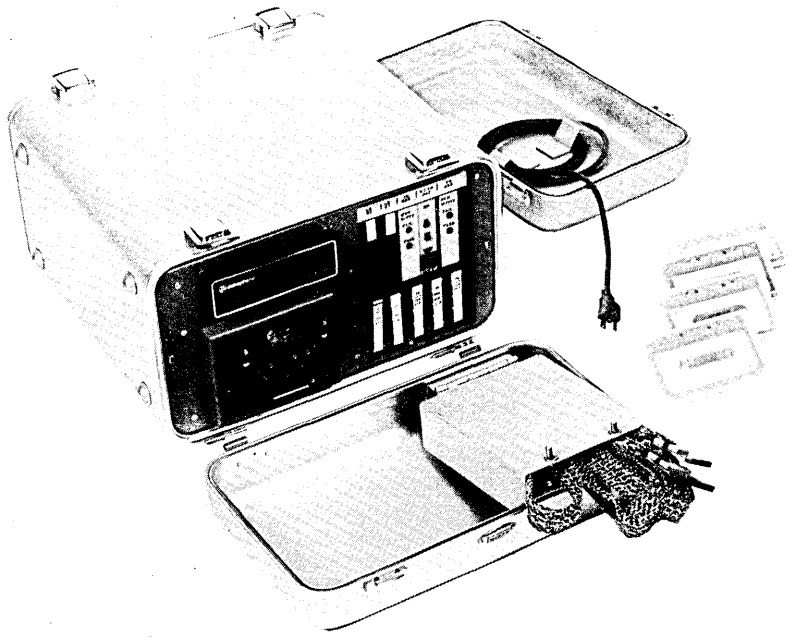


Fig. 1—Field test set.

functionally inoperative. In other cases, the traffic level may be so high that system overload mechanisms become active when unexpected results in the system indicate a software error. In either case, on-line utility systems, which assume basic functionality and nonoverload conditions, are not appropriate to isolate the problem. The Field Test Set (FTS) was designed specifically to meet this need in the field. It is strictly a monitoring device and therefore does not affect processor performance or rely on system operability. This non-interfering characteristic is extremely important when maintenance personnel are trying to isolate problems at a field site carrying a heavy traffic load.

The FTS is a small, portable unit (see Fig. 1) that is easily transported and connected to the 3B20D Processor through the Dual-Access Utility Circuit (DUC). The DUC contains hardware matchers and a 2048 entry trace memory and provides access to the processor for the FTS and GRASP (see Section 2.2). The external FTS unit connects to the DUC through an eight-foot cable. The FTS intelligence is contained in this external unit that includes a microprocessor with memory management, one megabyte of random-access memory (RAM), and a cassette transport. User access is provided through a local or remote terminal with phone access provided by the FTS.

The *UNIX** operating system was chosen as the FTS operating system. There are many advantages to using an operating system on the FTS and in particular the *UNIX* system. The FTS resident software was developed and tested as individual modules written in the high-level C language. This substantially reduced the software development time and effort. Also, the *UNIX* operating system commands provide substantial portions of the functionality required for the FTS software. Although the *UNIX* system requires disk storage for its file system, a disk system was not considered rugged enough for portability. Therefore, a "virtual disk" is supported as part of system memory. The *UNIX* operating system is booted into the system from cassette tape by resident erasable programmable read-only memory (EPROM) software. The EPROM also contains the unit's self-diagnostic software.

The FTS/DUC system supports a rich variety of trace and data-matching options. The lowest level trace, a so-called transfer trace, records program addresses of all transfers executed by a program or a range within a program. An intermediate-level function trace records program function call/return sequences. At a higher level, a record may be kept each time a different process begins execution. Multiple trace modes can be active simultaneously. Information is recorded into the trace memory under control of a variety of sophisticated matcher circuits. Masking capability is provided so that a matcher can look for a particular value of a single bit or groups of bits as well as word values. Matchers are included for address, address range, data, access type (e.g., read, write, or read/write) and process ID matching. When a matcher or a combination of matchers is triggered, a signal is produced that causes a "snap" of information into the trace memory. The matchers and matcher combinations allow very selective trace memory recording. This reduces both the size of the trace memory required and the amount of post-processing necessary to interpret the trace data.

The trace memory is operated in either a pre-trace or a post-trace mode. In the former case, the trace memory records information until it receives a stop trigger. The trace data represent program flow leading up to a particular event. In the post-trace mode, the trace memory starts recording upon receiving a trigger and stops when the trace memory is full. This provides a history of program flow after a particular event.

The DMERT operating system software is predominately written in the high-level C language. C enables the programmer to work with function-level rather than machine-level operations. To support this,

* Trademark of Bell Laboratories.

the FTS includes matching and tracing capabilities for software functions and process IDs. Function tracing records the program address and the data parameters passed on the stack to a selectable function or range of functions. Process ID matching and tracing becomes necessary in a virtual memory machine since processes are dynamically relocatable in physical memory. Processes are assigned unique ID values when they are created. The active process ID value is presented to the FTS process ID matchers and trace memory. These matchers, combined with the virtual address matchers, permit matching and tracing on virtual rather than physical addresses.

Since the FTS is an external system, it is the appropriate choice when problems must be investigated in code that has tight timing constraints or in a system that is heavily loaded. Its most attractive features are its excellent trace facility and the fact that the FTS operates in a mode that does not interfere with 3B20D operation. Although it was not designed to access machine registers or write memory, the FTS is a powerful tool in the hands of support personnel to isolate difficult system problems.

2.2 Generic access package

The concept of an on-line software debugging mechanism in real-time machines is not new.[1] Software problems may occur when the system is functional and processing traffic in a non-overload environment. Such problems can be solved in the 3B20D by use of the Generic Access Package (GRASP).

GRASP is an on-site tool for software debugging. Since it supports an interface to the DUC, GRASP provides a set of trace and data-access trap functions similar to those provided by the FTS. In addition, it provides the capability to place multiple breakpoints in code, to print the contents of memory and many machine registers, and (with some restrictions) to write memory and registers regardless of whether the DUC is available or operating correctly. GRASP has a self-regulating mechanism designed to prevent itself from taking too much real time and thereby interfering with traffic processing or driving the system into overload.

Since GRASP is "just another process" running on the machine, the design presents some unique challenges. GRASP needs to be able to identify the target process, assure that it is in main memory, and be able to gain access to its address space.

A logical process is specified by a logical tag (called a "utility ID") that is compiled into the process. All incarnations of a logical process will have the same tag since they all originate from the same object file on the disk. The tag is stored in system tables when the process is brought up and is available throughout the life of the process.

Upon a request from GRASP, the operating system searches the tables, prepares a list of real process tags (called "process IDs") for processes whose utility IDs match GRASP's request, and sends the list to GRASP. Translation between the utility ID, which is known to the craftperson, and the process ID, which is known to the operating system, is thus accomplished.

GRASP relies on cooperation with the target process to be informed when the target process is in main memory. All processes that GRASP may need to monitor must have two function calls compiled into the code, which form the run-time communication mechanism with GRASP. One is placed in the process's initial entry routine; the other is placed to execute "on demand" by GRASP.

After a process has been selected, it is forced into main memory through cooperation with the process. GRASP sends an agreed-upon event to the process; its only response to that event is to call the associated library function. That function identifies the process and notifies GRASP that it is in main memory.

Access to the target address space is then accomplished by using address translation hardware called Address Translation Buffers (ATBs). The Program Status Word (PSW) for each process is constructed to be able to handle two address spaces at one time. The identity of the address translation buffers being used by a particular process are included in that process's PSW. Instructions are provided in the instruction set to indicate which of the two address spaces to use. In addition, a special breakpoint instruction has been provided. When the breakpoint is executed by the target process, GRASP's PSW is modified so that GRASP is given access to the address space in which the breakpoint fired. This presupposes that GRASP and the target process are using different address translation buffers; that assumption is enforced by the operating system.

GRASP is especially useful when multiple breakpoints are needed (GRASP can handle up to 20), when breakpoints must be planted in several processes simultaneously, where register information is needed, or when investigation must be done remotely from a central maintenance facility.

2.3 IBROWSE

Neither the Field Test Set nor GRASP provides a mechanism to examine the kernel address space. IBROWSE, an interactive tool used only by Bell Laboratories and Western Electric support personnel, can be used to peruse the address space of any DMERT process in main memory; it fills the need to be able to view the operating system tables and message buffers in the kernel address space. IBROWSE also can be used on an off-line support processor to analyze tape dumps of main memory taken at field sites.

IBROWSE can display the contents of virtual or physical memory in a user-specified format. The user can direct that the raw machine data be represented as any combination of null-terminated strings, characters, or one-byte, two-byte (short), or four-byte (long) data types in octal, decimal, or hexadecimal format. This flexibility to specify the translation of raw data is immensely helpful when viewing DMERT data structures. IBROWSE supports the concepts of current address, next address, and current format, which are useful in displaying consecutive memory locations. It can view any process in memory, from kernel through kernel processes, supervisors, and user processes. It has the ability to search forward or backward for a specified data pattern, in either virtual or physical addressing modes. IBROWSE also supports a user-defined macro facility and I/O redirection.

The main strengths of IBROWSE are its ability to view the address space of any process in main memory and its capability to analyze data from an off-line Control Unit (CU). Since use of IBROWSE requires relatively detailed knowledge of DMERT, its users are intended to be specialized Bell Laboratories or Western Electric support personnel; for that reason, no attempt has been made to make IBROWSE part of the official DMERT release. Each time the support teams need it, they load it into the target machine.

2.4 Micro-level test set

Should a problem result in a "dead" system or one that is continually attempting automatic recovery actions and is unable to start the operating system, the Micro-Level Test Set (MLTS) is used. The MLTS is a low-level test system aimed primarily at hardware register and microcode access. It consists of an interface circuit that plugs into the 3B20D like any other board and an external control circuit. Since the MLTS is equipped with an RS232 interface and a 212A data set, it may be configured with a terminal on-site or may be operated from a remote location.

The MLTS is the only field utility tool that provides read/write access to all internal hardware and firmware registers and is the only one that facilitates access to the processor's microcode. The MLTS provides microcode breakpoints, can read and write microstore and main store locations, and can read and write machine registers that are not accessible to other troubleshooting tools. Although its primary use is in a laboratory environment, there are infrequent cases where such capabilities are required to solve problems during field tests.

III. OPERATIONAL UTILITIES

Since DMERT supports a hierarchical file system as well as the concept of processes, some types of problems must be dealt with and

resolved at the process or file-system level. For example, a process may be running when it should not be or the file system may contain some transient files that should have been cleared. The *UNIX* operating system itself provides many utilities for process control and file system maintenance; these same capabilities are needed in the Program Documentation Standard (PDS) syntax for Electronic Switching System (ESS) applications.

PDS field maintenance commands can be described in three categories:

(*i*) File system manipulation and maintenance

(*ii*) Process control

(*iii*) Magnetic tape operations that are support-processor compatible.

PDS commands are provided to allow the craft or support person to determine what files exist on the disk and what their access permissions are; the craft may alter the access permissions, add new files, or remove existing files. A basic text editor is provided to facilitate creation or modification of ASCII files. In addition, tools are provided to start a process, stop a process, and to determine what processes are known to the system.

Although these utilities do not fall into the class of "debugging" tools, they nevertheless provide a window into the system at a high level that is very useful to solve certain types of system problems.

IV. SUMMARY

Because of its architecture and technology, the 3B20D/DMERT system presents a number of challenges to those who must isolate problems in a running system in the field. Problems may be caused by hardware failures, software deficiencies, microcode errors, or operational overloads and inconsistencies. A set of tools has been developed to isolate problems that may occur in the field. Together, these utilities provide a continuum of system trouble identification capabilities for the 3B20D/DMERT system in the field.

V. ACKNOWLEDGMENTS

The authors acknowledge the contributions of Messrs. R. H. Holt, J. P. Kehn, G. A. Moore, J. D. Peterson, and D. J. Thompson, and Ms. C. A. Toman for their inputs, critiques, and reviews.

REFERENCES

1. G. F. Clement, P. S. Fuss, R. J. Griffith, R. C. Lee, and R. D. Royer, "1A Processor: Control, Administrative, and Utility Software," B.S.T.J., *56*, No. 2 (February 1977), pp. 237–54.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating Systems:

Fault Detection and Recovery

By R. C. HANSEN, R. W. PETERSON, and N. O. WHITTINGTON

(Manuscript received March 18, 1982)

The 3B20D Processor is designed to be a high-availability system for utilization in electronic switching systems. This high availability translates into the development of numerous features and capabilities for the 3B20D that distinguish it from other processors. The reliability objectives for the processor are described and related to the subsystems that have been developed to meet each objective. This article discusses processor and peripheral fault recovery, system integrity, and other software subsystems that provide the high availability and maintainability for the processor.

I. INTRODUCTION

The 3B20D Processor has extensive maintenance subsystems associated with it and is designed to meet the high-availability standards of Bell System electronic switching systems. This implies that the processor must perform within an objective of not more than two minutes downtime per service year when used in an electronic switching application. The many subsystems that have been developed to provide the high-availability capability are described in this article. In particular, software and hardware fault recovery are discussed along with the microcode assists for the recovery.

Much evolution has taken place in recovery architectures for electronic switching systems.[1,2] Earlier processor systems used extensive hardware-matching algorithms that resulted in intricate software recovery.[3,4] More recent hardware technologies have enabled the cost-effective design of processor systems with unique fault-detection capabilities.[1,5,6] These capabilities have led to much simpler recovery software. This article describes the detection mechanisms for the 3B20D and the software maintenance architecture.

II. SYSTEM RELIABILITY REQUIREMENTS

The reliability objective for the 3B20D Processor system, as with other similar systems, is to keep the overall system unavailability—i.e., the time that the system cannot be utilized by operational (call processing) functions—below 2.0 minutes per year.[7] In keeping with the ESS processor tradition, the total system downtime is allocated to four general categories: hardware, software, recovery, and procedural.

The processor has 0.4 minute per year allocated to malfunctions in the system hardware. Like other highly reliable systems, the 3B20D is equipped with redundant hardware units. Thus, failures must occur in both redundant units before the system is unable to establish a working configuration. In the case of simultaneous failures in both units, until one is repaired and system integrity is reestablished, the system is considered unavailable. This portion of the overall system downtime is a function of the failure rates of the various components (FIT rate), the system architecture, and the mean time to repair (MTTR). The hardware reliability model for the 3B20D Processor within a given application is dependent on the hardware configuration used and the maintenance technique used (this determines the repair time).

The processor has 0.3 minute per year allocated to malfunctions in the processor operational software. This is a classification of software faults that can render the system features inoperative. This allocation includes cases such as software faults that require a bootstrap to recover the system. As in the case of other high-availability systems, the 3B20D/DMERT system has a design objective of having no software failures the system cannot recover from. To help recover the system against software failures, DMERT has three levels of defenses that attempt to recover the system from such faults: hardware protection, system integrity monitor, and audits. The 3B20D Processor has several levels of hardware protection that detect the sanity of the system software. The system integrity monitor in the DMERT system has an elaborate scheme of software and hardware sanity timers as well as overload detectors that protect the system against software "resource hogs." DMERT audits include all of the explicit audits in the system as well as the defensive checks built into the common processor software. The intent of the audits is to help defend important processes against data mutilation.

The processor has 0.7 minute per year allocated to limitations in fault-recovery programs. These failures are classified by the inability of fault-recovery software to achieve a working configuration of the system due to some hardware failure condition even if a working state of the hardware is possible. These cases are characterized by the necessity for manual intervention to reestablish system integrity or by an automatic initialization to regain system integrity.

The 3B20D has a comprehensive fault-recovery scheme that attempts to recover the system from all foreseeable single hardware fault conditions. In several cases, recovery mechanisms are generated for multiple fault situations (e.g., memory failures) when that is considered to be a probable situation.

Finally, the processor has 0.6 minute per year allocated to procedural errors. This category covers cases where a craft person uses an improper maintenance procedure or follows a poorly designed procedure that results in a machine outage. The 3B20D is designed with a defensive craft interface using the PDS (Program Documentation Standards) and MML (Man Machine Language) languages.[8] The craft interface also includes emergency action and display-page capabilities that attempt to simplify the complexities of maintaining the 3B20D.

The system reliability requirements also include the various aspects of maintaining the 3B20D. These maintainability aspects include diagnostics, transient error analysis, emergency recovery procedures, routine maintenance procedures, growth and retrofit capabilities, system and process update capabilities, and field utilities. Diagnostics are provided to detect and assist the repair of classical hardware failures in the system. The diagnostic requirements include sufficient run-time performance so that a rapid repair can be carried out. Diagnostics provide greater than 90 percent fault detection.

The ability to repair circuitry exhibiting transient failures is provided through fault-recovery error reports. For example, data about transient memory faults is printed out to the craft and includes address and pack location where the error was detected. If that circuit pack continues to have a history of transient errors, the craft has sufficient information to effect a repair. Error analysis capabilities are provided on the 3B20D through the use of fault-recovery messages and error logs.

Emergency recovery procedures are provided to reconfigure the system when automatic recovery does not succeed. These capabilities allow the craft to repair the 3B20D in case of catastrophic failures. These procedures include use of the emergency action page, processor recovery message analysis, and dead-start diagnostics. Routine maintenance procedures are provided to keep the 3B20D in peak operating condition. Growth and retrofit procedures allow hardware additions and removals without affecting the system service. Finally, various utilities are provided with the DMERT system to locate system problems in field installations.

III. GENERAL RELIABILITY AND MAINTENANCE ARCHITECTURE

In this section, we provide an overview of the 3B20D fault-recovery architecture that is described in further detail in later sections. Figure

1 illustrates the hardware architecture of the 3B20D. As is indicated in the figure, the processor system has very loose coupling between any of the mate subsystems. The memory to memory update coupling is provided to keep both active and standby memories identical. This allows the switching of processors without losing the integrity of the software running on the system.

The other coupling between the processors is through the maintenance channel. The maintenance channel provides two capabilities important to the integrity of the processor. First, it provides a control and communication bus for the purpose of diagnosing the off-line processor from the on-line processor. Second, it provides low-level maintenance control for fault-recovery programs so that a switch in processor activity can be carried out with no operational interference. In addition, other maintenance controls can be exerted over the channel to start an initialization sequence on the other processor or to stop execution on the other processor. One other coupling, the Dual Duplex Serial Bus Selector (DDSBS), allows either processor to talk to any peripheral controller. Thus, no matching techniques are utilized between major subsystems or peripherals in the 3B20D for the purposes of fault detection in the hardware. This means that unique fault-detection techniques are essential in each subsystem of the 3B20D.

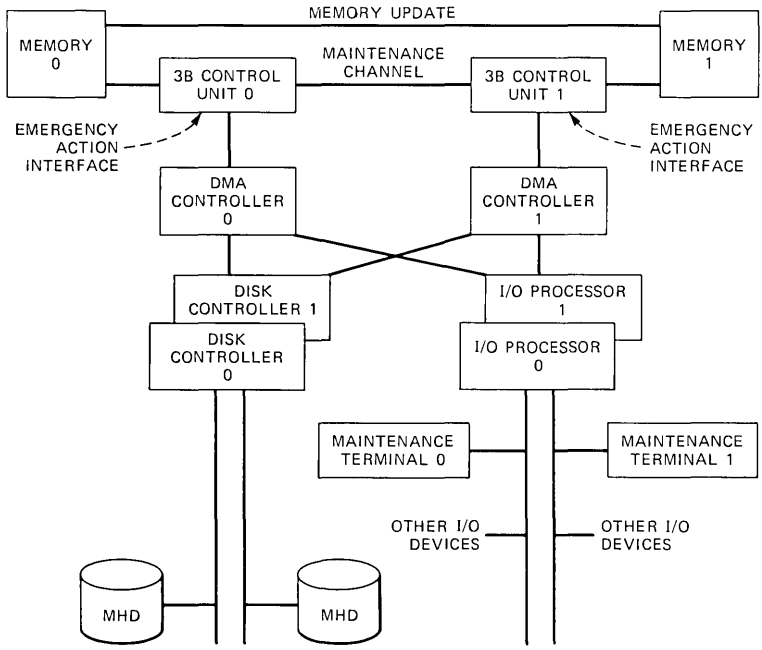


Fig. 1—The 3B20D system architecture.

To provide these detection algorithms, extensive use of local matching circuits, parity techniques on all buses, Hamming detection with single-bit error correction on the main store, cyclic redundancy codes on the disks, and numerous sanity timers throughout the control unit and peripherals are used as the primary fault-detection techniques. In addition, routine diagnostics are used to detect failures in the fault-detection hardware itself. Other routine sanity checks are used to ensure that peripheral subsystems are healthy. Finally, system-integrity checks catch certain subtle problems that are not caught by unique detectors.

3.1 Fault-recovery architecture

When any of the unique detectors determine an error condition, an error interrupt (or error report in the case of certain peripherals) is registered in the processor. The most severe of these will result in automatic hardware sequences that switch the activity of the processors (hard switch). Less severe errors result in microinterrupts that enter microcode and software charged with recovery of the system.

The microcode and recovery software provides a layered approach to the recovery architecture. Figure 2 illustrates this architecture with microcode providing low-level access to the hardware while the recovery software provides the high-level control mechanisms and decision making. This technique has resulted in several hardware design modifications requiring minimal change to the recovery software.

Figure 3 illustrates the principal architecture and features provided by the recovery software. The bootstrap and initialization routines provide a fundamental set of microcode and software algorithms to control the processor initialization and recovery. These actions are stimulated by a Maintenance Restart Function (MRF), which repre-

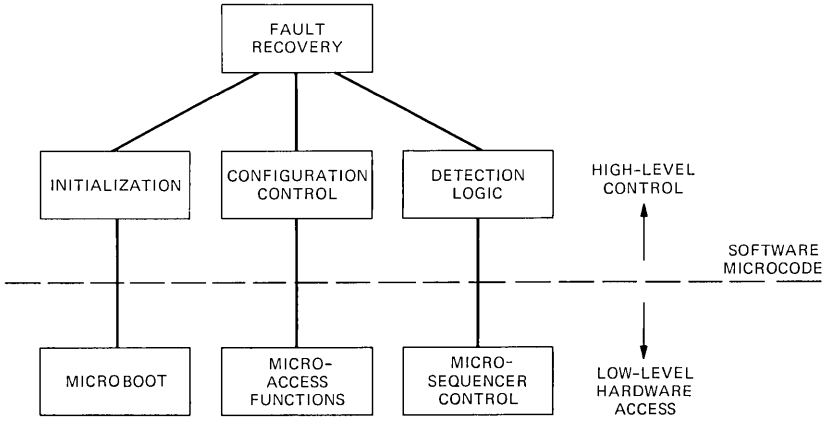


Fig. 2—Maintenance software structure.

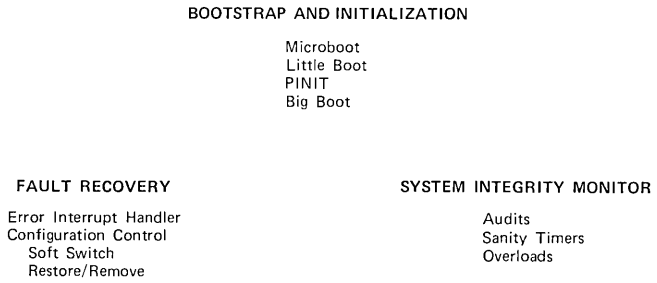


Fig. 3—Maintenance architecture.

sents the highest priority microinterrupt in the system. An MRF sequence can be stimulated from either hardware- or software-recovery sources.

The fault-recovery and system-integrity packages control fault detection and recovery for hardware and software, respectively. The Error Interrupt Handler (EIH) is the principal hardware fault-recovery controller. It receives all hardware interrupts and controls the recovery sequences that follow. The configuration-management program (CONFIG) determines if this particular error is exceeding predetermined frequency thresholds. If a threshold is exceeded, CONFIG requests a change in the configuration of the processor to a healthy state. Thus, CONFIG serves as an error-rate analysis package[10] in the 3B20D maintenance architecture for both hardware and software errors.

3.2 Software integrity architecture

Software fault recovery is very similar in architecture to hardware fault recovery. Each major unit of software is expected to have associated with it error-detection mechanisms (defensive checks and audits), error thresholds (provided by the system-integrity monitor and CONFIG), and error-recovery mechanisms (failure returns, data correcting, audits, and initialization techniques). In addition, both SIM (System Integrity Monitor) and EIH oversee the proper execution of the process. SIM ensures that a process does not put itself into an infinite execution loop or excessively consume some system resource (e.g., message buffers). EIH, through the use of hardware and microcode detectors, ensures that processes do not try to access memory outside of defined limits or execute instructions that are not permitted to the process. Each process has initialization and recovery controls (analogous to hardware) so that a recovery can be effected. Figure 4 illustrates the software-recovery architecture.

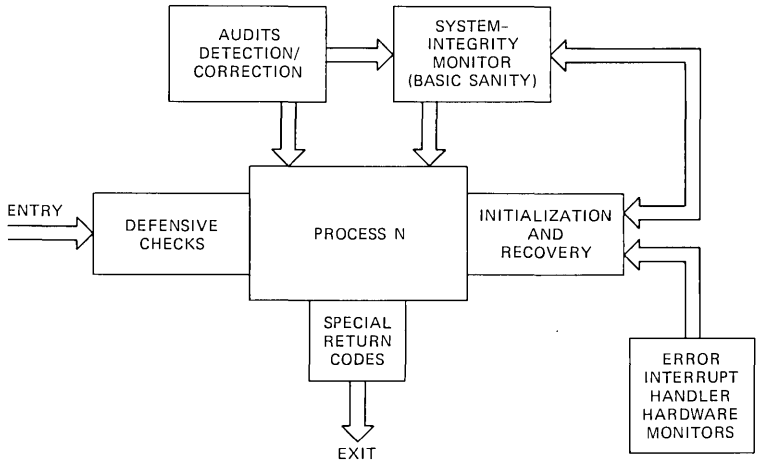


Fig. 4—Software fault-recovery architecture.

If recovery actions result in the removal of hardware units from the system, diagnostics[9] are dispatched automatically to analyze the specific problem. Audits represent the software counterpart for diagnostics with the exception that the routine interval is much shorter and correction is possible in the case of audits.

IV. FAULT RECOVERY

In this section, we describe the fault-recovery strategies associated with the 3B20D Processor. In particular, we describe the fault recovery and initialization strategy along with the microcode assists required to carry out these functions. We also describe the manual control capabilities provided by the processor and software. These control mechanisms are termed emergency mode. Finally, we describe some of the software audit and integrity techniques in the DMERT operating system.

4.1 Fault recovery and system initialization

Fault-recovery strategies are based on the fault-tolerant architecture of 3B20D. Major hardware units are fully duplicated. This duplication provides a high probability that a combination of operational units can be retained in the face of faults. The mate processors are only loosely coupled; interprocessor connections are limited to the maintenance channel and memory-update circuitry. This architecture forms the foundation of the hardware-recovery strategy employed in the 3B20D, namely to isolate an entire faulty processor as opposed to attempting fault resolution at the subunit level.

DMERT is a modular operating system that provides a wide range

of protection from various types of classical errors. Examples include write-protected memory areas, memory ranges that can be used only for text execution, and protected virtual address spaces. Thus much of the recovery from these types of errors is built into DMERT directly. Those overt recovery actions that are required are greatly simplified by the underlying architecture. Hard faults and other conditions requiring recovery actions are treated according to their severity. Fault categories that will be described individually are hard faults, thresholded faults, configuration faults, sanity time-outs, and software-requested recovery actions.

The 3B20D has built in self-checking circuitry designed to detect hard faults as soon as they occur. This circuitry simplifies recovery since early fault detection limits the possible damage done by the fault. Faults in this category indicate that the processor is no longer capable of proper operation and results in an immediate stop of the currently running processor and a switch to the standby processor. Since the standby processor does not match the active processor instruction by instruction, an initialization sequence is required to start execution properly.

Some types of faults and errors are not severe enough to justify an immediate stop and switch recovery action. Examples of errors of this kind are hardware faults detected in the standby processor memory and software errors such as write-protection violations. Another type of error in this category is hardware faults that are handled by self-correcting circuitry. Although most faults are detected by self checking, some units, such as main memories, have fault rates that justify self-correcting capabilities. Disks also are self correcting through the use of cyclic redundancy codes. All errors in this class are reported to the recovery system as error interrupts.

Recovery software classifies the interrupt by type, gathers and saves all available information about the interrupt, and reports the error to the system configuration-management package. If a particular software process is suspect as the cause of the interrupt, such as in a software-triggered event, the process that was running at the time of the interrupt is faulted and entered at its fault entry after a stable system configuration is guaranteed. The fault entry of a process contains recovery and initialization sequences that are special to the process involved.

All error interrupts are reported to configuration management. Errors are logged against the failing unit and error rates are compared to allowed error thresholds. If the affected threshold is exceeded, further action is required and is based on several factors. If the faulty unit is essential to the system and a mate unit is available, the faulty unit will be removed from service and scheduled for diagnostic testing.

If there is no available mate unit, the faulty unit will be initialized and returned to service since in the case of essential units it is better to have a faulty unit than no unit. Nonessential units are removed and scheduled for diagnostic testing whenever their error thresholds are exceeded.

Each processor has a sanity timer that will result in an initialization if it times out. The active processor maintains both its own and the standby sanity timer so that if the active processor is completely dead, an initialization of the standby processor will be triggered by a sanity timer time-out.

The system provides an Operating System Trap (OST) for use by software to request an initialization. This capability is used by critical system processes when they encounter errors that preclude performance of a critical system function. Initializations occur when an error or fault has been detected that cannot be recovered from without a change in hardware and/or software status. A stop and switch to the other processor may or may not be associated with any given initialization. All initializations include actions of varying severity depending on what is required to deal with various faults and errors.

The first event in the initialization sequence is a hard-wired transfer to a fixed location in the CU microstore where microcode makes a decision as to whether to bring this processor on-line or to switch to the other processor. If the current initialization is of level two or higher, the appropriate processes and data bases are loaded from disk. All available data about the initialization trigger is saved and a decision is made to bring this processor on-line or stop for the off-line initialization.

The DMERT kernel initialization or bootstrap routine is then called to restart system processes or to fault active processes as appropriate. The initialization is now complete and the system has returned to normal operation. If an initialization does not recover the system to an operational state, another and more severe initialization will be triggered automatically. Whether to escalate or not is controlled by the initialization interval. Any initialization that occurs during a window of time following the previous initialization will escalate to the next higher level. The length of the initialization interval is a system generation parameter that is established by the application. In addition to the DMERT initialization levels, provision is made for an application to specify between one and sixteen levels for each DMERT level. For example, if the application specifies two levels for DMERT level one, the normal execution of initialization levels would be (1,1), (1,2), (2,1), $\cdots$, where the first number indicates the DMERT level and the second number is the associated application level.

Data about various recovery actions taken by the system are sup-

plied to provide all possible information about what went wrong and to provide data that can be used by maintenance personnel to assist them in isolating difficult faults. Recovery data are provided in several forms. Each error interrupt is accompanied by a printout containing available information about the state of the processor when the interrupt occurred. A more difficult problem is presented by initializations. Since they are more severe than interrupts and in fact represent a discontinuity in processing, gathering and preserving error data is more difficult. Initializations, as well as interrupts, can occur at a rate much too fast for data to be printed. The solution is to save all pertinent data in a protected area of memory for printing after the system has recovered.

Various kinds of error data are not generally printed as a part of the standard system output but instead are saved in error files on the system disks. Examples of this kind of data are device-driver errors and failing-memory data. One of the more useful pieces of data output by the system are Processor Recovery Messages (PRM). These are low-level one-line messages that are printed in real time. The PRMs thus represent progress marks through the recovery sequences and are extremely useful in those cases where stability cannot be achieved or postmortem data cannot be gathered.

4.2 Special microcode for recovery

A large fraction of the total amount of CU microcode is provided to aid in recovery. The bulk of this recovery microcode is in PROM because most functions are required in spite of the power history of the CU or its boot devices. Functions that are required even if the CU is not ready to execute its instruction set include microinterrupt processing, maintenance channel assists so that one processor can access the other processor and microcode to initialize hardware subsystems. Additional recovery microcode that resides in writable microstore (WMS), extends the processor's instruction set to provide convenient diagnostic and recovery software access instructions. When diagnostic performance requirements do not justify a special instruction, a microstore scratch area is available that can be loaded with arbitrary microsequences that can then be executed for special tests or functions. Before software can run, the WMS must have been loaded from disk. This happens initially as part of the processing of the MRF microinterrupt.

4.2.1 MRF and microboot

When a maintenance restart interrupt occurs, a long sequence of microsteps begins to establish system sanity. Both processors may be in their MRF sequence at the same time and each one may try to

become the active processor. The MRF code first makes decisions on whether to do an off-line initialization or an on-line initialization. If a processor determines that it has just powered up, it clears main store and does an off-line initialization unless forced on-line.

A number of tests are made on data in the system status register, SSR, to select one of four possible actions: microboot, tapeboot, processor initialization, or stop and switch. The simplest actions are to begin execution of a processor initialization program called PINIT or to stop and switch to the other processor. This is accomplished by sending a switch command over the maintenance channel to the other processor.

Tapeboot is a complex sequence of microcode that is only done when requested manually via the craft interface. Its function is to create a new system disk from tape. Using the tape device and disk device selected by the craft interface it initializes those I/O units and initializes the WMS from tape. A boot program, called load disk from tape, is read from tape into main store, and memory-management tables are created to allow it to run the hardware complex without the operating system. This program then reads the tape to make a DMERT disk image.

Microboot uses information on the DMERT disk to initialize the writable microstore and read in the first software boot program called little boot. To do this, it must first select the disk drive to use as a boot device. If the craft interface has forced either the primary or secondary boot device, it uses that device. Otherwise, microboot selects a disk drive based on the state of hardware control bits. Alternate boots will use alternate devices. Microcode is read from the disk and then copied to WMS. Finally, little boot is read from the boot partition and given control.

4.2.2 Microaccess assists

Although the MRF sequence is the most complex microcode recovery assist, both diagnostics and recovery software have special microcode. There are six maintenance channel assists in PROM. They are:

Write Main Store
Read Main Store
Write Writable Microstore
Read Microstore
Write Utility Circuit
Read Utility Circuit

In addition there is microcode in WMS to support a set of instructions provided for the diagnostic and recovery software. Diagnostics

have instructions to manipulate the maintenance channel and aid in I/O diagnoses. They also share instructions with recovery. These instructions include groups of instructions for:

On-Line Main Store Controller
Off-Line Main Store Controller
Maintenance Store Operations

Finally, both diagnostic and recovery software use privileged instructions (shared with the operating system) to read or write special registers. They also can activate unit initialization sequences that are used in the various parts of the MRF microcode.

4.3 Emergency modes

Emergency mode on the 3B20D refers to the facilities and procedures provided to prevent the system from experiencing a total outage. For example, emergency facilities are applied when the system is unable to recover automatically. The most characteristic of these are: duplex failure of the control unit, duplex failure of the system disks, duplex failure of the essential I/O devices, failure of fault recovery to find a working configuration of hardware, software faults that will not allow the system to operate properly, errors that destroy the integrity of the disks, and software overwrites that introduce catastrophic errors into the software.

Emergency mode capabilities are built into the system to address these mechanisms that can fail the 3B20D as a system. The emergency action interface (EAI) on the 3B20D provides for manual initialization capabilities that can recover the system from several of the conditions mentioned above. This interface allows the craft to select a processor and disk configuration in a case where certain configurations may be leading to the problem. The interface also allows the craft to reconfigure the system to handle maintenance hardware failures. For example, the craft can inhibit error sources and sanity timers through EAI commands, thus allowing recovery from certain maintenance failures even though both processors are affected. The EAI also provides capabilities for craft initializations to deal with loss of subsystem capabilities.

The 3B20D also provides the craft with other manual capabilities through the port switch select, the disk power inverter select, and the unit power switches. These can be used to reconfigure the system to handle certain problems in the system. In rolling bootstrap conditions, the 3B20D outputs diagnostic information through processor recovery messages. This information provides a gross diagnostic capability in the event of a complete system outage.

Tape load boot is an emergency procedure provided for the situation where a system has destroyed its only valid copies on disk of the generic software. Here the site would have a tape copy of the generic and data base, and read the tapes into the disk via the EAI tape load boot facility.

The final backup repair procedure is the dead start diagnostics. Primarily used as an installation tool, the dead start diagnostics allow for the repair of a completely sick processor by using a remote host processor.

The craft interface provides the mechanism through which the status of the system can be determined, the configuration of the system's hardware or software can be changed, and special emergency actions can be taken during catastrophic failures of system component.[8] The emergency action interface (Fig. 5) allows the craft in the field to access the system during times when a major portion of the system is nonfunctional to the point where the normal capabilities of the craft interface cannot be used. The limited capabilities of the emergency action interface include forcing failing hardware off-line or on-line, notification of the status of critical system resources, and forcing a reinitialization of the system.

4.4 Software integrity

Section III described the architecture of the software integrity system. In this section, we describe some of the specific audit and overload measures that have been included in the DMERT system.

The DMERT audit package verifies the validity of critical data structures. Most audits exist throughout the system within the processes that control the data to be audited. In some cases, several audits are invoked consecutively to form a sequenced mode audit. Most requests for running audits come from an audit control structure, i.e., the audit manager.

Audits in DMERT verify data, not functions. The basic types of auditable data are system resources and stable data. Though most of the auditable data in the operating system resides in the kernel, additional data resides in other critical processes, such as the file manager and device drivers. Smaller amounts of auditable data reside in supervisor processes, such as in the *UNIX** operating system and the process manager.

Some audits, scheduled on a regular basis, are known as routine audits; others, scheduled on request, are known as demand audits. A partial list of the DMERT audits follows:

---

* Trademark of Bell Laboratories.

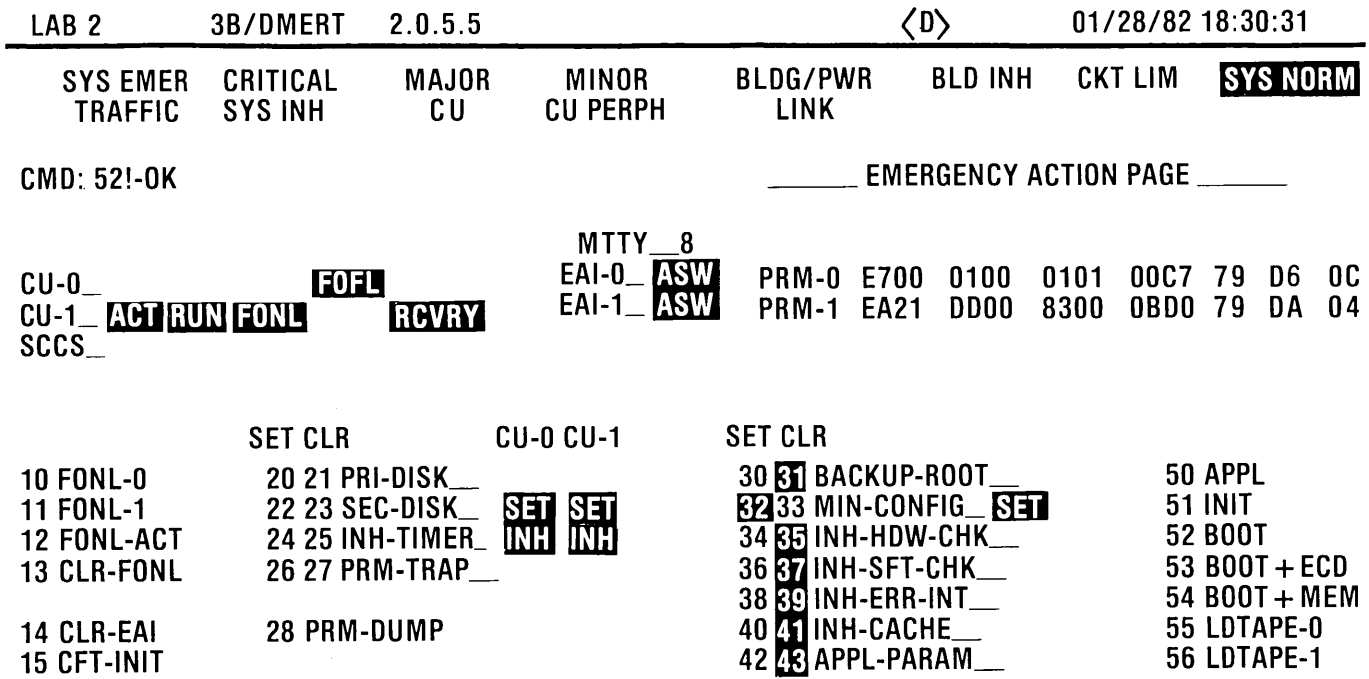


Fig. 5—Emergency action interface page.

(*i*) Message buffers—This audit finds and frees lost message buffers, i.e., messages that have been on a process's queue for extended periods of time.

(*ii*) Scheduler—This audit checks for linkage errors in the scheduler's ready and not-ready lists.

(*iii*) Memory manager—This audit recovers lost swap space and corrects any overlap of swap space.

(*iv*) File manager—This audit checks all internal file manager structures: task blocks, buffers, mount table, etc. The audit corrects the information and has the ability to back out an aborted task and free its resources.

(*v*) File system—This audit is demanded by the file manager whenever a file system is mounted read/write. It checks and corrects the file system's super block free list, and free-block bit map. This audit verifies the integrity of the mounted file systems concurrent with their use.

V. MAINTAINABILITY

The maintainability of the 3B20D system is the second vital component that guarantees the overall high reliability required of the system. There are conditions where automatic recovery is unable to restore the system to a fully functioning state. This is where maintainability is critical to satisfying DMERT's high-reliability requirements. The basic premise of maintainability is to provide basic data-gathering and data-analysis mechanisms as well as the ability to act on the results of that analysis. These mechanisms must be able to collect and analyze diagnostic and debugging information from various hardware and software components within the system in order to isolate the error. These mechanisms must then allow the craft to control and modify the configuration of the system based on the diagnostic and debugging information collected. Furthermore, these mechanisms must yield their information as quickly as possible while disturbing the rest of the system as little as possible.

Maintainability comprises such areas as diagnostics, transient-error analysis, routine maintenance procedures, field utilities, and plant measurements. Once the error has been isolated and analyzed, the problem must then be corrected as quickly and benignly as possible. This procedure is termed updatability, and it includes such aspects as growth and retrofit for hardware, emergency fixes, function update, and system update for software. Maintainability is quite naturally partitioned into diagnostics (hardware) and the various field utilities (software).[11] However, central to the ability of the craft to maintain and control the 3B20D hardware and software is the ability to interface to the various maintenance facilities provided within the system. This

is one of the very important capabilities of the craft-interface system. The craft interface provides the craft and others with the means to request diagnostics, receive error-analysis reports, initiate emergency-recovery procedures, gather plant-measurements data, and exercise routine maintenance programs. In addition, the craft-interface system allows configuration control by providing access to growth and retrofit procedures, system- and function-update capabilities, emergency-fix facilities, and the various field utilities. This section discusses the capabilities of the subsystems, which provide basic maintainability of the DMERT system. Diagnostics are discussed in Ref. 9.

One component of the maintainability required of DMERT-based systems is the ability of these systems to accept hardware and software changes in a way that does not interfere with their primary tasks. In other words, a DMERT-based system must be able to accept changes without disturbing call processing, networking, or other critical functions. DMERT supports this through several aspects of updatability. The first is growth; the ability to add or remove hardware and related software components to the running system. Growth extends from physically connecting new equipment—such as memory boards—through informing the system of its existence, exercising it, logically connecting it into the system's configuration, and committing its use in the system. Other subsystems—such as a hardware and software fault recovery and diagnostics subsystem—then take over to ensure that the new system component continues to be sane and usable.

The second aspect of updatability is retrofit: the ability to replace hardware components in the system with similar components of a different vintage or with different capabilities or interface characteristics. Retrofit procedures may "de-grow" or remove old units and then grow or add new ones. They also may add the new units first and then perform a transition from the old units to the new. Thus, retrofit of units may involve extensive periods of time where old and new units coexist in the system. Retrofit may also involve substantial software changes to interact with new units and to recognize the existence of both old and new units.

The third component of updatability, field update,[12] deals exclusively with software and data file changes in DMERT. Such changes are done logically, on a file-by-file or functional level. Just as with growth and retrofit, field update can install or replace system programs or files, inform the system about them, logically connect them into the system, exercise them in that state, and then commit to or back out of them. Field update is intended primarily for installing fixes or small features that do not perturb the system's architecture.

The fourth updatability component, system update, allows program and data changes of much greater magnitude, up to complete software

system replacement. A bootstrap is required to install the changes for any system update. By using disk redundancy or backup copies of sections of DMERT's disks, system update can prepare a new, partial, or total version of the system on disk and then switch to it (and back, if necessary). Where field update performs a logical change of files, system update does a physical change of a set of partitions (file systems and/or file partitions).

VI. SUMMARY

This article has described the basic architecture of the fault-recovery and system-integrity subsystem for the 3B20D Processor. These subsystems are tied into the maintainability aspects of the processor. All of the features provided are responses to the reliability objective of no more than two minutes downtime in each year of service. The features and architecture continue in the tradition of former high-availability processors.

VII. ACKNOWLEDGMENTS

The authors thank D. G. Gilbert, G. T. Surratt, B. G. Niedfeldt, and D. J. Fitch for their assistance with various sections of this article.

REFERENCES

1. R. C. Hansen, "System Reliability Strategies," Proc. Nat. Elec. Conf., *35* (1981), pp. 40–51.
2. P. D. Carestia and F. S. Hudson, "No. 4 ESS: Evolution of the Software Structure," B.S.T.J., *6*, No. 6 (July 1981), pp. 1167–1201.
3. R. W. Downing, J. S. Nowak, and L. S. Tuomenoksa, "No. 1 ESS Maintenance Plan," B.S.T.J., *43*, No. 5 (September 1964), pp. 1961–2019.
4. P. W. Bowman et al., "1A Processor: Maintenance Software," B.S.T.J., *56*, No. 2 (February 1977), pp. 255–87.
5. T. F. Storey, "Design of a Microprogram Control for a Processor in an Electronic Switching System," B.S.T.J., *55*, No. 2, (February 1976), pp. 183–232.
6. M. W. Rolund, J. T. Beckett, and D. A. Harms, "The 3B20D Processor & DMERT Operating System: 3B20D Central Processing Unit," B.S.T.J., this issue.
7. Jonas Butvila, "Reliability and Its Impact on System Design," Proc. Nat. Elec. Conf., *35* (1981), pp. 43–7.
8. M. E. Barton and D. A. Schmitt, "The 3B20D Processor & DMERT Operating System: Craft Interface," B.S.T.J., this issue.
9. J. L. Quinn and F. M. Goetz, "The 3B20D Processor & DMERT Operating System: Diagnostic Tests and Control Software," B.S.T.J., this issue.
10. M. M. Meyers, W. A. Routt, and K. W. Yoder, "No. 4 ESS Maintenance Software," B.S.T.J., *56*, No. 7 (September 1977), pp. 1139–67.
11. G. P. Eldredge, and J. G. Chevalier, "The 3B20D Processor & DMERT Operating System: Field Utilities," B.S.T.J., this issue.
12. R. H. Yacobellis, J. H. Miller, and B. G. Niedfeldt, "The 3B20D Processor & DMERT Operating System: Field Administration Subsystems," B.S.T.J., this issue.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

Diagnostic Tests and Control Software

By J. L. QUINN, R. L. ENGRAM, and F. M. GOETZ

(Manuscript received March 10, 1982

Comprehensive diagnostic tests and multifeatured control software designed for execution on several host processors help craft to quickly isolate faulty hardware anywhere in the 3B20D Processor. Besides meeting the requirements for Bell System switching systems, the 3B20D diagnostics provide a high degree of modularity and portability using an operating-system-based structure. The diagnostics are used in a wide range of development, production, and maintenance activities throughout the project life cycle. Many features of the system architecture and hardware are provided to allow thorough diagnosis in a time-shared noninterfering manner. Additional features are provided in the diagnostic control structure to extend the DMERT diagnostic capabilities to application systems based on the 3B20D Processor.

I. INTRODUCTION

Many of the diagnostic principles and features embodied in the 1A Processor[1] have been incorporated in the maintenance design for the 3B20D Processor. These design principles include: (*i*) use of a special-purpose test-design language that facilitates test interpretation; (*ii*) use of a table-driven control program approach; (*iii*) use of a common test data base covering all hardware versions of the 3B20D Processor; (*iv*) partitioning of diagnostic tests into phases associated with specific hardware functions; (*v*) control features allowing selective test execution and variable degrees of detail in outputted results; and, (*vi*) optional automatic trouble location. In the 3B20D Processor design, however, a more general diagnostic design approach was followed. This approach resulted in a more portable diagnostic control structure;

it allowed diagnostic execution in several environments: factory testing using a support processor, installation testing using a remotely located processor, and in-service, on-line testing of a standby mate processor.

II. OBJECTIVES

As with earlier processor designs, the 3B20D Processor diagnostics must be effective and efficient in fault detection, provide consistent test results, protect the contents of memory, be noninterfering with normal system operation, allow automatic trouble location, and be easy to maintain and update. In addition to meeting these objectives, the 3B20D diagnostics were required to be:

(*i*) Portable—The diagnostic software must execute in several environments. Throughout this paper the execution environment is referred to as the host processor (or computer).

(*ii*) Flexible—The diagnostics would test multiple system configurations containing various vintages of circuits.

(*iii*) Modular—Standard control interfaces must accommodate differing test access facilities to the processor under test, input/output facilities, and DMERT application processes that are used to diagnose application-dependent hardware that interfaces to the 3B20D Processor.

(*iv*) DMERT compatible—Diagnostics must be integral with, rather than separate from, the operating system.[2,3]

To meet these design objectives, the diagnostic control structure was designed as an integral part of the operating system and had to support the evolutionary stages of development.

III. DIAGNOSTIC ENVIRONMENTS

As shown in Fig. 1, the 3B20D Processor can be diagnosed from several execution environments. During the early phase of processor development, a local host computer was used to support hardware, software, and diagnostic design. This access arrangement continues to be used in factory testing. Later in the development, more efficient use was made of the host computer by providing access to a remote target 3B20D Processor over a dial-up telephone line. Ultimately, in the standard duplex-system configuration, the active control unit is capable of diagnosing its own peripheral controllers and the standby control unit. Each of these access arrangements is discussed below.

3.1 Local host diagnostics

Figure 1a shows three local-host access arrangements. In the first, diagnostic programs executing in a host computer send test inputs and receive test results over a standard communications port to a Micro-level Test Set (MLTS). The MLTS connects directly to the 3B20D

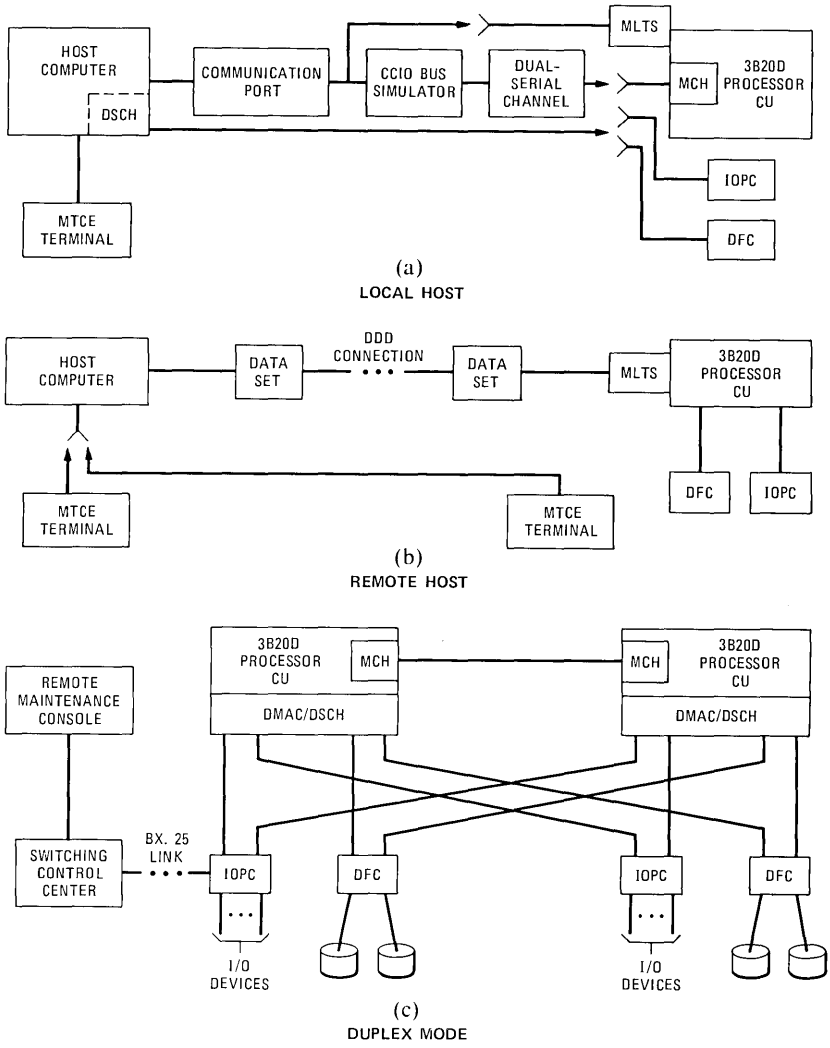


Fig. 1—The 3B20D Processor diagnostic environments.

control unit backplane, and provides complete access and control of the processor's microprogram control circuitry. For the second access path, a circuit was designed to simulate the Central Control Input/Output (CCIO) internal bus. The CCIO Bus Simulator (BS) is accessible using a standard communication input port. A Dual Serial Channel (DSCH) connected to the CCIO/BS can then communicate directly with a Maintenance Channel (MCH), the circuit designed for control-unit access. Like the MLTS the MCH can access the central control at a low level. However, only the MCH is used in the duplex configu-

ration (see Section 3.3); it communicates with either another MCH or a DSCH. As shown, the CCIO/BS–DSCH access path can also be used to diagnose the Input/Output Processor Controller (IOPC) and the Disk File Controller (DFC). Notice that when the local host is a 3B20D Processor, the path is from the DSCH of the host 3B20D Processor to the MCH, IOPC, or DFC of the target machine.

3.2 Remote host diagnostics

The DSCH is designed to communicate over distances of approximately 100 feet. Remote-host (Fig. 1b) access arrangements can be used for diagnosing over longer distances. Using data sets and a telephone line, tests stored and executed on a remote computer can be applied through the MLTS to the control unit. Peripheral controllers (IOPC and DFC) can also be diagnosed by downloading tests into the control unit and executing them. Although remote-host diagnostics are useful in cases where a local host is unavailable, execution performance is limited by the transmission facilities used.

3.3 Duplex mode diagnostics

The primary diagnostic execution environment is the 3B20D Duplex Processor (Fig. 1c). The active (on-line) processor acts as a local host for diagnosing the standby (off-line) processor. An MCH-to-MCH link provides the access path for testing the control unit. In the duplex mode, the DFC and IOPC are diagnosed from the on-line control unit using the operational interface path, a DSCH attached to the Direct Memory Access Controller (DMAC). Tests of the links from the off-line processor to the peripherals also can be run under the control of the active processor. As shown in Fig. 1c, the duplex system configuration also supports remote monitoring and control of diagnostics over a dedicated link to a Switching Control Center (SCC).

3.4 Multiple-target processors

Although the target processor is always a 3B20D Processor, it can be of many types, versions, and sizes. The diagnostic control program accounts for these differences by referencing the Equipment Configuration Database (ECD). All information relevant to the particular diagnostic tests that should be applied to each hardware unit is contained in the ECD. This information includes the name of each hardware unit within a subsystem, subunits, and their logical interconnections, equipage options, and auxiliary information, such as channel address and baud rate. Whenever a circuit design is originated or updated, diagnostic tests are designed and appropriate ECD changes are specified.

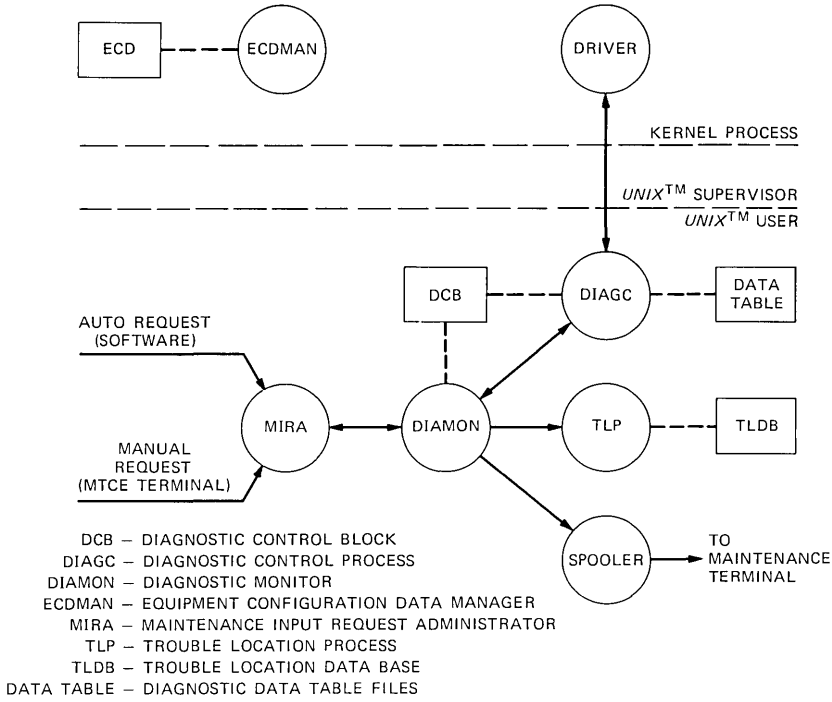


Fig. 2—Diagnostic control structure.

IV. DIAGNOSTIC CONTROL STRUCTURE

The diagnostic control structure is depicted in Fig. 2. At the kernel process level are the modules that provide access to the ECD or drive the communication links previously discussed. The *UNIX** operating system supervisor resides at the supervisor level,[3] and provides a protected environment and operating system services for the higher-level processes. The modules operating under the *UNIX* operating system that pertain exclusively to diagnostics are: the Maintenance Input Request Administrator (MIRA), the Diagnostic Monitor (DIAMON), the Diagnostic Control process (DIAGC), and the Trouble Locating Process (TLP). Output messsages from the diagnostic structure are sent to the system spooler for printing. The first three of these modules are discussed below; the TLP is described in Section VII.

4.1 MIRA

Scheduling and dispatching maintenance requests is the function of MIRA, the front-end process of the diagnostic structure. MIRA main-

* Trademark of Bell Laboratories.

tains a waiting queue and an active queue to administer each service request. Requests are serviced according to priority and resource availability; manual requests have higher priority than those initiated automatically. For each service request, MIRA spawns a DIAMON process and sends it a message. When the request is completed, DIAMON sends a message back to MIRA. Interfaces are provided in MIRA to administer routine exercise requests and inputs from the error-interrupt handler.[4]

Nine general types of request are handled by MIRA; they are described in Table I.

4.2 DIAMON

Execution of each diagnostic is directed from start to finish by DIAMON. Specifically, DIAMON will:

(*i*) Initiate and control the actual diagnostic processes specified in a message from MIRA.

(*ii*) Communicate with the Equipment Configuration Database Manager (ECDMAN) and the appropriate device driver (the software control module for a particular hardware unit) to extract control data from the ECD and retrieve path names of related utility files.

(*iii*) Build the diagnostic control block containing all the data required by a diagnostic.

(*iv*) Spawn the appropriate diagnostic control process (DIAGC); separate processes are provided for the control unit and peripherals.

(*v*) Communicate diagnostic output to MIRA and the output spooler.

(*vi*) Spawn the remove and restore processes.

(*vii*) Interface with the TLP.

Table I—Description of diagnostic requests to MIRA

Command	Description
Diagnose (DGN)	Diagnoses the unit specified in the request.
Remove (RMV)	Removes the specified unit from service.
Restore (RST)	Diagnoses the unit and restores it to service if all tests pass (ATP).
Restore Unconditional (RSTU)	Restores the unit to service without running the diagnostic.
Exercise (EX)	Starts the diagnostic in the interactive mode. This command allows stepping to a particular test, pausing, or looping over a diagnostic "phase" (ie., group of functionally related tests) segment.
Terminate (STOP)	Stops execution of a diagnostic.
Display (OP)	Displays status of queued requests in MIRA.
Inhibit (INH)	Inhibits diagnostic requests from other processes that automatically or routinely initiate diagnostics.
Allow (ALW)	Allows diagnostic sources, canceling any active INH request.

4.3 DIAGC

DIAGC is a generic name that refers to a class of processes. The DIAGC is a unit or application-dependent module that controls execution of tests. Containing all unit-dependent task routines, DIAGC translates the interpretive diagnostics and provides the interface with DIAMON. A unit's diagnostic phase table (DPT) contains the name of a particular DIAGC process to be used in the diagnosis. DIAMON imposes no limit on the number of processes that can interface with it. The following functions are provided by DIAGC:

(*i*) Opens the diagnostic driver
(*ii*) Shares the buffer (DCB) provided by DIAMON
(*iii*) Initializes the raw data buffer
(*iv*) Executes the diagnostic
(*v*) Computes the test results
(*vi*) Provides interactive control if required
(*vii*) Provides an interface to DIAMON for test results and abnormal terminations (aborts).

4.4 Portability

All diagnostic control modules are written in the C language and execute in the *UNIX* operating system environment. This facilitates the porting of the control structure to other host processors that support C and *UNIX* operating system software. Variations of processor configuration and hardware vintage can be described in the ECD. DMERT application processes can provide additional DIAGCs and Data Tables to control diagnostic test execution for interfacing hardware. Several driver processes can be supported to allow diagnostics to be executed over standard communication ports, dual-serial channels, or maintenance channels.

V. MAINTENANCE FEATURES

The combination of hardware-access circuits and modular-control programs, just discussed, provides the 3B20D Processor with considerable maintenance flexibility. Tests are selected according to the vintage of circuit under diagnosis. Displayed in Fig. 3 is a typical diagnostic input message in the PDS (Program Documentation Standards)[5] syntax, one of three command languages supported by DMERT. Requests can be made to diagnose an entire unit, a particular subunit, or all subunits in a specified community. Individual test phases or ranges of phases can be executed and the results printed with optional amounts of detail. Some diagnostic test phases—because of either their long execution time requirements or their dependency on other system hardware availability—are restricted to manual initiation. In-

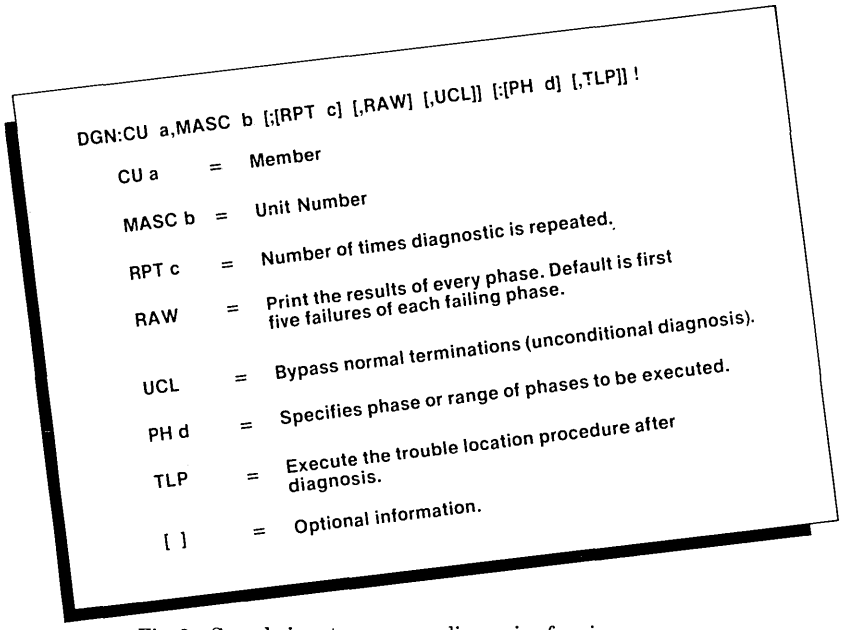


Fig. 3—Sample input message—diagnosis of main memory.

teractive features such as stepping, pausing, and looping are provided for facilitating difficult repairs. Units can be restored to service automatically if they pass all tests. Several host computer versions are supported along with application-dependent interfaces.

Diagnostics can be initiated either manually or automatically. Manual requests can be entered from either a local maintenance terminal or through a work-station terminal associated with a Switching Control Center System (SCCS) connected to the 3B20D Processor via a synchronous data link. Automatic requests originate from other software modules such as the error-interrupt handler, the routine exercise scheduler, or an application-software module.

VI. DIAGNOSTIC TEST DESIGN

6.1 General

The stringent availability requirements of Bell System applications using the 3B20D Processors had a significant impact on all aspects of system design. Diagnostic and maintenance engineers were actively involved in meeting these goals commencing with the initial architectural planning and requirements generation. Many hardware features are provided to monitor system integrity, to detect errors, to reconfigure the system, and to facilitate repair of the faulty equipment.[4,6] Although some of the features are for fault isolation during pack repairs, most are used at the system level to effect repair through pack

replacement. Diagnostics, the primary repair capability for the system, make extensive use of these hardware features for control and observation of the circuitry.

6.1.1 Circuit pack tests

The initial factory testing of circuit packs uses test vectors, which can be applied at terminals of the pack connector in a commercially available computer-controlled test set. Most of the vectors are generated independently from system-level diagnostic tests. The packs are given additional tests in a 3B20D system test bed using diagnostics and some operational sequences.

6.1.2 System-level tests

All 3B20D Processor diagnostics run under the DMERT operating system as user-level processes. To communicate with the unit being tested, the user-level process passes the test scenarios to a kernel process that interfaces to the hardware. Each of these kernel process drivers runs at its standard system priority level to perform the tests. If some time-critical tests are necessary, the priority level can be elevated to avoid interruption by other system processes.

Each of the diagnostic programs is structured to avoid any negative impact on the normal system functions. Special driver functions allow the drivers to handle error conditions generated by the diagnostic tests, thereby avoiding the normal error-handling routines. Since many fault conditions result in system errors, this capability is especially vital to allow thorough testing in the operating system environment.

In the Control Unit (CU) diagnostic, additional safeguards are implemented to assure proper handling of the system recovery and integrity hardware. Even with faults in the off-line CU integrity circuits, the system will maintain normal functionality during CU diagnostics.

The system diagnostics are organized on a unit basis, for example the Control Unit (CU), I/O processor (IOP) or Disk Controller (DFC). Each unit diagnostic is structured into test phases that pertain to a particular subunit. The phases are organized in a hierarchical fashion beginning with the more elemental operations and applying to the hard core of a subunit. Subsequent phases expand in complexity and in the totality of the circuitry exercised.

The user-level diagnostic processes, namely the control program and the test data tables, contain all of the information necessary for control and sequencing of tests. The control program has the interpretative routines for decoding each data table test statement. The program also can use various system configuration parameters, test results, and data table decision functions to modify program flow or to terminate the diagnostic.

6.1.3 Maintenance channel access

As shown in Fig. 1c, the primary interconnection between the Control Units is the maintenance channel (MCH). This circuit is an enhanced version of the dual-serial channel, with special capabilities aimed at maintenance access and control of the off-line CU. It provides the ability to run, stop, load, clear, and step the CU. The MCH allows the active CU to read some off-line CU registers directly and others indirectly using microcoded sequences. Diagnostic-test programs can be loaded through the MCH to the off-line microstore or mainstore. The MCH is controlled by a DMERT kernel process driver that carries out the diagnostic test sequences.

6.2 Control unit diagnostics

The Control Unit (CU) has seven types of subunits: Central Control (CC), Main Memory Store (MAS), Store Address Translation (SAT), Direct Memory Access (DMA), I/O Channels (DSCH), Cache (CSU), and Utility Circuit (UC). The latter three are optionally equipped; the UC is normally used for program testing and is not further discussed herein. Some 3B20D Processor applications have special circuits that are part of the CU; diagnostics for them are concatenated to the CU diagnostic. A pictorial view of the CU hierarchy is shown in Fig. 4, which depicts the multiple levels of units as defined in the ECD.

6.2.1 Central control tests

The first CU subunit tested is the CC, which contains the core of the CU. The CC diagnostics in turn test the maintenance channel, microcontrol logic and memory, registers, data-manipulation logic, memory access, timers, interrupts, I/O interface, error-control hard-

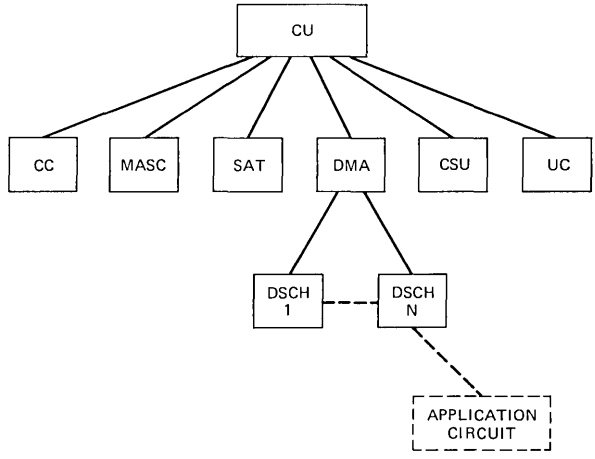


Fig. 4—Unit hierarchy of the control unit.

ware and integrity circuits. The testing uses a series of MCH operations for the basic tests. For the more complex test routines, down-loaded microcoded test programs are executed by the CU under test. The CC is extensively exercised in tests of the remaining subunits.

6.2.2 Memory tests

Initial testing checks out the basic memory-controller operations and the control and data paths from CC to the MAS. Tests are carried out on the error-detection and correction circuits in preparation for using them in array testing. Testing of the memory arrays (up to 16M bytes can be equipped) is with down-loaded microdiagnostic routines. Since the test-pattern programs are executing autonomously in the CU under test, all of its real time is used for testing. Whenever a hardware error is generated, control of the CU passes to a diagnostic error handler. The combination of self error detection and the micro-routines allows extensive pattern checking to be executed rapidly over the complete memory spectrum.

6.2.3 Store address translation tests

Functional tests are performed via MCH on all of the SAT control logic. The memory cells are then tested with various microcoded test patterns. The remainder of the tests, implemented as microdiagnostics, check out the multiplexor, compare logic, matchers, protection logic, and SAT to MAS interface.

6.2.4 Cache tests

The cache is comprised of a high-speed four-way-set associative memory and a 2K by 36-bit interrupt stack. The diagnostic performs extensive tests of the memory cells, matchers, and select logic. In addition to functional tests, a special diagnostic routine called the cache exerciser is used to stress, at high data rates, the cache interfaces to CC, MAS, and SAT. This kind of testing is effective at detecting marginal fault conditions.

6.2.5 Direct memory access tests

The DMA diagnostic checks out the CC-to-DMA communication and control paths, the internal DMA functionality and the DMA operations to MAS. Many of the tests are coded into ROM (Read Only Memory) contained in the DMA. The remainder consist of down-loaded microcoded tests and off-line, main-memory, resident test programs. The final sequence of tests verifies the DMA cache "handshaking" operations. It is noteworthy that in the DMA diagnostics, except for control and down loading through the MCH, all test sequences are executed completely by the CU under test.

6.2.6 Channel tests

The channel diagnostic carries out the remainder of the testing of the CU's I/O capability. Basic tests are performed on the communication and handshaking of the CU to all in-service system peripherals. More exhaustive tests (demand diagnostics) can be specified by the maintenance personnel for troubleshooting more elusive problems. These diagnostic phases require that a peripheral unit be configured as a "helper unit" (specified in the diagnostic input message) to allow the CU to carry out peripheral operations at a high rate.

6.3 Peripheral unit diagnostics

The Disk File Controller (DFC) and Input Output Processor (IOP) are described in Refs. 7 and 8. The DFC can control up to eight Moving Head Disks (MHD) of various capacities, types and manufacturers. The Peripheral Controllers (PC), which are under control of the IOP, are special-purpose I/O units described in Ref. 8. The testing for the DFC and IOP, which share a common front end, is primarily carried out under control of the on-line CU. Since both of these are intelligent controllers, many of the specific tests can be executed autonomously. The peripheral diagnostics utilize the DMERT kernel process drivers to interface to the hardware. Throughout these diagnostics, extensive use is made of driver-maintenance orders and special handling of error conditions.

6.3.1 IOP and DFC tests

The peripheral diagnostics use common-control programs IODIAG and DFDIAG that contain all the CU resident tests and control routines. Separate sets of data table and down-loaded microcode files are used for each unit diagnostic. The overall sequence of testing proceeds from CU/controller interface to complex internal controller operations. Most of the latter make use of the operational firmware in the controller to carry out the test sequences. The more complicated controller tests are part of the resident diagnostic firmware and are initiated by special-driver operations. At the successful conclusion of DFC or IOPC testing, the unit is restored to service to allow testing to proceed on MHD or PC circuits.

6.3.2 Moving head disk tests

Relatively limited maintenance capabilities are provided in the MHD itself. Most of the testing is carried out by the firmware routines in the DFC. To provide an overall check on MHD performance, one cylinder is devoted to diagnostic testing of each read/write head. The error-detection/correction capabilities can also be checked using this

area. As each MHD is tested successfully it can be updated from its mate copy and restored to service.

6.3.3 Peripheral-controller tests

Each controller is microprocessor controlled and can carry out most of its diagnostics autonomously. Some of the tests are firmware resident in the PC's. The remainder of the diagnostic routines are downloaded into the PCs' RAM (Random Access Memory). Some types of PCs also can exercise the units they control, for example a tape transport, and report back the results for use by the maintenance personnel.

VII. TROUBLE LOCATING PROCESS

If the diagnostic request specifies the TLP option, the TLP process is invoked at the completion of diagnostic testing. The process compares characteristics of the failures with a resident data base of fault signatures. In each data table, the designers have partitioned the tests into groups. Any test failure in a group will set a flag bit, called a key, which is permanently assigned to the group. The TLP search, based on the phase and key information, results in a rank-ordered list of closest signatures and, ultimately, into an ordered list of suspected faulty equipment. This approach makes the data base and process less sensitive than earlier methods to circuit or test changes and to marginal failures. The data base (TLDB) is generated off-line from the results of physically inserting faults into units in a test laboratory. Test engineers also can modify the TLDB directly by inserting information into the test data tables. Fig. 5 depicts a typical diagnostic output message from a faulty memory unit.

VIII. EVALUATION

Although many diagnostic tests were generated with the aid of hardware logic simulators, many tests were developed manually. To assure that the diagnostics met the objective—at least 90 percent of the simulated faults detected—an extensive evaluation process was carried out. Using physical fault insertion at the DIP (Dual In-Line Package) terminals, many thousands of faults were inserted. This approach has provided timely and effective design feedback for diagnostic test and TLDB development.

IX. CONCLUSION

In addition to providing a variety of test-control options, the 3B20D diagnostics were designed for multiple execution environments. As a result, the diagnostics have been useful throughout the development

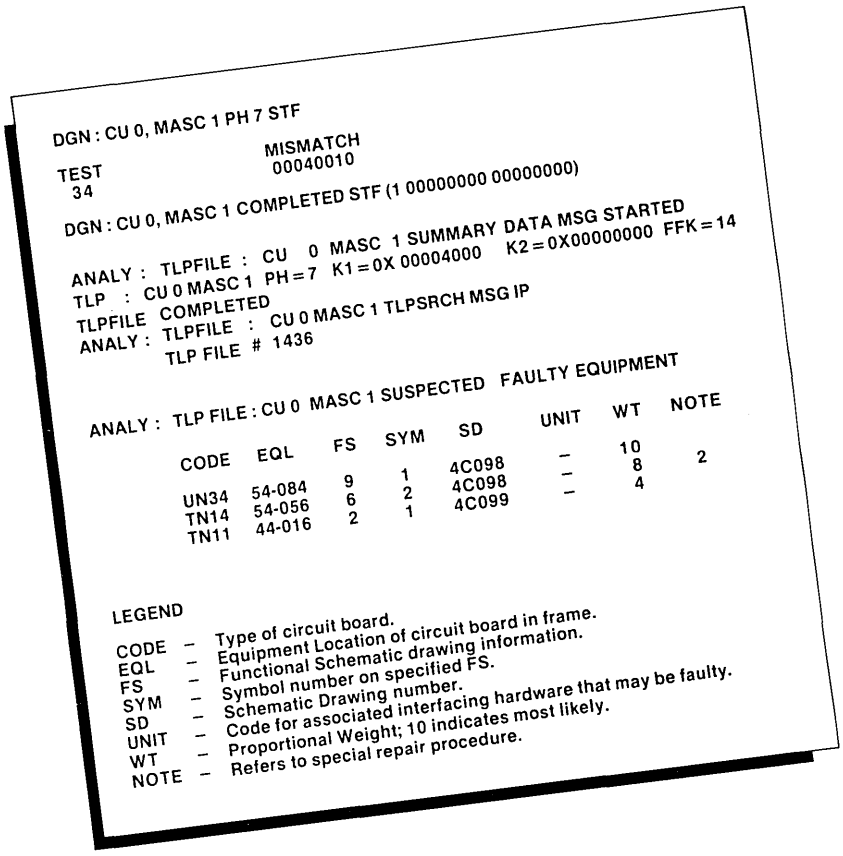


Fig. 5—Sample output message—diagnosis of memory with TLP option.

cycle and have supported the design laboratory, factory testing, installation, normal system operation, application interfaces, and field support. These diagnostics are the major tool for validating the 3B20D Processor hardware and for isolating any faults. The provision of a high degree of hardware self-checking, standby and active redundancy, self-diagnosis, microdiagnostics, and remote testing capability all have contributed to making the 3B20D Processor a high-availability real-time system. Coupled with the DMERT operating system, with its robust complement of features, the 3B20D Processor meets the needs of a wide variety of Bell System projects.

X. ACKNOWLEDGMENTS

The authors would like to acknowledge the many designers and testers at Bell Laboratories and Western Electric whose combined efforts resulted in the diagnostic package described herein.

REFERENCES

1. P. W. Bowman, M. R. Dubman, F. M. Goetz, R. F. Kranzman, E. H. Stredde, and R. J. Watters, "1A Processor: Maintenance Software," B.S.T.J., *56*, No. 2 (February 1977), pp. 255–88.
2. J. R. Kane, R. E. Anderson, and P. S. McCabe, "The 3B20D Processor & DMERT Operating System: Overview, Architecture, and Performance of DMERT," B.S.T.J., this issue.
3. M. E. Grzelakowski, J. H. Campbell, and M. R. Dubman, "The 3B20D Processor & DMERT Operating System: DMERT Operating System," B.S.T.J., this issue.
4. R. C. Hansen, R. W. Peterson, and N. O. Whittington, "The 3B20D Processor & DMERT Operating System: Fault Detection and Recovery," B.S.T.J., this issue.
5. M. E. Barton and D. A. Schmitt, "The 3B20D Processor & DMERT Operating System: Craft Interface," B.S.T.J., this issue.
6. M. W. Rolund, J. T. Beckett, and D. A. Harms, "The 3B20D Processor & DMERT Operating System: Central Processing Unit," B.S.T.J., this issue.
7. R. E. Haglund and L. D. Peterson, "The 3B20D Processor & DMERT Operating System: 3B20D File Memory Systems," B.S.T.J., this issue.
8. A. H. Budlong and F. W. Wendland, "The 3B20D Processor & DMERT Operating System: Input/Output System," B.S.T.J., this issue.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

3B20D Craft Interface

By M. E. BARTON and D. A. SCHMITT

(Manuscript received March 10, 1982)

The 3B20D craft interface package includes hardware, firmware, and software that enables telephone company craftspeople to obtain the status of and exert control over the system. Because this package consists of one or more standard keyboard-display terminals for human-machine interactions, it is flexible and can be adapted to a broad variety of applications. Furthermore, the use of standard terminals and data link protocols allows for inexpensive remote access with capabilities similar to local access capabilities. Finally, the use of video displays has made it possible to provide easy-to-use menus that guide the craftspeople through some of the complex control operations. This article describes the 3B20D craft interface capabilities and the internal architecture of the package.

I. INTRODUCTION

The "craft interface" is that part of the 3B20D Processor that enables people to obtain status information and exert control over the system. To those not involved in telephony, the word "craft" may seem odd. It has traditionally been used to refer to the people who work in and around telephone switching offices performing various maintenance functions on the equipment. In this article, the term is used somewhat liberally to mean any person who interacts with the 3B20D to perform administrative and maintenance functions.

The 3B20D's craft interface is a marked departure from previous systems developed at Bell Laboratories because it relies almost exclusively on video displays and keyboard controls instead of the key-lamp panels and teletypewriters usually found in the Master Control Center (MCC) of electronic switching systems. Status information is presented

visually as graphical displays and text messages on various terminals and printers. There is also a capability to provide audible status by connecting the 3B20D to an audible alarm circuit. System control is exerted primarily via a keyboard attached to the video display terminal, although the 3B20D also includes a separate power control panel for each major hardware unit.

Another important enhancement lies in the ability to access and control all aspects of the system from remote locations such as Switching Control Centers (SCCs). In the past, remote access was obtained by "piggy-backing" data links onto the typewriter terminals in the telephone office and by connecting a telemetry unit to the key-lamp control panel. The 3B20D has introduced a more "intelligent" data link using the CCITT X.25 communication protocol. This link can carry considerably more information and is less vulnerable to noise and other data communication failures. Furthermore, the use of the international standard message protocol (X.25) will standardize remote access to the 3B20D via packet switching networks.

This article first provides an overview of the 3B20D craft interface, primarily concentrating on how the system appears to the craftspeople. Then the internal architecture is described and the various 3B20D applications usages of the general facilities provided in the common system are explained.

II. OVERVIEW

This section describes the 3B20D craft interface as it appears to the people who use it to administer and maintain the system.

2.1 Hardware

The most frequently used parts of the craft interface are shown mounted in two equipment frames in Fig. 1. The left frame contains a "read-only printer" or ROP* on which all important status messages are logged. The right frame contains a keyboard-display terminal that is commonly referred to as the "maintenance CRT," or MCRT. Telephone switching applications of the 3B20D can choose either a frame-mounted or desk-mounted arrangement for the ROP and MCRT. A desk mounted version is shown in Fig. 1.

2.2 Text messages

One way in which the 3B20D communicates with the craftspeople is via text messages. For example, when the message

* From the viewpoint of a programmer, it is a "write-only printer," since the programmer can only send (i.e., write) messages to it. However, the craftsperson cannot type on this device, and so from that viewpoint it is a "read-only printer."

```
DGN:CU 0;UCL!
```

is typed, the central processing unit 0 (CU 0) is diagnosed. The UCL keyword indicates that the diagnosis is "unconditional," which means that all tests will be run even if some of the early tests fail. When the diagnosis is complete, the CU diagnostic prints a text message such as:

```
DGN CU 0 COMPLETED ATP
```

This means that the diagnosis has been completed and all tests passed (ATP). For initial 3B20D applications, the text messages conform to the Bell System craft interface syntax, commonly known as the Program Documentation Standard (PDS) Language. However, all new switching systems developments will be adopting a craft interface language sanctioned by the International Telegraph and Telephone Consultative Committee (CCITT) under the name MML. Since PDS and MML are similar, and since the 3B20D is expected to enjoy broad use in international applications, the operating system was designed so that each application can easily choose the appropriate syntax.

Text messages are typed on the MCRT keyboard, and the response messages are displayed on the MCRT video display and/or printed on the ROP. The basic repertoire of messages available with the 3B20D covers a broad range of maintenance and administration activities. Each application can easily add its own messages to this repertoire.

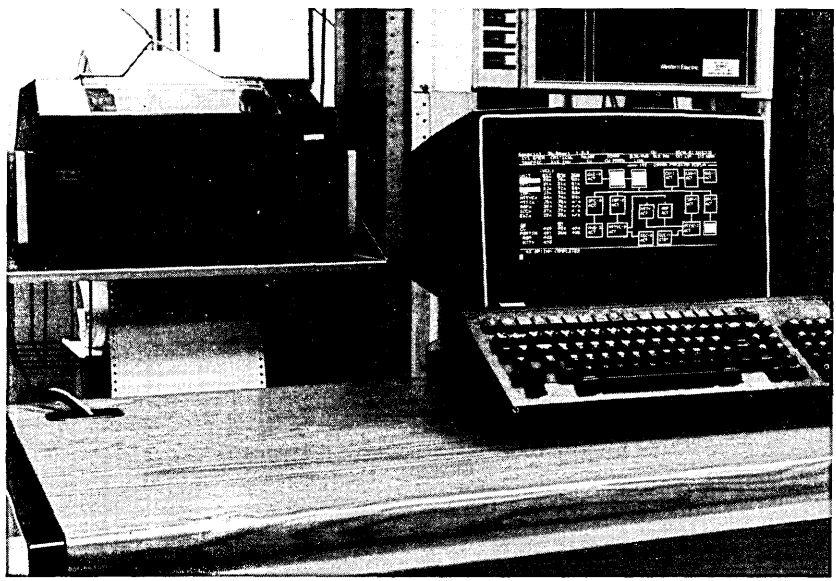


Fig. 1—Craft interface printer and terminal.

2.3 Control and display functions

As mentioned earlier, previous systems used key-lamp panels to display system status and to receive control signals from the craftspeople. The 3B20D uses the MCRT video screen and keyboard, as shown in Fig. 2, in place of such a panel. The upper part of the screen always contains a summary of important system indicators, including CRITICAL, MAJOR, and MINOR severity alarms and "type" alarms, such as CU and BLDG/PWR, which is the indicator for building power. The middle part can display a variety of "pages" that show system status in a graphical form. Finally, the lower part of the screen is used for text input and output.

The standard 3B20D software includes several display pages related to the common processor equipment, and each application can easily add its own pages. The "Common Processor Display Page" shown in Fig. 3 provides a diagram of the redundant components in the basic processor complex. At the left of the diagram is a "menu" listing the control operations that can be invoked when this page is displayed. To select a menu item, the craftsperson depresses the CMD/MSG key, which switches the craft interface from text message mode to command

Fig. 2—Craft interface video screen and keyboard.

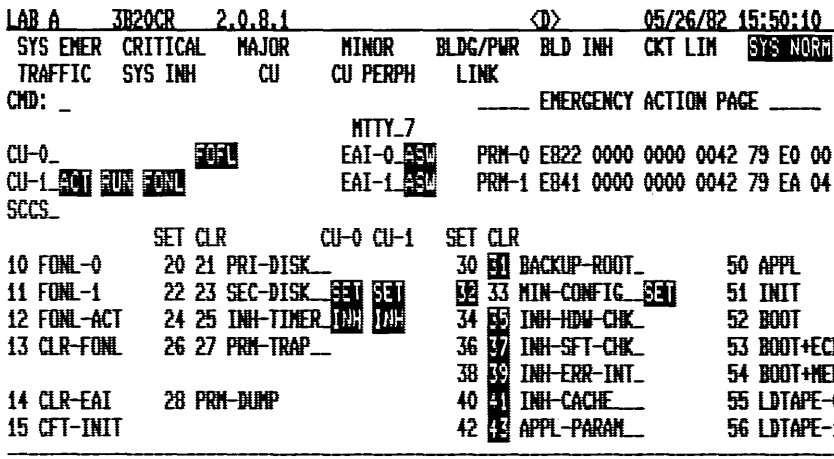


Fig. 3—Emergency action interface display page.

mode. Then the craftsperson types the menu item number, replacing "x" with a 0 or 1 where necessary. Finally, depression of the RETURN key (or the ! key) causes the command to be executed.

The craft interface stays in the command mode until the CMD/MSG key is depressed again. This key is one of four "special function keys." The ALM RLS key is used to retire audible and visual alarms. The EAI DISP key places the craft interface in the emergency action mode, which is described below. When in EAI mode, the NORM DISP key returns the screen to its previous display.

The Emergency Action Interface Page is different from other pages because it is directly controlled by a microprocessor in the MTTY controller (MTTYC) and, therefore, can be used even when the 3B20D software is not operating. As shown in Fig. 4, this page contains menu items that enable the craftsperson to re-initialize the system or to force the redundant units into a particular configuration. Typically, this page is used only when system sanity is suspect.

2.4 Remote access

All capabilities of the craft interface except the power control panels can be accessed from a remote maintenance center via a dedicated data link that is attached to the MTTYC. The standard arrangement includes a primary and a backup link, both of which use the CCITT X.25 communication protocol. The remote site is usually a Switching Control Center (SCC) that contains a collection of computers and terminals that interface with these X.25 links and provide the SCC craftspeople with sophisticated analysis and maintenance tools.

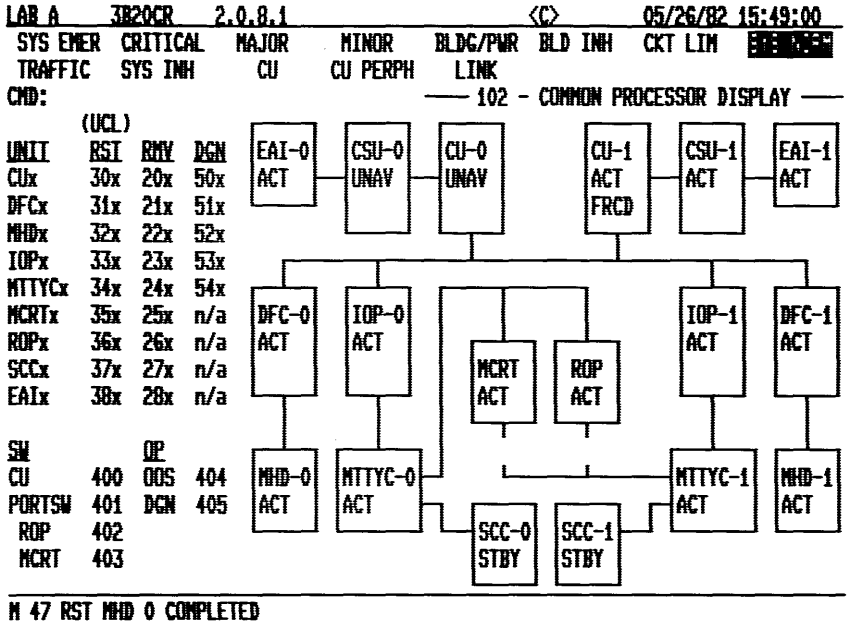


Fig. 4—Common processor display page.

III. CRAFT INTERFACE ARCHITECTURE

This section discusses the hardware and software architecture of the 3B20D craft interface. Figure 5 shows the arrangement of hardware units pertinent to the craft interface, while Fig. 6 shows the software modules. The discussion of the hardware architecture that follows will cover the I/O Processor (IOP) driver and MTTYC handler software, as they are the fundamental parts of DMERT required to access the hardware.

3.1 Hardware architecture

Referring to Fig. 5, one sees that each of the duplex processors is connected to both IOPs, and that each IOP supports up to sixteen peripheral controllers (PCs). Various PCs exist for terminals and printers, data links, tape units, etc. The IOP driver process, which is the part of DMERT responsible for communication with the IOPs, contains "handlers" that deal with the specialized functions of the PCs. The following handlers are pertinent to the craft interface: craft interface handler, X.25 handler, emergency action interface handler, general-purpose terminal handler, and alarm handler.

3.1.1 Craft interface handler

The MCRT, ROP, and X.25 links are attached to a PC known as the maintenance teletype controller, or MTTYC. The craft interface

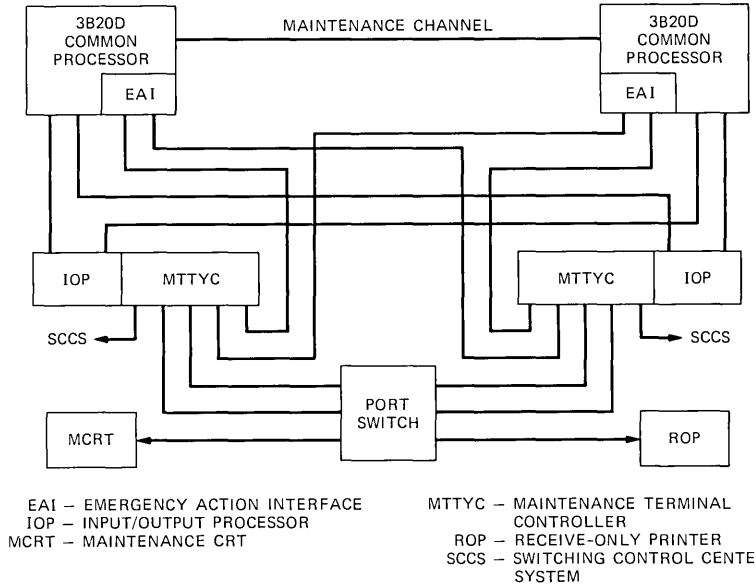


Fig. 5—Craft interface hardware overview.

handler controls the transfer of data to and from the peripheral devices associated with the MTTYC. The MCRT and ROP are administered directly by the MTTYC, while the X.25 links require the additional services of the X.25 handler described later. For each device or data link attached to the MTTYC, the handler supports all standard access operations of the *UNIX** operating system. In addition, this handler treats the single MCRT terminal as two "virtual terminals," with the upper part of the screen used for control/display functions and the lower part used for text messages as shown in Fig. 2. Each virtual terminal appears to the higher-level software as a separate device.

3.1.2 X.25 handler

The X.25 handler provides communication with a remote maintenance center via 1200 to 9600 bits per second synchronous data links using levels 2 and 3 of the CCITT X.25 protocol. Level 2, referred to as the link layer, provides link initialization, error control, and flow control on the physical data link and is implemented as firmware within the MTTYC. Level 3, referred to as the packet layer, multiplexes several independent data streams (logical channels) on the physical link and is implemented by the X.25 handler software. In effect, the X.25 handler makes the single MTTYC look like a multitude of independent communication channels.

* Trademark of Bell Laboratories.

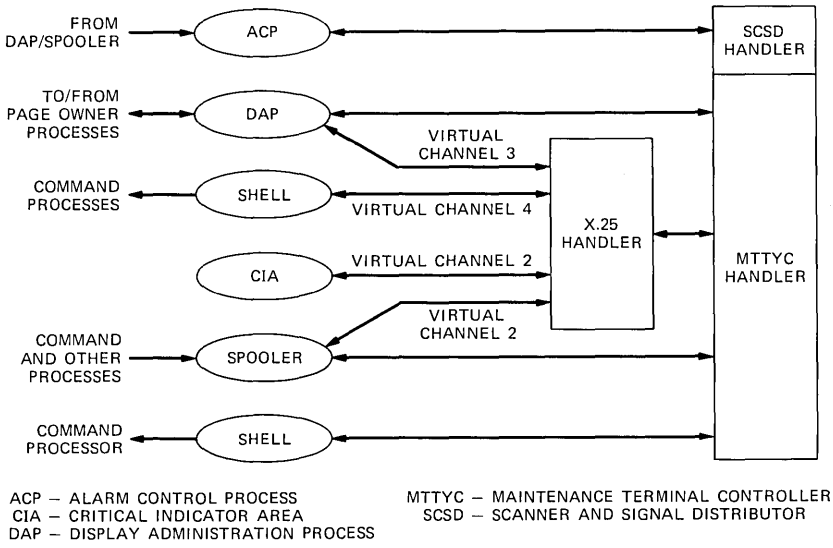


Fig. 6—Craft interface software architecture.

3.1.3 Emergency action interface handler

The Emergency Action Interface (EAI) is a processor circuit pack that provides basic status information and manual control even under extreme circumstances. That is, the EAI circuit gives craftspeople limited access to the 3B20D regardless of DMERT software sanity. This access is in the form of the EAI page display shown in Fig. 4, which is controlled totally by the firmware in the MTTYC. The MTTYC interacts with the EAI circuit via the connection shown in Fig. 5 to acquire the status information for display on the MCRT. Also, when the craftsperson selects a menu item from the EAI display, the MCRT delivers the corresponding commands to the EAI circuit.

The emergency action interface handler only comes into play when DMERT is operating sanely. It has two major functions. First, it periodically "punches in" with the EAI circuit to indicate that the software is operating correctly. If the EAI (and, subsequently, the MTTYC) fails to receive this periodic signal, it will automatically initiate a system recovery operation to restore software sanity. Second, the EAI handler receives some non-emergency commands from the MTTYC via the EAI, including non-emergency initialization and reconfiguration signals.

3.1.4 General-purpose terminal handler

The craft interface subsystem supports terminals other than the MCRT and ROP via the teletype controller, or TTYC. This controller

offers all MCRT and ROP functions except access to the EAI circuit. Its handler is similar to the craft interface handler used with the MTTYC and, in particular, offers the dual virtual terminal mode of operation needed to intermix control/display functions and text functions on a single terminal. This capability is typically used for dial-up monitoring of a system from a Western Electric or Bell Laboratories product support center.

3.1.5 Alarm handler

The scanner and signal distributor (SCSD) peripheral controller provides sense and control points that are tied into the system power controls and alarms. The SCSD handler detects sense point state changes and sends commands to change the states of control points. Higher-level software uses these capabilities to detect situations such as power removal, fuse operation, or thermal warnings and to respond by activating audible alarms or power shutdown circuits. The application can also tie into the SCSD and configure the higher-level software to detect and react to such things as building intrusion alarms.

3.2 Software architecture

Figure 6 shows the modules that comprise the standard DMERT craft interface software subsystem. Already discussed were the device handlers that connect to the modules on the right of the figure. Each application usually adds its own modules that tie into the interfaces on the left. Also, many other parts of DMERT (e.g., the diagnostic subsystem) connect to the craft subsystem via these front-end interfaces. The modules in the middle of the figure are the "workhorses" of the craft subsystem and provide the internal interfaces used by DMERT programmers to interact with the craft personnel.

3.2.1 Text input processing (shell)

The Shell is the module that interfaces between the handlers and the processes that respond to text input commands. The term "shell" is borrowed from the *UNIX* operating system, and DMERT's craft shell operates in a manner similar to the other shell. That is, the DMERT shell reads an input line, parses it into a command verb and a list of "tokens," searches for the command process in the appropriate disk directory, creates the command process, and passes the list of tokens to it. The command process has access to a "shell library" that includes functions to do further parsing of the tokens.

The major difference between the DMERT shell and the other shell is that DMERT must parse commands that are typed in either the PDS or MML syntax, where as the *UNIX* operating system shell uses

a more general syntax for a broader variety of applications.* Another difference is that the PDS and MML languages include the notion of a "locking acknowledgment." That is, an input command process is required to give a two-character response (e.g., OK if the command is successful, PF if a printout follows) within a few seconds after the person types the message, and no other command can be typed until the acknowledgment appears. In the *UNIX* operating system, message acknowledgments are not required and command type-ahead is allowed. Therefore, the DMERT shell library includes functions that pass the acknowledgment back to the handler in order to unlock the terminal.

Referring again to Fig. 6, note that each text input channel has its own instance of the DMERT shell. This allows each channel to operate independently of the others, which means that several craftspeople can simultaneously interact with the system.

3.2.2 Text output processing (spooler)

The DMERT output spooler accepts text strings from higher-level processes and directs them to the appropriate output devices. [The term "spooler" is a computer science anachronism that comes from the days when information waiting to be printed was temporarily stored on reels (spools) of magnetic tape.] One might ask why the higher-level process doesn't write directly to the device (via the device's handler, of course). There are two reasons to avoid direct writing.

First, the PDS and MML languages require that each message be enclosed in an "envelope" that clearly delineates the message. This envelope generally contains a time stamp, a message priority/alarm indicator, and an end-of-message delimiter. The time stamp can be as simple as the number of minutes past the current hour, or it can be the complete date and time. The priority/alarm indicator shows whether the message is a result of a manual or an automatic action and whether the action being reported requires immediate attention. Finally, the end-of-message delimiter provides for automatic logging and browsing of messages by a computer in the remote maintenance center. Centralizing the generation of the output message envelope in the spooler simplifies the work that higher-level processes must do to produce text output. Also, changes or additions to the envelope can be introduced easily by modifying only the spooler.

The second reason for using the spooler approach is that many messages must be sent to several places. For example, the usual mode

* For DMERT applications that require the more general *UNIX* shell on terminals other than the MCRT, it is possible to configure the system in such a way that the *UNIX* shell is automatically activated on some or all general-purpose terminals. In other words, both the DMERT and the *UNIX* operating system shells are compatible with the general-purpose terminal handler described earlier.

of operation when a remote maintenance center is attached via the X.25 links is to send every output message to both the MCRT, ROP, and the remote center. However, if the ROP runs out of paper or if there are no craft personnel near the 3B20D, messages can be routed only to the remote center. To handle these and the diverse message routing situations that can arise, the spooler maintains a "map" showing where messages are to be routed based on their priority/alarm indicator and on a message type code that is received from the process that generated the message. The map also can be configured to route some message types to disk files instead of or in addition to printing them. This feature is useful for keeping a log of messages that are sometimes needed for problem analysis but that would overload the ROP or X.25 links if sent routinely. Input commands are provided to print the contents of these logging files when needed.

3.2.3 Control/display processing

The Display Administration Process (DAP) administers the upper part of the MCRT (and, possibly, other video terminals) containing the displays that replace the traditional key/lamp panels for 3B20D applications. DAP's fundamental purpose is to display "pages" from its repertoire and to accept commands listed on "menus" associated with the pages. Figure 3 shows the Common Processor Display Page, which is one of the standard pages delivered with DMERT. Typically, the majority of display pages are defined by the specific application processes.

For each page, there is a Page Description File (PDF) containing a pseudo-program that describes how the page should be "painted" on the video screen and what menu selections are allowed. PDFs are constructed like programs and compiled by a page description file generator (PDFGEN) program.

3.2.3.1 Display functions. When DAP begins execution, no pages are active. Then, as the various parts of DMERT and the application are initialized, they send interprocess messages to DAP requesting that specific PDFs be loaded into main memory and activated as display pages. A maximum of 64 pages can be active at any one time. Other interprocess messages tell DAP which of the active pages to display on each video terminal.

Each page consists of up to 128 graphical constructs known as "indicators," a term reminiscent of the lighted indicators on a key/lamp panel. The process that initially informs DAP to activate a page becomes the owner of that page and it and other processes can subsequently inform DAP (via interprocess messages) to change the states of the indicators. For example, one popular type of indicator is the rectangle enclosing a few text items, such as the CU-0 box in Fig.

3. The CPDP page owner can send messages to DAP causing the phrase UNAV to change to OOS when control unit 0 (CU-0) changes from the unavailable state to the out-of-service state.

These state changes can be communicated in detail, for example, by sending a message to DAP specifying the characters OOS to replace UNAV. However, the usual method is to use state numbers instead of the actual characters. DAP has access to a table of 256 state entries specifying the standard text and video attributes associated with each of the 256 possible indicator states. The standard maintenance states, such as active, standby, and out-of-service, have predefined state numbers, and each application can define additional states for its own needs. The advantage of using state numbers is that the text and video attributes for each state can be centrally controlled. For example, one application could use the text ACT for the active state while another application used ACTIVE, and the only difference would be in the state table definition entries.

Video attributes were mentioned above in addition to the text that can be associated with an indicator. For the usual black-and-white terminals, DAP recognizes the "blink" attribute and the "reverse" attribute. The conventional use for the reverse attribute is to show that an indicator is, in some sense, active. In other words, a reversed indicator is similar to a lit lamp on a key/lamp panel. The blink attribute is used to draw attention to a situation that requires immediate action, just like a flashing lamp. In Fig. 3, the SYS NORM indicator is reversed to show that the system is operating normally. If a major alarm occurs, the MAJOR indicator will blink until the ALM RLS key is pressed.

DAP also includes the capability to deal with color terminals, which have a much richer set of attributes. For example, the MAJOR indicator could be displayed as white characters against a red background, while the MINOR indicator would be white against yellow. It is possible to define indicator states in the most general way for color terminals and then have the Equipment Configuration Database contain MTTYC or TTYC options that "downgrade" the color states for black-and-white terminals. In the example mentioned above, both the red and yellow backgrounds would be mapped into the reverse attribute.

3.2.3.2 Control functions. In addition to displaying pages on the MCRT screen, DAP also can receive menu commands typed on the MCRT keyboard. These commands usually are referred to as "pokes" since they are similar to the action of poking a key on a key/lamp panel. As mentioned earlier, depression of the CMD/MSG key switches the terminal from the message mode to the command mode and vice-versa. When in the command mode, DAP displays a CMD: prompter towards the top of the screen, as shown in Fig. 3, and positions the cursor so that the typed characters appear in that line.

A command line consists of a number (usually 3 or 4 digits long) that optionally can be followed by some text characters. The craft interface handler, knowing that the terminal is in command mode, routes this input to DAP instead of to the SHELL. DAP examines the number to determine if it corresponds to a local or a global menu item. Local menu items are associated with the page(s) currently being displayed on the video screen. Global items are associated with any active page, even if the page is not currently displayed. In other words, a globally defined item will always be accepted and acted upon, even if its page is not being displayed.

If DAP successfully locates the menu item corresponding to the number that was typed, it usually sends an interprocess message to the owner of the page defining that menu item. This message contains the item number and the additional text characters typed, if any, as well as the originating terminal identification. The owner then takes whatever action is appropriate.

We used the word "usually" above because in some cases the response to a command is some simple action such as flipping to a new page on the display. In other cases, the PDF can specify a function to be executed by DAP upon receipt of the command, thereby bypassing the overhead of interprocess messages. One interesting aspect of this feature is the ability for DAP to translate a menu command into a text message to be passed to an instance of the SHELL, with the additional characters substituted in the message. This makes it possible for an application to design easy-to-use menus as an alternative to text message input, but to handle all terminal inputs internally as if they came through the SHELL.

A final note on commands has to do with locking acknowledgments. As with the SHELL, DAP requires a positive response to each command before another command can be accepted. For a command passed to a page owner via an interprocess message, the owner must send an acknowledgment message back to DAP within a certain time period or be abandoned. For commands handled via the function call approach, the function returns an acknowledgment code to DAP.

3.2.4 Forms processing

As described above, DAP is typically used for control/display functions related to maintenance activities such as configuration control. However, some applications require more general display functions than the indicator/menu approach appropriate for these maintenance scenarios. The craft subsystem provides facilities for entering textual information as part of displays.

A DAP page can be defined to contain input areas other than the standard command input line. When such a page is displayed, depression of the CMD/MSG key places the cursor at the first input area

instead of at the command line. The craftsperson can enter text into this area and/or move the cursor, using the terminal cursor control keys, to the next input area on the page. When the RETURN key is hit, DAP passes the typed information to the page owner via an interprocess message.

3.2.5 Alarm processing

The Alarm Control Process (ACP) is the part of the craft subsystem responsible for sounding audible alarms and displaying a summary of current system status at the top of the video screen. ACP is created during DMERT initialization and notifies DAP to activate the page known as the System Summary Area (SSA). It also attaches itself to the SCSD handler to gain access to the signal distributor points used to sound audible alarms. The plant measurements data base is automatically updated for severity-type alarm counts.

As the spooler and DAP receive messages from the higher-level processes, they check for situations requiring audible alarms and/or changes in the system status summary. For the spooler, alarm information is contained in the message prefix received from the higher-level process. For DAP, this information is derived from the indicator state data. Both cases result in messages being sent to ACP, which then operates the signal distributor points via the SCSD handler and/or sends DAP messages to change the states of indicators on the SSA page. Application processes also send messages directly to ACP for alarms.

Another message received by ACP from DAP is a notification that the ALM RLS key has been depressed. This causes ACP to reset the signal distributor point controlling the audible alarms. This key depression is also reported from the MTTYPC directly to the SCSD audible alarm retire scan point.

The Critical Indicator Area (CIA) process is closely related to ACP and DAP. Its function is to extract from the SSA page sixteen "critical indicators" of system status and periodically send them to the remote maintenance center via the X.25 link for alarming and display. The remote center can use this periodic "heartbeat" from the CIA process as one test that the system is operating sanely.

3.2.6 Common processor display page

Thus far we have described the general hardware and software modules that comprise the 3B20D's craft interface subsystem. DMERT also includes several processes that are, in effect, users of the craft subsystem. One of these is the process that owns the Common Processor Display (CPD) page that we have frequently used for examples (see Fig. 3). This Real-Time Status (RTS) process is created

as part of the DMERT initialization sequence and immediately sends messages to DAP to activate the CPD page. RTS also attaches itself to the Equipment Configuration Data Base (ECD), which it periodically examines to determine if any units shown on the CPD page have changed state. Spontaneous equipment configuration changes are reported to RTS through a library interface (CONFIG) from the various device handlers. In either case, appropriate messages are sent to DAP.

IV. SUMMARY

The 3B20D/DMERT system has taken a significant departure from earlier switching processors in many areas, but perhaps none is so visible as the craft interface. The use of flexible video displays makes it possible to adapt the 3B20D to diverse applications quickly and economically. Also, the introduction of a reliable, secure, high-capacity data link for remote maintenance makes the 3B20D/DMERT system well suited for unattended operation, with resultant cost savings.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating Systems:

Integration and System Test

By W. F. KLINKSIEK and H. L. MITCHELL

(Manuscript received March 18, 1982)

This article describes the general approach that was taken in integrating and system testing the 3B20D Processor system. Since both the system hardware and software were developed simultaneously, the goals of the system test and integration plan naturally shifted emphasis and expanded their scope from achieving hardware stability to establishing software functionality and finally to demonstrating system stability. This article also overviews some of the project management techniques and procedures applied during the development of the 3B20D Processor.

I. INTRODUCTION

An important aspect of the development of any complex system such as the 3B20D Processor is the methodical integration and system testing during all phases of the development consistent with the experience gained from previous developments.[1-4] Since the hardware, software, and microcode were designed and developed simultaneously, the initial efforts focused primarily on the hardware and firmware using stand-alone exercise modules and system diagnostics run from a laboratory support processor. After the hardware reached sufficient stability, emphasis turned to functional testing of each major software subsystem and feature. Finally, as full functionality was achieved, the major thrust of testing focused on system integrity and reliability using the previously developed tests as a regression test package to assure no loss in functionality as problems were cleared. The development methodology is summarized in the relative timeline sequence chart shown in Fig. 1.

Also discussed in this article are some of the project management

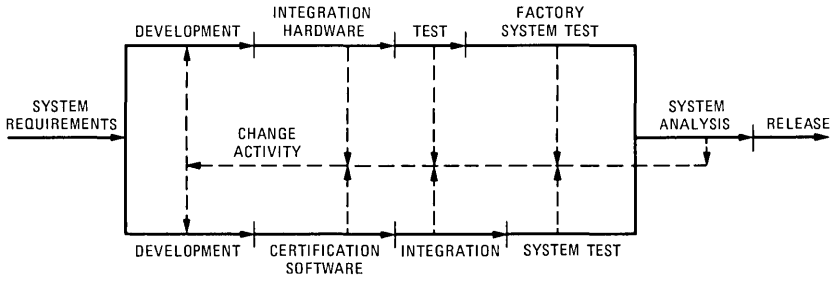


Fig. 1—Generalized development model.

techniques and administration tools used to control the changes and new features introduced into the system.

II. EARLY HARDWARE/SOFTWARE INTEGRATION AND TEST STRATEGY

2.1 Objectives

The objective of the initial integration and test effort on the prototype hardware was to verify basic instruction execution and memory access, establish full diagnostic capability of the hardware,[5] prove in peripheral access and functionality, and establish stable communication interfaces. In achieving these objectives, a stable software development environment was achieved for the major portion of the software development.

2.2 Stand-alone exercise modules

The diagnostics were developed to initially run from the laboratory support processor in conjunction with the hardware development. This simultaneous development of the diagnostics and the processor hardware had the unique advantage of providing individual functional verification of each circuit pack or major unit before integration of the operational system was attempted, thereby saving much laboratory time ferreting out faulty hardware. The initial functional integration started with simplistic CPU test modules that afforded stand-alone verification of basic operation. Upon reaching acceptable functionality, stand-alone test modules were used to establish communication with the disk file controller and moving head disks.

2.2.1 Central processing unit integration

Two test modules were used extensively to integrate the early Central Processing Unit (CPU) hardware, firmware, and subsequent changes. The first test module was designed to test basic main-memory access and instruction execution with output to serial channel on the

Central Control Input/Output (CCIO) bus.[6] Loading this module from the laboratory support processor verified the communication link from the support processor to the 3B20D Processor. In addition, the execution of the module not only verified basic hardware functionality but also verified the data-link capability to a TTY via the serial channel. The second test module, in combination with a primitive version of the operating system, established two processes and cyclically sent messages between them. This capability not only tested more of the hardware features of the CPU, but also provided a means to verify stable operation over long periods of time. This test module was then expanded to verify memory update on the off-line Control Unit (CU) and "soft switch" capability between the duplex units.[7]

2.2.2 File system integration

Once basic operation of the CPU was verified, attention was pointed toward the file-system operation requiring integration of the Direct Memory Access (DMA) unit, the Disk File Controller (DFC) unit, and the Moving Head Disk (MHD).[8] Again a stand-alone test module, based on the disk driver software and the primitive operating system, was used for the integration of the hardware and firmware. Because of the large percentage of the hardware that had to be operational for successful execution of this test module, it became an invaluable tool not only for the integration of the preproduction hardware but also for Western Electric manufacturing, testing, and installation of early models of the 3B20D Processor in application system laboratories.

2.3 System software

Once the hardware was integrated and verified to the limits of the stand-alone test modules, development of the operating system and system-initialization software proceeded rapidly, and the integration effort switched emphasis from strictly hardware to system software. The strategy was to incrementally integrate—from the primitive operating system—each new capability of the operating system and system-initialization software with the hardware until a fully cycling stand-alone basic processor system was achieved.

With the basic capability to initialize the system and cycle the operating system, integration proceeded to verify the 3B20D resident diagnostic control structure and diagnostics.

By this time, additional integration tests were necessary to more fully expand coverage of the system. Thus, a test process was developed that created disk read and write jobs with a variable number of these child processes specifiable up to the number of allowable Dispatch Control Table (DCT) entries. Because of the large percentage of the processor used by this test process and because of the controllable

activity, it became an invaluable regression-test vehicle for subsequent integration activities as well as a system stress test.

2.4 Results

The primary result of this early effort was the establishment of a stable hardware and operating software base for the development of the features.

III. INTEGRATION AND SYSTEM TEST

The 3B20D system-level testing is actually divided into three distinct functional groups consisting of System Integration, System Test, and System Analysis. A brief historical review of the evolution of these groups is perhaps the best way to describe their respective functions. In early 1979 a decision was made to delegate the system testing of DMERT to Western Electric.[9] A Western Electric department was formed with the goal of taking over full responsibility for DMERT system testing by January 1, 1980. This transition actually took place about six months ahead of schedule in July 1979 and the system remains a Western Electric responsibility. The goals of the system testing group at that time were to release laboratory quality prereleases to DMERT applications to allow parallel application software development with the DMERT development. The system testing group also developed an extensive, documented set of tests that could be used not only to test the prereleases but would also serve as a base for testing all generic software releases in the years to follow.

Another group, the System Integration group, was responsible for planning and coordinating the building (compiling) of each DMERT release, getting the release installed and cycling in the 3B20D development labs, and assuring that basic functions worked. Once this was accomplished, responsibility for the detailed testing was turned over to the system testing group. Thus, the system testing group could concentrate more on actual testing and problem resolution and less on bringing up the internal loads.

3.1 Integration

System integration controls the flow of software changes from the time a developer completes a change through the release of that change to a customer. The specific areas of responsibility include:

(*i*) Load engineering and planning
(*ii*) Benchmark tracking and analysis
(*iii*) Integration testing
(*iv*) Release-letter generation
(*v*) Modification Request (MR) tracking and MR data base integrity.

3.1.1 Load engineering and load planning

For each DMERT release, an individual is assigned to be the load engineer. This individual serves as the focal point for all load-building activities. Specifically, the load engineer analyzes all changes planned for the release by the generic engineer, decides in what sequence changes should be taken, oversees the building of the load, and coordinates installation and integration testing of the load in the development labs.

Members of the integration team report to the load engineer who assures that all activities needed to deliver the load on schedule are assigned and completed. The load engineer with assistance from the integration team resolves daily problems and, as necessary, reschedules activities and people.

The load engineer in conjunction with the program administration staff coordinates the actual building of the load. The load engineer must thoroughly understand the mechanics of how the system is built, what software dependencies exist and how source code is controlled via the CMS/M2 system.[10]

3.1.2 Benchmark tracking and analysis

Each new generic feature or major software enhancement results in a set of benchmarks that identify the date at which major activities are scheduled for completion. Benchmarks serve a dual purpose: first, as a management tool for measuring how the project is doing relative to the plan; and second, as a planning aide for other people or groups identifying dependencies for other features, hardware availability, or lab installation.

Several tools have been used for identifying, tracking and reporting on feature benchmarks within the DMERT development organization.

3.1.3 Integration testing

One of the major objectives of the integration team is to assure that the load given to the system testing group is of sufficient quality to allow detailed functional testing. To verify that the system is of such quality, basic functional tests are run to assure that the major subsystems are operational. These include diagnostics, processor duplex operation, disk and I/O capabilities, and Recent Change and Verify.

3.1.4 Release-letter generation

Typically the applications that use the 3B20D Processor want the new DMERT software releases as soon as possible after the completion of system testing. This has presented a unique challenge to DMERT development management: the need to get releases, complete with essential documentation, to a number of different customers within one day of the completion of system test.

One vehicle used to supply necessary timely documentation to the customers is the release letter. This letter has evolved into a rather detailed document covering:

(*i*) Support processor installation procedures
(*ii*) 3B20D installation procedures
(*iii*) List of all file names
(*iv*) List of all changed files
(*v*) List of all required data base changes
(*vi*) MR descriptions for all MRs resolved in the release
(*vii*) MR exceptions list.

Of particular importance is the MR exceptions list. The intent of this list is to communicate to the customers known problems that exist in the release and, when available, action to be taken if it is observed on their machines. This communication mechanism saves many hours that applications personnel would spend analyzing problems already identified by the DMERT organization.

To assure timely distribution of this letter, all sections are put on a support computer and support programs are executed to assemble them into a document that is available on the day of the software release.

3.1.5 Detailed MR tracking and data base integrity

The integration team also was chartered to establish the integrity of MR data base, to produce accurate and timely reports, and to respond promptly to high-priority problems. Weekly audits of the entire data base are performed to assure that MRs do not remain in a transient state for an unreasonable length of time.

3.2 System test

The primary objective of the 3B20D System Test group is to test the DMERT system on the 3B20D Processor in order to validate that all advertised features and capabilities perform according to their documented requirements. System tests are designed to test all the functional capabilities of the processor and its hardware both in no-load and stress environments.

In the two and one half years since its inception, the System Test group has developed a complete system testing package containing over 700 test cases. As new features are developed, test cases are developed and each feature is thoroughly tested. Test cases are documented and in many cases processes are written to automatically execute the tests. Once a feature is released for customer use, a subset

of the defined test cases is included as part of an on-going regression-test package.

A concept of certification testing was established to identify problems early in the development cycle. This allowed more problems to be debugged and fixed before release and resulted in a more stable system testing environment and higher ultimate product quality.

Certification testing requires the developers to build in the official environment[10] and to demonstrate the proper execution of their new code to a system tester before it can be delivered to the integration team. The system tester has an option to request particular tests to be run with the new code and thus certify that the software to be submitted has passed some basic tests and can be approved for further processing. Software not passing certification is rejected and the developers have to correct the deficiencies and schedule a follow-up certification test.

3.3 System analysis effort

The System Analysis Group (SAG) effort was planned as an extension to Integration and System Testing. As its objectives, SAG was to perform tests aimed at measuring the performance and reliability of the 3B20D as a system. A separate development laboratory was constructed with the primary intention of simulating and functioning as a field site. Since this was the only 3B20D laboratory planned to run for long periods of time without rebooting, many problems of a periodic or long-term nature were first observed there.

SAG members approached the stability aspect of the job by first defining measurable metrics. Objectives were defined based on the measured system reliability. The SAG team then identified and investigated problems that impacted system reliability and reported the effects on system stability once the problems were resolved.

Stability data was collected during weekend testing. The tests involved running a controlled-load package containing system exercise processes for specified periods of time, usually several days. These tests were generally run unattended to evaluate hands-off machine performance. All messages to the Read Only Printer (ROP) were stored on disk, dumped at the end of the test and analyzed using a program developed for this purpose.

Three sets of objectives were defined for data analysis: a long-term objective for system reliability; a cut objective that identified satisfactory stability levels for first application at in-service offices; and the objective of establishing concern thresholds. Any data above the concern threshold was clearly unacceptable for even initial in-service machines. Data lying in the area between the cut objective and the

concern thresholds needed additional understanding in order to make a go/no-go decision on cutover.

An example of one of the metrics used to track stability is shown in Fig. 2 for ten releases of DMERT prior to the first machine cutover in September 1981.

IV. FACTORY SYSTEM TEST

Factory System Tests (FST) and Quality Assurance (QA) tests are the final hardware tests run at the Western Electric Company manufacturing plants to assure that a quality hardware product is delivered to the customer.

4.1 Objectives

The objectives of FST and QA are to test the hardware functionality and interconnections of fully assembled systems to assure that the processors as built meet design intent. These extensive tests assure the highest possible quality in the product when shipped to the customer.

4.2 FST test strategy

Instead of developing special test software for the FST, the actual DMERT operating system is enhanced with additional exercise processes to form the Factory/Installation Software Test (FIST) package. The testing is divided into two phases: the normal operation phase and the stressed operation phase. These tests apply to all hardware delivered by the factory including the system as ordered, the comple-

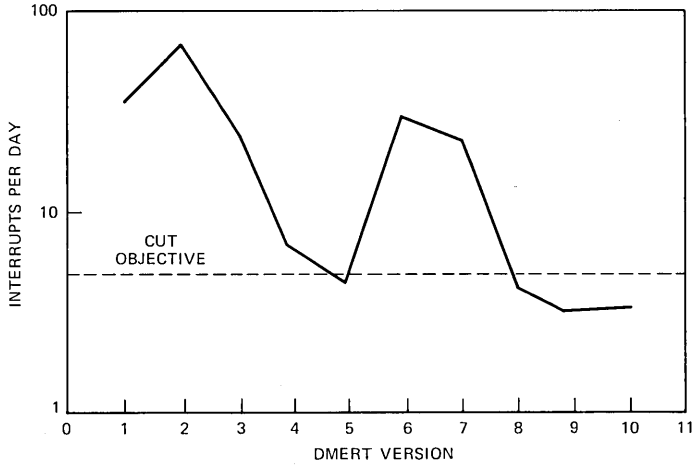


Fig. 2—Interrupt incidence history.

ment of spare circuit packs, growth units and circuit packs, and repaired product.

4.2.1 Normal operational tests

The normal operational tests are designed to verify the functionality, interconnections, and basic maintenance operations associated with each unit under normal operating conditions. Included in these tests are the activation of system initializations under all possible minimum configurations using the power switch and the craft interface terminal. The tests then assure functionality of all units under simulated maintenance conditions by removing and restoring each unit using both the power switch and the craft-interface terminal. During this test the system must remain operational. The next phase of testing requires the running and passing of all diagnostics for each unit within the system. Finally a series of special exercise processes are used to simulate actual operation of the disks, tape units, TTY and other data link controllers, and a CU soft-switch process for duplex capability verification.

4.2.2 Stressed environmental operational tests

The 3B20D Processor is designed to operate under a wide range of temperature and battery conditions. To assure that the system meets the design intent to operate under these conditions, two additional test environments are imposed on the machine before shipment.

4.2.2.1 Low voltage. The power converters are stressed most under conditions of low-input voltage; thus, the system must pass all the tests prescribed above at an input voltage of -43.75 ± 0.05 volts. This voltage is 91 percent of the nominal -48 volts.

4.2.2.2 High temperature. High-temperature operation of the 3B20D Processor is critical to avoid outages during commercial power or mechanical failures that result in the loss of building air-conditioning systems. The system tests prescribed above must pass in a system that has been operating at a stable elevated temperature of $49°C \pm 1°C$ for a period of at least four hours.

4.3 QA testing

In addition to the factory system test on all systems, additional tests are rerun under the auspices of the Bell Laboratories quality assurance organization and the Western Electric quality review organization to assure that statistical quality control limits are not exceeded, thus maintaining a high level of quality for the customers.

4.4 Result

A major milestone was achieved in March 1980, when the first field shipment to the Traffic Service Position System (TSPS) site in San Antonio, Texas, was not only delivered on schedule, but passed the complete battery of factory system tests.

V. ADMINISTRATION

In this section, a brief overview of some of the important aspects of project-management and project-control techniques are presented.

5.1 Change authorization

From the beginning of the project, the hardware design was under very tight controls. All changes or feature additions had to be approved by a management-change committee with representation from Bell Laboratories and Western Electric. This committee provided both a forum to review designs and design changes and to discern the economic impact of each change. This committee then established a joint subcommittee, called the Engineering Support Group, to schedule and track each change from design through manufacture and ultimately to the installation into the various system development laboratories.

Software change control was less tightly controlled during the initial development and relied heavily on the software development supervisors responsible for each subsystem. Once the software was delivered to the application more stringent controls were introduced. At that point, feature content, overall coordination, and generic scheduling are the responsibility of the Generic Engineer and the Project Manager.

5.2 Application interfaces

To assure that the 3B20D Processor system meets the needs of the variety of Bell System applications, a group was established to act as the single focal point for the applications for all feature requests and MRs.

5.2.1 Feature content

To establish feature content of the system, the Application Interface group, in concert with the applications, developed a prioritized list of feature requests and enhancements for the Project Manager and the Generic Engineer to review. Thus, a final list of features and enhancements was established taking into account customer needs, schedules, and resource limitations.

5.2.2 Modification requests

Initially the Application Interface group also acted as a clearinghouse to prioritize, from the users point of view, the problems that

they discovered as the generic matured. This list, in conjunction with internally generated MRs, formed the basis for the Generic Engineer to approve MRs to be fixed for inclusion in the generic. Once an MR was approved, the Load Engineer tracked its progress through development, integration, system test, and release.

Once the first generic was cut into service, a committee was established with representation from applications, DMERT development, generic engineering, system test, load engineering, and field support. This committee's function was to tightly control and adjudicate all software changes so as to assure that field service was not adversely affected and that real service problems were quickly attended to and delivered on a timely basis.

5.3 Project-tracking tools

A finite-state MR control mechanism was put into place to track and record changes in the status of MRs during the development cycle.[10] From this data base, various reports were automatically generated for use by all organizations associated with the project. This central source of project-status information was an essential ingredient to the determination of areas of concern so that action could be taken, as well as a repository of all schedule information relating to MRs. This capability formed the nucleus of the automated project-management tools.

VI. CONCLUSION

The 3B20D Processor is operating effectively in the field since the first cutover in September 1981. The rapid field buildup during the first six months (24 machines cut into service) could not have been possible if all parts of the system were not of the highest quality and designed for high reliability. Much of the success of the project is attributed to the extensive testing both by the DMERT development organization, Western Electric organizations and application organizations during each step of the system's introduction.

VII. ACKNOWLEDGMENTS

The authors are reporting on the work of many processor system and application personnel in Bell Laboratories and Western Electric. We wish to acknowledge all of their efforts and their dedication, which made the project so successful. In particular, the authors acknowledge the following individuals for assisting in the preparation of this text: R. J. Colby, J. M. Field, K. A. Giesting, D. I. Sandel, P. J. Stankus, R. R. Stozek, and D. S. Trushin.

REFERENCES

1. H. A. Hilsinger, K. D. Mozingo, C. F. Starnes, and G. A. Van Dine, "1A Processor: Testing and Integration," B.S.T.J., *56* (February 1977), p. 289.
2. C. Haugk, S. H. Tsiang, and L. Zimmerman, "System Testing of the No. 1 Electronic Switching System," B.S.T.J., *43* (September 1964), p. 2575.
3. D. R. Barney, P. K. Giloth, and H. G. Kienzle, "No. 1 ESS ADF: System Testing and Early Field Experience," B.S.T.J., *49* (December 1970), p. 2975.
4. B. P. Donohue, III and J. F. McDonald, "SAFEGUARD Data-Processing System: Processor-System Testing and the System Exerciser," B.S.T.J. (1975), SAFEGUARD Supplement, p. S111.
5. J. L. Quinn, R. L. Engram and F. M. Goetz, "The 3B20D Processor & DMERT Operating System: Diagnostic Tests and Control Software," B.S.T.J., this issue.
6. M. W. Rolund, J. T. Beckett, and D. A. Harms, "The 3B20D Processor & DMERT Operating System: 3B20D Central Processing Unit," B.S.T.J., this issue.
7. R. C. Hansen, R. W. Peterson, and N. O. Whittington, "The 3B20D Processor & DMERT Operating System: Fault Detection and Recovery," B.S.T.J., this issue.
8. R. E. Haglund and L. D. Peterson, "The 3B20D Processor & DMERT Operating System: 3B20D File Memory Systems," B.S.T.J., this issue.
9. M. E. Grzelakowski, J. H. Campbell, and M. R. Dubman, "The 3B20D Processor & DMERT Operating System: DMERT Operating System," B.S.T.J., issue.
10. B. R. Rowland and R. J. Welsch, "The 3B20D Processor & DMERT Operating System: Software Development System," B.S.T.J., this issue.

ACRONYMS AND ABBREVIATIONS

ACHI	application channel interface
ACP	alarm control process
AJT	active job table
ALU	arithmetic logic unit
ALW	allow
API	attached processor interface
APS	Attached Processor System
ATB	address translation buffer
ATP	all tests passed
BGB	bidirectional gating bus
BIC	bus interface controller
BLDG PWR	building power
BPI	bits per inch
BS	bus simulator
BWM	broadcast warning message
CAD	computer-aided design
CC	central control
CCIO	central control input/output
CCIS	common channel interoffice signaling
CCITT	International Telegraph and Telephone Consultative Committee
CH	channel
CHAN	channel
CIA	critical indicator area
CIH	craft interface handler
CMS	Change Management System
CONFIG	configuration management program
CPD	common processor display
CPH	communication protocol handler
CPU	central processing unit
CRT	cathode ray tube
CSU	cache store unit
CU	control unit
DAP	display administration process
DATA TABLE	diagnostic data table files
DBEVOL	Data Base Evolution System
DBS	duplex bus selector
DCB	diagnostic control block
DCT	dispatch control table
DDCMP	digital data communication message protocol
DDL	data definition language

DDLP	data definition language processor
DDSBS	duplex dual-serial bus selector
DEV	device
DFC	disk file controller
DFDIAG	disk file diagnose
DFI	disk file inverter
DGN	diagnose
DIAGC	diagnostic control
DIAMON	diagnostic monitor
DIO	DMA I/O bus
DIP	dual in-line package
DMA	direct memory access
DMAC	direct memory access controller
DMERT	Duplex Multiple Environment Real Time
DML	data manipulation language
DMU	data manipulation unit
DPT	diagnostic phase table
DRAM	dynamic random access memory
DSCH	dual serial channel
DST	destination
DUC	dual-access utility circuit
DUI	direct user interface
EAI	emergency action interface
ECC	error correction code
ECD	equipment configuration data or data base
ECDMAN	equipment configuration database manager
EIH	error interrupt handler
EOS	extended operating system
EPROM	erasable programmable read-only memory
ER	error register
ESS	Electronic Switching System
EX	exercise
FIFO	first in-first out
FIST	factory/installation software test
FPS	form processing system
FST	factory system tests
FTAM	forms translation and mapping
FTS	field test set
GRASP	generic access package
IB	instruction buffer
INH	inhibit
IODRV	IOP driver process
IOP	input/output processor
IOPC	input/output processor controller

IPC	interprocess communication
IPS	inches per second
KPCB	kernel process control block
LC	line controller
MSGS	messages
MSI	medium-scale integration
MTC	maintenance
MTTR	mean time to repair
MTTY	maintenance TTY
MTTYC	maintenance terminal controller
MU	mask unit
MUX	multiplexor
NCP	network control point
NK	non-killable
NRZ	non-return to zero
ODIN	on-line data integrity
OOS	out of service
OP	display
OST	operating system trap
PA	program address
PC	peripheral controller
PCB	process control block
PCSD	peripheral controller subdevice
PCU	power control unit
PD	peripheral device
PDF	page description file
PDFGEN	PDF generator
PDS	program documentation standard
PE	phase encoded
PFC	peripheral frame control
PIC	peripheral interface controller
PID	process identifier
PINIT	processor initialization program
PRM	processor recovery message
PROM	programmable read-only memory
PSBR	primary segment base register
PSDC	parallel serial data interface
PSI	peripheral system interface
PSW	program status word
QA	quality assurance
RAM	random access memory
RC/V	recent change/verify
RFI	radio frequency interference
RMU	rotate mask unit

RMV	remove
ROP	read-only printer
ROP	receive-only printer
RST	restore
RSTU	restore unconditional
RTS	real-time status
RU	rotate unit
SAC	store address control/controller
SAG	system analysis group
SAR	store address register
SAT	store address translator
SBR	segment base register
SC/SD	scanner/signal distributor
SC	software control
SCANS	Software Change and Notification System
SCC	switching control center
SCCS	Source Code Control System
SCCS	Switching Control Center System
SCH	serial channel
SCM	store complete signal
SCR	silicon controlled rectifier
SCR	store control register
SCSD	scanner and signal distributor
SD	software development
SDC	store data controller
SDP	software demand paging
SDR	store data register
SDS	Software Development System
SG	system generation data base
SGO	store go signal
SGS	Software Generating System
SID	segment identifier
SIM	system integrity monitor
SIR	store instruction register
SMD	storage module drive
SP	software production
SRC	source
SREG	special registers
SSA	system summary area
SSBR	secondary segment base register
SSI	small-scale integration
SSR	system status register
STOP	terminate
SUPR	system update program

SYSGEN	system generator/generation
TB/IS	transaction block and integrity subsystem
TLDB	trouble locating data base
TLP	trouble locating process
TSPS	Traffic Service Position System
TTL	transistor-transistor logic
TTY	teletypewriter or terminal
TTYC	terminal controller
TUS	Test Utility System
TV	transfer vector
UC	utility circuit
UID	utility identifier
USP	*UNIX* supervisor process
VLSI	very large-scale integration
VOH	voltage output high
VOL	voltage output low
VTOC	volume table of contents
WCS	writeable control store
WMS	writeable microstore
YACC	yet another compiler compiler

CONTRIBUTORS TO THIS ISSUE

Roy E. Anderson, B.S.E.E., 1970, University of Illinois; M.S.E.E., 1972, University of Maryland; Bell Laboratories, 1970—. Mr. Anderson initially worked in the field of computer-controlled electrical measurement equipment for transmission facilities. His interest in computers led him into the area of designing real-time operating systems. Today, he is involved with developing a distributed digital electronic switch.

Marshall E. Barton, B.A., 1962, M.S. (Mathematics), 1964, Miami University; Bell Laboratories, 1964—. Mr. Barton initially was responsible for the development of support software for No. 2 ESS and No. 3 ESS. Since 1979 he has been associated with 3B20D DMERT, most recently as Supervisor of the Craft Interface Design Group.

J. T. Beckett, B.S. (Engineering), 1961, Harvey Mudd College; M.S.E.E., 1964, Ph.D. (Electrical Engineering), 1967, Case Western Reserve University; Bell Laboratories, 1967—. Part time lecturer, Electrical Engineering, Illinois Inst. of Tech., 1968–1975. Responsible for microcode on 3A processor and for the development of the 3B20D microcode and microcode tools. He also has been responsible for the development of software debugging tools. He is currently Supervisor of the No. 5 ESS Project Management Tool Development Group. Member, ACM, IEEE.

Harry. L. Bosco, B.S.E.E., 1972, Monmouth College, M.S.E.E., 1974, Polytechnic Institute of Brooklyn; Bell Laboratories, 1965—. Mr. Bosco has worked in data communications and hardware design for No. 4 ESS. He was promoted to Supervisor of the Network Design Group for No. 5 ESS in 1977 and to Head of the No. 5 ESS Line Interface and Peripheral Circuits Department in 1980. In 1981, he became Head of the No. 5 ESS Product Management Department, responsible for the generic planning, system architecture, and management of the No. 5 ESS product line. Member, Sigma Pi Sigma, Eta Kappa Nu.

John M. Brown, B.S.M.E., 1967, M.S. (M.E. & I.E.), 1969, University of Michigan; Bell Laboratories, 1969—. Mr. Brown started his career with Bell Laboratories by working on the development of 1A Technology apparatus. He has worked on the development of 1A Processor semiconductor stores and supervised the physical design of the 3B20D Processor. In 1980, he was appointed Head of a department responsible for the circuit design of the 3B20D Processor Control Frame and for physical design of the 3B Processor family.

A. H. Budlong, B.E.E., M.S. (Physics), Marquette University, 1950; Member of the teaching staff of the Department of Physics, Marquette University, 1948–1952; Bell Laboratories, 1953—. Since joining Bell Laboratories, Mr. Budlong has been engaged in exploratory development of electronic switching circuits and has conducted a group in charge of switching training at Bell Laboratories. He was involved in the development of the No. 1 electronic switching system in the areas of trunk and service circuits, and automatic message recording equipment, and in the design of high-speed communication buses and power facilities for the 1A Processor. He has been in charge of a group developing magnetic tape file systems, printers, and miscellaneous peripheral circuits for the 3B20D Processors. Mr. Budlong is the coauthor of a book entitled *Electronic Switching Theory and Circuits.* He is the Dean of the Undergraduate School and also a Professor of Electronic Engineering at the Midwest College of Engineering. Member, Sigma Pi Sigma and Pi Mu Epsilon.

J. H. Campbell, B.S. (Mathematics), 1964, Pittsburg State University; M.S., 1969, Ph.D, 1974 (Computer Science), Iowa State University; Bell Laboratories, 1974—. Since joining Bell Laboratories, Mr. Campbell has worked on the Extended Operating System (EOS) project and on the 3B20D/DMERT project doing operating systems development. Currently, he supervises a group engaged in the development and support of I/O drivers for real-time applications. Member, ACM, Sigma Xi.

J. G. Chevalier, B.E.E., 1951, Ohio State University; Bell Laboratories, 1956—. Mr. Chevalier has worked on a printed wiring board processes, applications for both military and telephone projects, connector design, studies of contact finishes, and the physical design and packaging of electronic equipment for switching systems. He presently supervises a group responsible for the development of packaging technologies for future versions of the 3B20D Processor.

Noel X. DeLessio, B.S.E.E., 1960, M.S.E.E., 1961, Ph.D. (Electrical Engineering), 1966, Polytechnic Institute of Brooklyn; Bell Laboratories, 1966—. Mr. DeLessio worked on SAFEGUARD system design and supervised guidance design for the SPRINT missile system. Subsequently, he supervised the Exploratory Development Group of the Operator System Laboratory and is currently Head of that Laboratory's Processor Applications Department.

M. R. Dubman, A.B., 1957, Princeton University; M.S., 1959, Massachusetts Institute of Technology; Ph.D., 1970, University of California-Los Angeles; Bell Laboratories, 1970—. From 1959 to 1970, Mr. Dubman was engaged in the development of statistical analysis techniques used in testing of Saturn/Apollo rocket engines for Rockwell International. At Bell Laboratories he has been involved in the development of software for the 1A and 3B20D Processors. Presently, he is Supervisor of a group responsible for the 3B20D system laboratories.

Gary P. Eldredge, B.S.E.E., 1971, M.S.E.E., 1971, Brigham Young University; Bell Laboratories, 1972—. Mr. Eldredge has designed a number of programs for the 1A Processor to assist debugging and updating software in operational field sites. Since 1977, he has been involved in the development, administration, and system testing of DMERT software. He recently was responsible for a group that designs utilities that are used in field sites to test, monitor, and maintain various parts of the software. He currently is Supervisor of the 3B20D fault recovery group. Member, Tau Beta Pi, Phi Kappa Phi, Sigma Xi.

Robert L. Engram, B.S.E.E., 1969, Howard University; M.S.E.E., 1973, Stanford University; Motorola, Portable Products Division, 1969–72; Bell Laboratories, 1972—. Prior to joining Bell Laboratories Mr. Engram was involved in design and development of analog and digital circuits for paging and terminal equipment. He received a patent for his work inhibiting shock falsing in a two-tone sequential decoder. Upon joining Bell Laboratories, Mr. Engram was initially involved in exploratory development of digital circuits and systems including AP3 and VSS. In 1977, he became Supervisor of Test Facilities for TSPS and later supervised a group performing Software Integration and Testing. He joined the 3B20D/DMERT Project in 1980 as Head of the Processor System Integrity Department and presently heads the Operating System Development Department. Member, IEEE, Tau Beta Pi, National Technical Association.

Rudolph J. Frank, B.S.E.E., 1966, Seattle University, M.S.E.E., 1968, Ph.D. (Electrical Engineering), 1971, Oregon State University; M.S. (Business Management), 1981, Stanford University; Pacific Northwest Bell, 1964–1966; Bell Laboratories, 1971—. At Pacific Northwest Bell, Mr. Frank was an electronics data processing supervisor in the comptroller's division. At Bell Laboratories he has worked on new feature planning and exploratory development in the Traffic Service Position System laboratory. In 1975, he was designated Bell

Laboratories Visiting Professor of Electrical Engineering at Southern University (Baton Rouge, La). Mr. Frank became Supervisor of the No. 4 ESS Network Management Control Group in 1976 and has managed several large software development projects. In 1980 he was awarded a Sloan fellowship in the School of Business at Stanford University. Mr. Frank is now Head of the Toll Digital Maintenance Planning and Development Department at Bell Laboratories, Member, IEEE, Eta Kappu Nu.

Alan W. Fulton, B.S.E.E., 1966, University of Arizona; M.S.E.E., 1967, Ph.D. (Electrical Engineering), 1971, Stanford University; Bell Laboratories, 1966—. Mr. Fulton has been involved in the design of processors for electronic switching systems. He currently is Head of the Processor Technology Department. This department is responsible for developing and integrating the silicon, packaging, and computer aided design tools required for processors. Member, Tau Beta Pi, Sigma Xi.

Lee E. Gallaher, B.S.E.E., 1951, M.S.E.E., 1956, Case Western Reserve University; Instructor in Electrical Engineering at Case Western Reserve University, 1952–1955; Bell Laboratories, 1955—. Mr. Gallaher has worked on memory systems for ESS including the Flying Spot Store, the Twister Memory and integrated circuit memories. He was promoted to Department Head in 1965 and was responsible for the design of the switch units for the No. 101 ESS and circuit designs for the No. 2 ESS and the No. 3 ESS. More recently he has been responsible for the architecture and development of the 3B20D processor. Member, IEEE, Tau Beta Pi, Sigma Xi, Eta Kappa Nu.

Frank M. Goetz, B.E.E., 1953, Manhattan College; M.S. (Mathematics), 1960, New York University; Bell Laboratories, 1953—. Mr. Goetz has worked on logic design, software development, and system design for electromechanical and electronic switching systems. He received five patents in the areas of electronic counters, error-correction circuitry, signal detection, and processor system design. Since 1962, he has been a Supervisor responsible for various aspects of electronic switching system and processor maintenance. At present, he is responsible for maintenance design of the 3B20S Processor. Member, IEEE, Eta Kappa Nu.

Maureen Grzelakowski, B.S., 1976, M.S., 1978 (Computer Science), Northwestern University; Bell Laboratories, 1977—. Ms. Grzelakowski was initially responsible for designing software development

environments for ESS projects. In 1980 she participated in the development of the DMERT nucleus. Since then she has supervised the Operating System Architecture and Planning, and Generic 2 Engineering Groups.

Ronald E. Haglund, B.S.E.E., 1960, M.S.E.E., 1963, and Ph.D. (Electrical Engineering), 1969, Iowa State University; Bell Laboratories, 1970—. Mr. Haglund initially worked on file store design for the No. 1A Processor. In 1979 Mr. Haglund became Supervisor of the 3B20D File Store Group with responsibility for the file stores for the 3B20D and other processors. He currently is Supervisor of the Peripheral Technology Group with responsibility for all 3B peripheral devices.

Robert C. Hansen, B.S.E.E., 1966, and M.S.E.E., 1969, Michigan State University; Ph.D. (Control Science), 1973, University of Minnesota; Bell Laboratories, 1973—. Mr. Hansen has worked on maintenance software for No. 4 ESS and the 3B20D. He is presently Supervisor of the Requirements and Architecture Group, concerned with feature evolution of the 3B20D. Member, IEEE, Tau Beta Pi.

D. A. Harms, B.S.E.E., 1963, M.S.E.E., 1965, University of Minnesota; Bell Laboratories, 1965—. Mr. Harms has been involved with the design of processors since 1967. Since 1970, he has supervised groups responsible for the design of memory subsystems, microstores, and peripherals for the No. 2 ESS, No. 2A ESS, No. 3 ESS, 3A Processor, and the 3B20D Processor. He is currently Supervisor of the Common Processor Circuits group responsible for the design of a diagnostic processor for 3B20S and for circuits incorporating the *Bellmac*™-32A microprocessor in single board computers. Member, Eta Kappa Nu, Phi Theta Kappa.

Irvine K. Hetherington, B.S.E.E., 1966, Lehigh University; M.S.E.E., 1967, Stanford University; Bell Laboratories, 1972—. Mr. Hetherington initially was associated with exploratory development of integrated circuit analog switching networks for telecommunications applications. From 1972 to 1976, he was involved with the design of the microcontrol store and main memory system of the 3A Processor. Since 1977, Mr. Hetherington has had several design and supervisory responsibilities associated with the CPU and memory systems of the 3B20D Processor. He currently supervises the 3B20D Processor Data Communications Peripherals Group. Member, Tau Beta Pi, Eta Kappa Nu.

J. Richard Kane, B.S.E.E., 1968, University of Pittsburgh; M.S.E.E., 1970, Northwestern University; Ph.D. (Computer Science), 1973, Northwestern University; Bell Laboratories, 1968—. Mr. Kane initially worked in the areas of fault detection and system recovery for No. 4 ESS and later on testing tools. He system tested the first No. 4 ESS. Mr. Kane then supervised groups responsible for the development of the DMERT operating system, testing tools for the 3B20D Processor and developing local area networks. He currently supervises a group planning the design and development of new processors.

W. F. Klinksiek, B.S.M.E., 1966, M.S.M.E., 1967, and Ph.D. (Mechanical Engineering), 1971, Virginia Polytechnic Institute and State University; Bell Laboratories, 1971—. Mr. Klinksiek has worked on the physical design and packaging of the 1A Processor and the 3B20D Processor specializing in thermal and power engineering. He has supervised the development of packaging technology and computer automated design systems and managed software administration for the 3B20D. Recently, he was a Supervisor responsible for generic engineering, application interface, and application engineering for the 3B20D Processor System. Currently, he is Head of Processor Systems Customer Support Department. Member, Phi Kappa Phi, Tau Beta Pi, ASME.

S. H. Kulpa, B.S.M.E. 1972, Michigan Technological University; M.S.M.E., Stanford University; Bell Laboratories, 1973—. Mr. Kulpa has been involved with the design and development of the packaging technology for the 3B20D Processor. He currently is Supervisor of a physical design group.

Peter Kusulas, B.S.E.E., 1963, M.S.E.E., 1965, Rutgers University; Bell Laboratories 1963—. Mr. Kusulas has worked on the development of magnetic and semiconductor memory systems for the 101 ESS, No. 2 ESS, and the 3A and 3B20D Processors. Since 1980 he has supervised several groups engaged in the development of the *Bellmac*™ microprocessor module and systems based upon it.

N. A. Martellotto, B.E.E., and B.S., Applied Mathematics, 1957, Georgia Institute of Technology; M.E.E., 1959, New York University; M.B.A., 1970, University of Chicago; Bell Laboratories, 1957—. Starting with the Bell System Data Processor project in 1957, where he did logic design and programming, Mr. Martelloto has been involved with computers and software development throughout his Bell Laboratories

career. Early projects include EPBX and No. 1 ESS. Mr. Martellotto holds a patent related to the basic notion of ESS generic programs. In 1966 he became Department Head of the Indian Hill Computation Center (IHCC). In 1976, he resumed design and development of ESS software development support programs and other related work. In late 1979, Mr. Martelloto became DMERT project manager and for the next two years was involved with all aspects of the project, from operating system development to field support. He is now Head of the Software Development Systems Department. Member, IEEE, Tau Beta Pi, Eta Kappa Nu.

Peter S. McCabe, B.S. (Engineering), 1956, Trinity College; B.S.E.E., 1957, Rensselaer Polytechnic Institute; Bell Laboratories, 1957—. Mr. McCabe has worked on memories for Nike-Zeus, digital circuits for the 101 EPBX, software design for the UNICOM, AUTOVON, four-wire No. 1 ESS, and No. 4 ESS projects. Mr. McCabe supervised support software system development and program administration development and operation for No. 4 ESS. Since 1980 Mr. McCabe has supervised the performance measurements group for the DMERT project. Member, Tau Beta Pi, Eta Kappa Nu, Sigma Pi Sigma; associate member, Sigma Xi.

JoAnne H. Miller, B.S. (Mathematics), 1967, University of Michigan; M.S. (Computer Science), 1976, University of Colorado; GTE-Sylvania, 1968–1970; University of Colorado-Institute of Behavioral Science, 1971–1976; Bell Laboratories, 1976—. Prior to joining Bell Laboratories Ms. Miller was involved in system analysis and real-time programming for missile control and interactive computer graphic applications. Upon joining Bell Laboratories Ms. Miller was initially involved in No. 1 ESS software restructure design. Since early 1979 she has supervised groups responsible for the system testing and delivery of microprocessor software development systems, and design and development of several components of 3B20D/DMERT System, including the Recent Change/Verify System and equipment configuration data base. Currently, she supervises a group responsible for the field deployment and support of the 3B20D/DMERT system. Member, IEEE, ACM.

H. L. Mitchell, B.S.E.E., 1970, Rensselaer Polytechnic Institute; M.B.A., 1981, Illinois Benedictine College; Western Electric, 1970—. Mr. Mitchell's original assignment was to work with Bell Laboratories on system testing of the Safeguard MSR system. He has worked on

No. 4 ESS software development and testing and on 3B20D/DMERT system testing, system integration, and 3B20D Field Support. He is currently Department Chief, Call Processing and Residential Feature Development—1/1A ESS.

Robert W. Mitze, B.S. (Mathematics), 1969, California Institute of Technology; M.S. (Mathematics), M.S. (Computer Science), 1976, University of Wisconsin; Bell Laboratories, 1976—. Mr. Mitze has worked in the development of software engineering environments for the C language, project management of the DMERT operating system development, and distributed architecture specification for special-purpose applications. Since March 1981, he has been Head of the Advanced Operating System Development Department.

B. G. Niedfeldt, B.S.E.E., 1959, University of Maryland; M.S.E.E., 1961, New York University; Bell Laboratories, 1959–1962; Bellcomm, Inc., 1962–1970; Bell Laboratories, 1970—. Mr. Niedfeldt was engaged in exploratory development of high-speed data terminals. While at Bellcomm, he participated in the evaluation of the guidance and navigation aspects of the Apollo lunar landing missions. He then joined the Safeguard project, working in the multiprocessor operating system area. In 1974, he joined the No. 4 ESS team and, among other things, was responsible for the first generic (4E0) system test and release. In 1977, he joined the 3B20D/DMERT software development team, and is presently Supervisor of the Operating System, Software Integrity and Data Base Systems Group. Member, Tau Beta Pi, Eta Kappa Nu, Phi Kappa Phi, and Omicion Deta Kappa.

Leland D. Peterson, B.S.M.E., 1971, Illinois Institute of Technology; M.S.M.E., 1974, Northwestern University; Bell Laboratories, 1966—. Mr. Peterson has been involved with physical design of the 1A Processor, Voice Storage System (VSS), 1A Attached Processor System, and the 3B20D family of processors. He has participated in exploratory physical design and in the development of new hardware packaging technology. Currently he is Supervisor of the Large Processor Physical Design Group.

Ralph W. Peterson, B.S. (Physics), 1961, Wayne State University; M.S.E.E., 1963, New York University; Bell Laboratories, 1961—. Mr. Peterson has worked on software development on several electronic switching systems, including extensive work on software development tools. He presently is Supervisor of the Processor Microcode Group,

responsible for the development of microcode for 3B Processors. Member, ACM, Eta Kappa Nu, Tau Beta Pi.

John L. Quinn, B.S.E., 1959, Stevens Institute of Technology; M.S.E.E., 1961, New York University; Bell Laboratories, 1959—. Mr. Quinn worked on the design and system testing of the No. 1 ESS. He has been Supervisor of groups concerned with ESS system testing and development of maintenance software. For the 1A Processor project he was responsible for the processor diagnostics and for CPU logic/fault simulation. During the development of the 3B20D Processor he was responsible for the Processor Control Unit diagnostics and for the diagnostic control programs. Member, IEEE, Eta Kappa Nu.

Michael W. Rolund, B.S.E.E., 1961, Cooper Union; M.S.E.E., 1963, New York University; Bell Laboratories, 1961—. Mr. Rolund has been involved with the development of memory systems and processors including No. 1 ESS, the 1A Processor, and the 3B processor family. He currently is Head of the Processor Design Department responsible for the 3B20D and its evolution. Member, IEEE, Tau Beta Pi.

Bruce R. Rowland, B.S., 1973, Michigan State University; M.S., 1975, and Ph.D. (Computer Science), 1977, University of Wisconsin-Madison; Bell Laboratories, 1977—. Mr. Rowland began work in the area of languages and compilers, where he helped to extend the C programming language and coordinated the development of compilation tools for four processors, including the 3B20D Processor. He currently is supervising a group planning the development of 3B Processor networking and doing exploratory work in computer system design.

Jack M. Scanlon, B.S. (Applied Science), 1964, University of Toronto; M.S.E.E., 1965, Cornell University; Bell Laboratories 1965—. Mr. Scanlon joined Bell Laboratories in 1965, and initially worked on fault-diagnosis, resolution, and recovery techniques for the Bell System's No. 1 ESS. He also worked on a missile flight simulator for the Safeguard project. In 1968, he became Supervisor of the No. 1 ESS Call Program Group, responsible for the development of new customer call-features. In 1971, he joined the initial development team for No. 4 ESS with responsibility for system design and call-handling software. In 1974, he was promoted to Head of the No. 4 ESS System Design and Operations Department, where he was responsible for design of new capabilities for No. 4 ESS in domestic and international

applications, and for exploratory work on new software techniques. In 1977, he became Director of a Laboratory where he was responsible for the exploration of a secure voice/data communications system for the government. In June 1979, he was appointed Executive Director, Processor and Common Software Systems Division. In November 1982 he was appointed to the newly created position of Vice President, Processors, Western Electric. This work involves development of processors, microprocessors, *UNIX** operating systems, and programming languages and tools for Bell System applications. Mr. Scanlon has been granted four patents in processor design and electronic switching system design. He has published numerous articles on processor design for real-time, time-shared systems, electronic switching systems design, and software development techniques. Member, Computer Science Board of the National Academy of Sciences, IEEE.

D. A. Schmitt, B.S.E.E., 1965, St. Louis University; M.S. (Mathematics), 1968, Stevens Institute of Technology; Southwestern Bell, 1962–1965; Bell Laboratories, 1965–1969 and 1973—. At Bell Laboratories, Mr. Schmitt has been involved in the TSPS, No. 3 ESS, and VSS projects. He currently is in charge of the DMERT Operating System Architecture Department. Member, Eta Kappa Nu, Pi Mu Epsilon, Alpha Sigma Nu.

W. C. Schwartz, B.S.E.E., 1966, Purdue University; M.S.E.E., 1967, University of Michigan; Bell Laboratories, 1966—. Mr. Schwartz has worked on No. 4 ESS call-processing and fault-recovery software subsystems, software testing methodologies, and logic analysis and simulation systems. More recently, he was the generic engineer and application interface for DMERT. He currently is Supervisor of the Operating System Group in the Processor and Operating Systems Development Laboratory. Member, IEEE, Tau Beta Pi, Sigma Xi.

Wing N. Toy, B.S.E.E., 1950, and M.S.E.E., 1952, University of Illinois, and Ph.D. (Electrical Engineering), 1969, University of Pennsylvania; Bell Laboratories, 1952—. Mr. Toy has been involved in the design of highly reliable processors for the Bell System electronic switching systems and other telephone-related applications for the past 25 years. He was on the Faculty of the Computer Science Division of Electrical Engineering at the University of California, Berkeley, as a Visiting MacKay Lecturer during the 1973–1974 academic year. Mr.

* Trademark of Bell Laboratories.

Toy holds 19 U.S. patents and is co-author of two books, *Mini/Microcomputer Hardware Design* and *Microprogrammed Control and Reliable Design of Small Computers.* He is currently Supervisor of the Voice/Data Technology Group. Fellow, IEEE.

Susan S. Weber, B.S. (Mathematics and Computer Science), 1977, Iowa State University; M.S. (Computer Science), 1978, Purdue University; Bell Laboratories, 1977—. Ms. Weber initially designed software development systems with her primary emphasis on source administration and object generation. Since becoming involved with DMERT, she has worked on both system update and field update and has participated in generic engineering and application interface issues. Currently, she is the Supervisor of the Field Update Group in the Operating System Development Department.

R. J. Welsch, B.S. (Mathematics), 1967, Marquette University; M.S. (Computer Science), 1972, Northwestern University; Bell Laboratories, 1967–1968; U.S. Army, 1968–1970; Bell Laboratories, 1970—. At Bell Laboratories, Mr. Welsch has had experience with No. 1 ESS/ADF, No. 4 ESS, and systems engineering for electromechanical switching systems. From 1974 to 1978, he was a member of the Program Administration and Support Program Group of the Operator Systems Laboratory (TSPS). From 1978 to 1980, Mr. Welsch was involved with developing software development systems utilizing PDP 11/70's running the *UNIX* operating system as front-end processors to larger IBM main frames. Since 1980, he has been involved with 3B20D/DMERT software development and administration. Mr. Welsch is currently Supervisor of the DMERT Administrative Software Systems Group.

F. W. Wendland, B.S.E.E., 1965, M.S.E.E., 1966, Cornell University; Bell Laboratories, 1966—. Mr. Wendland has been involved in the development of electromagnetic compatibility facilities and standards for ESS systems, automatic test facilities for ESS processor subsystems, 1A and 3B20D Processors, and I/O system architecture and design. He currently supervises a group working on the architecture and design of networking hardware for the 3B processor line. Member, IEEE.

Neil O. Whittington, B.S. (Physics), 1965, M.S. (Physics), 1970, Illinois State University; Bell Laboratories, 1970—. Mr. Whittington has worked on the development of software for the 1A Processor and the 3B20D Processor. He is presently Head of the Application Support Department, concerned with DMERT generic planning and application interfaces with the 3B20D. Member, IEEE.

Robert M. Wolfe, B.S.E.E., 1952, University of Louisville; M.S.E.E., 1957, Columbia University; Bell Laboratories, 1952—. From 1952 to 1962, Mr. Wolfe worked on applied research on ferromagnetic and ferroelectric devices and on data collection and data transmission systems. In 1962, he joined the Switching System Development area, where he headed the development of the Automatic Intercept System, the AMA Recording System, the Service Evaluation System, and the Network Control Point System. He is currently Head, Network Control Point Department. Member, ACM, IEEE.

Robert H. Yacobellis, B.S. (Mathematics), 1967, Carnegie-Mellon University; M.S., 1969, Ph.D., 1973 (Information Sciences), University of Chicago; Bell Laboratories, 1967—. Mr. Yacobellis worked initially on No. 1 ESS 4-wire AUTOVON (Government Switching). From 1969 through 1977 he was in No. 4 ESS working on overload control and later on program administration. In 1978 he became Supervisor of a group providing common loaders and simulators to ESS projects, and in 1980 became responsible for the 3B20D Field Update Group. He currently is in charge of documentation and software quality for the 3B20D. Member, Tau Beta Pi, ACM.

THE BELL SYSTEM TECHNICAL JOURNAL is abstracted or indexed by *Abstract Journal in Earthquake Engineering, Applied Mechanics Review, Applied Science & Technology Index, Chemical Abstracts, Computer Abstracts, Current Contents/ Engineering, Technology & Applied Sciences, Current Index to Statistics, Current Papers in Electrical & Electronic Engineering, Current Papers on Computers & Control, Electronics & Communications Abstracts Journal, The Engineering Index, International Aerospace Abstracts, Journal of Current Laser Abstracts, Language and Language Behavior Abstracts, Mathematical Reviews, Science Abstracts (Series A, Physics Abstracts; Series B, Electrical and Electronic Abstracts; and Series C, Computer & Control Abstracts), Science Citation Index, Sociological Abstracts, Social Welfare, Social Planning and Social Development,* and *Solid State Abstracts Journal.* Reproductions of the Journal by years are available in microform from University Microfilms, 300 N. Zeeb Road, Ann Arbor, Michigan 48106.