RELIABLE COMPUTER SYSTEMS

DESIGN AND EVALUATION

SECOND EDITION

DANIEL P. SIEWIOREK
ROBERT S. SWARZ

digital
DIGITAL PRESS

Printed in the United States of America.
9 8 7 6 5 4 3 2 1

Order number EY-H880E-DP

The Publisher offers discounts on bulk orders of this book. For information, please write:

Special Sales Department
Digital Press
One Burlington Woods Drive
Burlington, MA 01003

Design: Outside Designs
Production: Technical Texts
Composition: DEKR Corporation
Printer: Arcata/Halliday

Trademark products mentioned in this book are listed on page 890.

Library of Congress Cataloging-in-Publication Data

Siewiorek, Daniel P.
Reliable computer systems : design and evaluation / Daniel P. Siewiorek, Robert S. Swarz. — 2nd ed.
p. cm.
Rev. ed. of: The theory and practice of reliable system design. Bedford, MA : Digital Press, c1982.
Includes bibliographical references and index.
ISBN 1-55558-075-0
1. Electronic digital computers—Reliability. 2. Fault-tolerant computing. I. Swarz, Robert S. II. Siewiorek, Daniel P. Theory and practice of reliable system design. III. Title.
QA76.5.S537 1992
004-dc20 92-10671
CIP

CREDITS

Figure 1–3: Eugene Foley, "The Effects of Microelectronics Revolution on Systems and Board Test," *Computers*, Vol. 12, No. 10 (October 1979). Copyright © 1979 IEEE. Reprinted by permission.

Figure 1–6: S. Russell Craig, "Incoming Inspection and Test Programs," *Electronics Test* (October 1980). Reprinted by permission.

Credits are continued on p. 885, which is considered a continuation of the copyright page.

To Karon and Lonnie

A Special Remembrance:

During the development of this book, a friend, colleague, and fault-tolerant pioneer passed away. Dr. Wing N. Toy documented his 37 years of experience in designing several generations of fault-tolerant computers for the Bell System electronic switching systems described in Chapter 8. We dedicate this book to Dr. Toy in the confidence that his writings will continue to influence designs produced by those who learn from these pages.

CONTENTS

8 HIGH-AVAILABILITY SYSTEMS

INTRODUCTION

Dynamic redundancy is the basic approach used in high-availability systems. These systems are typically composed of multiple processors with extensive error-detection mechanisms. When an error is detected, the computation is resumed on another processor. The evolution of high-availability systems is traced through the family history of three commercial vendors: AT&T, Tandem, and Stratus.

AT&T SWITCHING SYSTEMS

AT&T pioneered fault-tolerant computing in the telephone switching application. The two AT&T case studies given in this chapter trace the variations of duplication and matching devised for the switching systems to detect failures and to automatically resume computations. The primary form of detection is hardware lock-step duplication and comparison that requires about 2.5 times the hardware cost of a nonredundant system. Thousands of switching systems have been installed and they are currently commercially available in the form of the 3B20 processor. Table 8–1 summarizes the evolution of the AT&T switching systems. It includes system characteristics such as the number of telephone lines accommodated as well as the processor model used to control the switching gear.

Telephone switching systems utilize natural redundancy in the network and its operation to meet an aggressive availability goal of 2 hours downtime in 40 years (3 minutes per year). Telephone users will redial if they get a wrong number or are disconnected. However, there is a user aggravation level that must be avoided: users will redial as long as errors do not happen too frequently. User aggravation thresholds are different for failure to establish a call (moderately high) and disconnection of an established call (very low). Thus, a telephone switching system follows a staged failure recovery process, as shown in Table 8–2.

Figure 8–1 illustrates that the telephone switching application requires quite a different organization than that of a general-purpose computer. In particular, a substantial portion of the telephone switching system complexity is in the peripheral hardware. As depicted in Figure 8–1, the telephone switching system is composed of four major components: the transmission interface, the network, signal processors, and the central controller. Telephone lines carrying analog signals attach to the voice band interface frame (VIF), which samples and digitally encodes the analog signals. The output is pulse code modulated (PCM). The echo suppressor terminal (EST) removes echos that may have been introduced on long distance trunk lines. The PCM

TABLE 8–1 *Summary of installed AT&T telephone switching systems*

System	Number of Lines	Year Introduced	Number Installed	Processor	Comments
1 ESS	5,000–65,000	1965	1,000	No. 1	First processor with separate control and data memories
2 ESS	1,000–10,000	1969	500	No. 2	
1A ESS	100,000	1976	2,000	No. 1A	Four to eight times faster than No. 1
2B ESS	1,000–20,000	1975	>500	No. 3A	Combined control and data store; microcoded; emulates No. 2
3 ESS	500–5,000	1976	>500	No. 3A	
5 ESS	1,000–85,000	1982	>1,000	No. 3B	Multipurpose processor

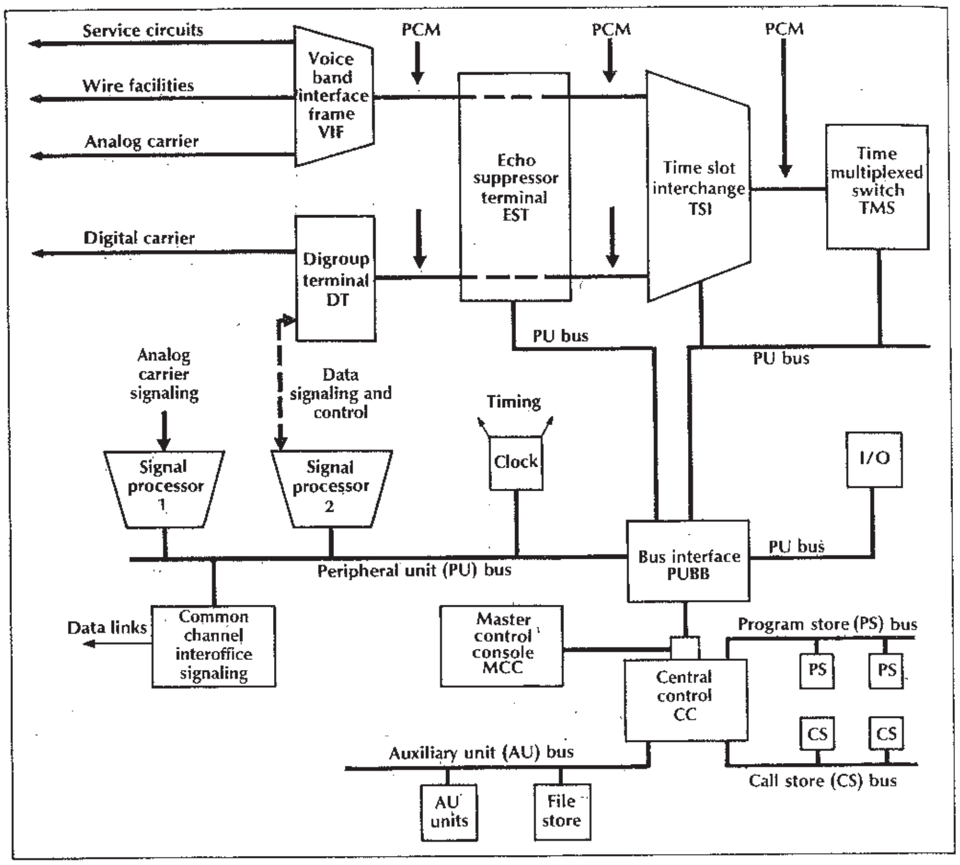
TABLE 8–2 *Levels of recovery in a telephone switching system*

Phase	Recovery Action	Effect
1	Initialize specific transient memory.	Temporary storage affected; no calls lost
2	Reconfigure peripheral hardware. Initialize all transient memory.	Lose calls being established; calls in progress not lost
3	Verify memory operation, establish a workable processor configuration, verify program, configure peripheral hardware, initialize all transient memory.	Lose calls being established; calls in progress not affected
4	Establish a workable processor configuration, configure peripheral hardware, initialize all memory.	All calls lost

signals are multiplexed onto a time-slotted digital bus. The digital bus enters a time-space-time network. The time slot interchange (TSI) switches PCM signals to different time slots on the bus. The output of the TSI goes to the time multiplexed switch (TMS), which switches the PCM signals in a particular time slot from any bus to any other bus. The output of the TMS returns to the TSI, where the PCM signals may be interchanged to another time slot. Signals intended for analog lines are converted from PCM to analog signals in the VIF. A network clock coordinates the timing for all of the switching functions.

The signal processors provide scanning and signal distribution functions, thus relieving the central processor of these activities. The common channel interface signaling (CCIS) provides an independent data link between telephone switching systems. The CCIS terminal is used to send supervisory switching information for the

FIGURE 8–1
Diagram of a typical telephone switching system



various trunk lines coming into the office. The entire peripheral hardware is interfaced to the central control (CC) over AC-coupled buses. A telephone switching processor is composed of the central control, which manipulates data associated with call processing, administrative tasks, and recovery; program store; call store for storing transient information related to the processing of telephone calls; file store disk system used to store backup program copies; auxiliary units magnetic tapes storage containing basic restart programs and new software releases; input/output (I/O) interfaces to terminal devices; and master control console used as the control and display console for the system. In general, a telephone switching processor could be used to control more than one type of telephone switching system.

The history of AT&T processors is summarized in Table 8–3. Even though all the processors are based upon full duplication, it is interesting to observe the evolution from the tightly lock-stepped matching of every machine cycle in the early processors to a higher dependence on self-checking and matching only on writes to memory. Furthermore, as the processors evolved from dedicated, real-time controllers to mul-

TABLE 8–3 *Summary of AT&T Telephone Switching Processors*

Processor/ Year Introduced	Complexity (Gates)	Unit of Switching	Matching	Other Error Detection/Correction
No. 1, 1965	12,000	PS, CS, CC, buses	Six internal nodes, 24 bits per node; one node matched each machine cycle; node selected to be matched dependent on instruction being executed	Hamming code on PS; parity on CS; automatic retry on CS, PS; watch-dog timer; sanity program to determine if reorganization led to a valid configuration
No. 2, 1969	5,000	Entire computer	Single match point on call store input	Diagnostic programs; parity on PS; detection of multiword accesses in CS; watch-dog timer
No. 1A, 1976	50,000	PS, CS, CC, buses	16 internal nodes, 24 bits per node; two nodes matched each machine cycle	Two-parity bits on PS; roving spares (i.e., contents of PS not completely duplicated, can be loaded from disk upon error detection); two-parity bits on CS; roving spares sufficient for complete duplication of transient data; processor configuration circuit to search automatically for a valid configuration
No. 3A, 1975	16,500	Entire computer	None	On-line processor writes into both stores; *m*-of-2*m* code on microstore plus parity; self-checking decoders; two-parity bits on registers; duplication of ALU; watch-dog timer; maintenance channel for observability and controllability of the other processor; 25% of logic devoted to self-checking logic and 14% to maintenance access
3B20D, 1981	75,000	Entire computer	None	On-line processor write into both stores; byte parity on data paths; parity checking where parity preserved, duplication otherwise; modified Hamming code on main memory; maintenance channel for observability and controllability of the other processor; 30% of control logic devoted to self-checking; error-correction codes on disks; software audits, sanity timer, integrity monitor

tiple-purpose processors, the operating system and software not only became more sophisticated but also became a dominant portion of the system design and maintenance effort.

The part I of the AT&T case study in this chapter, by Wing Toy, sketches the evolution of the telephone switching system processors and focuses on the latest member of the family, the 3B20D. Part II of the case study, by Liane C. Toy, outlines the procedure used in the 5ESS for updating hardware and/or software without incurring any downtime.

TANDEM COMPUTERS, INC.

Over a decade after the first AT&T computer-controlled switching system was installed, Tandem designed a high-availability system targeted for the on-line transaction processing (OLTP) market. Replication of processors, memories, and disks was used not only to tolerate failures, but also to provide modular expansion of computing resources. Tandem was concerned about the propagation of errors, and thus developed a loosely coupled multiple computer architecture. While one computer acts as primary, the backup computer is active only to receive periodic checkpoint information. Hence, 1.3 physical computers are required to behave as one logical fault-tolerant computer. Disks, of course, have to be fully replicated to provide a complete backup copy of the database. This approach places a heavy burden upon the system and user software developers to guarantee correct operation no matter when or where a failure occurs. In particular, the primary memory state of a computation may not be available due to the failure of the processors. Some feel, however, that the multiple computer structure is superior to a lock-step duplication approach in tolerating design errors.

The architecture discussed in the Tandem case study, by Bartlett, Bartlett, Garcia, Gray, Horst, Jardine, Jewett, Lenoski, and McGuire, is the first commercially available, modularly expandable system designed specifically for high availability. Design objectives for the system include the following:

- "Nonstop" operation wherein failures are detected, components are reconfigured out of service, and repaired components are configured back into the system without stopping the other system components
- Fail-fast logic whereby no single hardware failure can compromise the data integrity of the system
- Modular system expansion through adding more processing power, memory, and peripherals without impacting applications software

As in the AT&T switching systems, the Tandem architecture is designed to take advantage of the OLTP application to simplify error detection and recovery. The Tandem architecture is composed of up to 16 computers interconnected by two message-oriented Dynabuses. The hardware and software modules are designed to be fast-fail; that is, to rapidly detect errors and subsequent terminate processing. Software modules employ consistency checks and defensive programming techniques. Techniques employed in hardware modules include the following:

Checksums on Dynabus messages
Parity on data paths
Error-correcting code memory
Watch-dog timers

All I/O device controllers are dual ported for access by an alternate path in case of processor or I/O failure. The software builds a process-oriented system with all communications handled as messages on this hardware structure. This abstraction allows the blurring of the physical boundaries between processors and peripherals. Any I/O device or resource in the system can be accessed by a process, regardless of where the resource and process reside.

Retry is extensively used to access an I/O device. Initially, hardware/firmware retries the access assuming a temporary fault. Next, software retries, followed by alternative path retry and finally alternative device retry.

A network systems management program provides a set of operators that helps reduce the number of administrative errors typically encountered in complex systems. The Tandem Maintenance and Diagnostic System analyzes event logs to successfully call out failed field-replaceable units 90 percent of the time. Networking software exists that allows interconnection of up to 255 geographically dispersed Tandem systems. Tandem applications include order entry, hospital records, bank transactions, and library transactions.

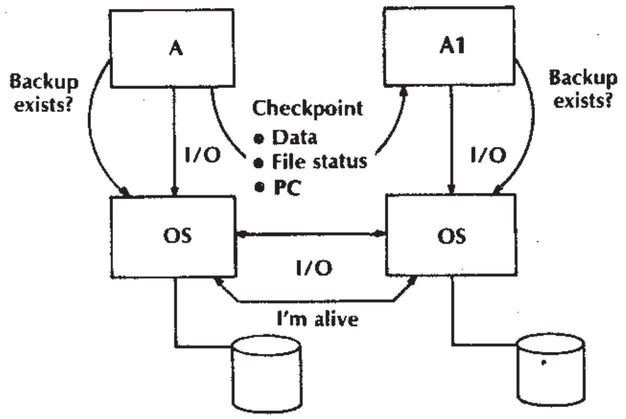
Data integrity is maintained through the mechanisms of I/O "process pairs"; one I/O process is designated as primary and the other is designated as backup. All file modification messages are delivered to the primary I/O process. The primary sends a message with checkpoint information to the backup so that it can take over if the primary's processor or access path to the I/O device fails. Files can also be duplicated on physically distinct devices controlled by an I/O process pair on physically distinct processors. All file modification messages are delivered to both I/O processes. Thus, in the event of physical failure or isolation of the primary, the backup file is up-to-date and available.

User applications can also utilize the process-pair mechanism. As an example of how process pairs work, consider the nonstop application, program A, shown in Figure 8–2. Program A starts a backup process, A1, in another processor. There are also duplicate file images, one designated primary and the other backup. Program A periodically (at user-specified points) sends checkpoint information to A1. A1 is the same program as A, but knows that it is a backup program. A1 reads checkpoint messages to update its data area, file status, and program counter.

The checkpoint information is inserted in the corresponding memory locations of the backup process, as opposed to the more usual approach of updating a disk file. This approach permits the backup process to take over immediately in the event of failure without having to perform the usual recovery journaling and disk accesses before processing resumes.

Program A1 loads and executes if the system reports that A's processor is down (error messages sent from A's operating system image or A's processor fails to respond

FIGURE 8–2
Shadow processor in Tandem



to a periodic "I'm alive" message). All file activity by A is performed on both the primary and backup file copies. When A1 starts to execute from the last checkpoints, it may attempt to repeat I/O operations successfully completed by A. The system file handler will recognize this and send A1 a successfully completed I/O message. Program A periodically asks the operating system if a backup process exists. Since one no longer does, it can request the creation and initialization of a copy of both the process and file structure.

A major issue in the design of loosely coupled duplicated systems is how both copies can be kept identical in the face of errors. As an example of how consistency is maintained, consider the interaction of an I/O processor pair as depicted in Table 8–4. Initially, all sequence numbers (SeqNo) are set to zero. The requester sends a request to the server. If the sequence number is less than the server's local copy, a failure has occurred and the status of the completed operation is returned. Note that the requested operation is done only once. Next, the operation is performed and a checkpoint of the request is sent to the server backup. The disk is written, the sequence number incremented to one, and the results checkpointed to the server backup, which also increments its sequence number. The results are returned from the server to the requestor. Finally the results are checkpointed to the requester backup, which also increments its sequence number.

Now consider failures. If either backup fails, the operation completes successfully. If the requester fails after the request has been made, the server will complete the operation but be unable to return the result. When the requester backup becomes active, it will repeat the request. Since its sequence number is zero, the server test at step 2 will return the result without performing the operation again. Finally, if the server fails, the server backup either does nothing or completes the operation using checkpointed information. When the requester resends the request, the new server (that is, the old server backup) either performs the operation or returns the saved results. More information on the operating system and the programming of nonstop applications can be found in Bartlett [1978].

TABLE 8–4
Sample process-pair transactions

Step	Requester SeqNo = 0	Requester Backup SeqNo = 0	Server SeqNo = 0	Server Backup SeqNo = 0
1	Issue request to write record →			
2			If SeqNo < MySeqNo, then return saved status	
3			Otherwise, read disk, perform operation, checkpoint request →	Saves request
4			Write to disk SeqNo = 1 → checkpoint result	Saves result SeqNo = 1
5	←		Return results	
6	Checkpoint results →	SeqNo = 1		

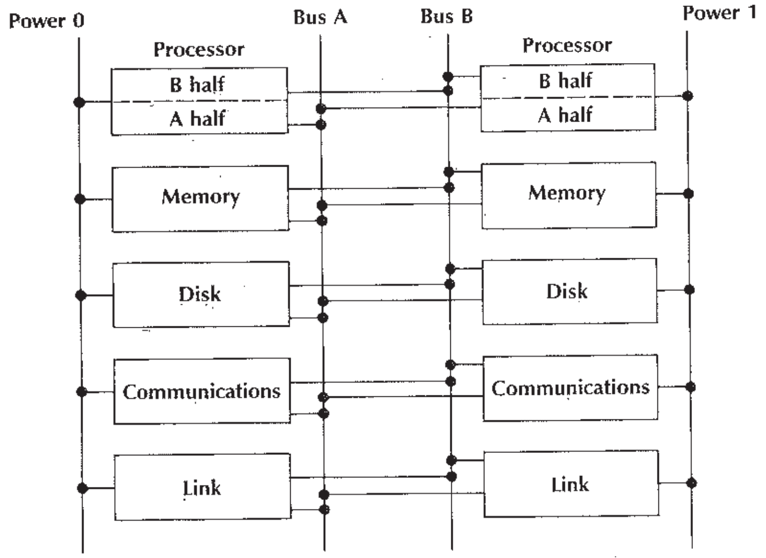
Source: Bartlett, 1981; © 1981 ACM.

STRATUS COMPUTERS, INC.

Whereas the Tandem architecture was based upon minicomputer technology, Stratus entered the OLTP market five years after Tandem by harnessing microprocessors. By 1980, the performance of microprocessor chips was beginning to rival that of minicomputers. Because of the smaller form factor of microprocessor chips, it was possible to place two microprocessors on a single board and to compare their output pins on every clock cycle. Thus, the Stratus system appears to users as a conventional system that does not require special software for error detection and recovery. The case study by Steven Webber describes the Stratus approach in detail.

The design goal for Stratus systems is continuous processing, which is defined as uninterrupted operation without loss of data, performance degradation, or special programming. The Stratus self-checking, duplicate-and-match architecture is shown in Figure 8–3. A module (or computer) is composed of replicated power and backplane buses (StrataBus) into which a variety of boards can be inserted. Boards are logically divided into halves that drive outputs to and receive inputs from both buses. The bus drivers/receivers are duplicated and controlled independently. The logical halves are driven in lock-step by the same clock. A comparitor is used to detect any disagreements between the two halves of the board. Multiple failures that affect the two independent halves of a board could cause the module to hang as it alternated between buses seeking a fault-free path. Up to 32 modules can be interconnected into a system via a message-passing Stratus intermodule bus (SIB). Access to the SIB is by dual 14 megabyte-per-second links. Systems, in return, are tied together by an X.25 packet-switched network.

FIGURE 8–3
The Stratus pair-and-spare architecture



Now consider how the system in Figure 8–3 tolerates failure. The two processor boards (each containing a pair of microprocessors), each self-checking modules, are used in a pair-and-spare configuration. Each board operates independently. Each half of each board (for example, side A) received inputs from a different bus (for example, bus A) and drives a different bus (for example, bus A). Each bus is the wired-OR of one-half of each board (for example, bus A is the wired-OR of all A board halves). The boards constantly compare their two halves, and upon disagreement, the board removes itself from service, a maintenance interrupt is generated, and a red light is illuminated. The spare pair on the other processor board continues processing and is now the sole driver of both buses. The operating system executes a diagnostic on the failed board to determine whether the error was caused by a transient or permanent fault. In the case of a transient, the board is returned to service. Permanent faults are reported by phone to the Stratus Customer Assistance Center (CAC). The CAC reconfirms the problem, selects a replacement board of the same revision, prints installation instructions, and ships the board by overnight courier. The first time the user realizes there is a problem is when the board is delivered. The user removes the old board and inserts the new board without disrupting the system (that is, makes a "hot" swap). The new board interrupts the system, and the processor that has been running brings the replacement into full synchronization, at which point the full configuration is available again. Detection and recovery are transparent to the application software.

The detection and recovery procedures for other system components are similar, although the full implementation of pair-and-spare is restricted to only the processor and memory. The disk controllers contain duplicate read/write circuitry. Communications controllers are also self-checking. In addition, the memory controllers monitor the bus for parity errors. The controllers can declare a bus broken and instruct all boards to stop using that bus. Other boards monitor the bus for data directed to them. If the board detects an inconsistency but the memory controllers have not declared the bus broken, the board assumes that its bus receivers have failed and declares itself failed.

The Stratus hardware approach is attractive in that it does not require on-line recovery from faults. The spare component continues processing until its fault counterpart can be replaced. No data errors are injected into the system; hence, no software recovery mechanisms are required for the pair-and-spare components. Complexities caused by checkpointing/restart programming and other software fault-tolerant considerations are eliminated. In addition to ease in programming, the Stratus approach to maintenance reduces the yearly service cost to 6 percent of life-cycle cost, as compared to an industrial average of 9 percent.

REFERENCES

Bartlett, 1978, 1981.

THE AT&T CASE

Part I: Fault-Tolerant Design of AT&T Telephone Switching System Processors

W.N. TOY

INTRODUCTION

Except for computer systems used in space-borne vehicles and U.S. defense installations, no other application has a higher availability requirement than a stored-program–controlled (SPC) telecommunications switching system. SPC systems have been designed to be out of service no more than a few minutes per year. Furthermore, design objectives permit no more than 0.01 percent of the telephone calls to be processed incorrectly [Downing, Nowak, and Tuomenoksa, 1964]. For example, when a fault occurs in a system, few calls in progress may be handled incorrectly during the recovery process.

At the core of every system is a single high-speed central processor [Harr, Taylor, and Ulrich, 1969; Browne et al., 1969; Staehler, 1977]. To establish an ultrareliable switching environment, redundancy of system components, including duplication of the processor itself, is the approach taken to compensate for potential machine faults. Without this redundancy, a single component failure in the processor might cause a complete failure of the entire system. With duplication, a standby processor takes over control and provides continuous telephone service.

When the system fails, the fault must be quickly detected and isolated. Meanwhile, a rapid recovery of the call processing functions (by the redundant component(s) and/or processor) is necessary to maintain the system's high availability. Next, the fault must be diagnosed and the defective unit repaired or replaced. The failure rate and repair time must be such that the probability is very small for a failure to occur in the duplicate unit before the first unit is repaired.

ALLOCATION AND CAUSES OF SYSTEM DOWNTIME

The outage of a telephone (switching) office can be caused by facilities other than the processor. While a hardware fault in one of the peripheral units generally results in only a partial loss of service, it is possible for a fault in this area to bring the entire system down. By design, the processor has been allocated two-thirds of the system downtime. The other one-third is allocated to the remaining equipment in the system.

Field experience indicates that system outages due to the processor may be assigned to one of four categories, as shown in Figure 8–4 [Staehler and Watters, 1976]. The percentages in this figure represent the fraction of total downtime attributable to each cause. The four categories are as follows.

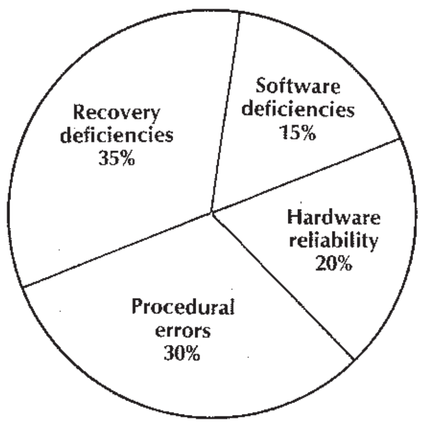
• *Hardware Reliability*: Before the accumulation of large amounts of field data, total system downtime was usually assigned to hardware. We now know that the situation is more complex. Processor hardware actually accounts for only 20 percent of the downtime. With growing use of stored program control, it has become increasingly important to make such systems more reliable. Redundancy is designed into all subsystems so that the system can go down *only* when a hardware failure occurs simultaneously in a unit and its duplicate. However, the data now show that good diagnostic and trouble location programs are also very critical parts of the total system reliability performance.

• *Software Deficiencies*: Software deficiencies include all software errors that cause memory mutilation and program loops that can only be cleared by major reinitialization. Software faults are the result of improper translation or implementation of the original algorithm. In some cases, the original algorithm may have been incorrectly specified. Program changes and feature additions are continuously incorporated into working offices. Software accounts for 15 percent of the downtime.

• *Recovery Deficiencies*: Recovery is the system's most complex and difficult function. Deficiencies may include the shortcomings of either hardware or software design to detect faults when they occur. When the faults go undetected, the system remains extensively impaired until the trouble is recognized. A recovery problem can also occur if the system is unable to properly isolate a faulty subsystem and configure a working system around it.

The many possible system states that may arise under trouble conditions make recovery a complicated process. Besides those problems already mentioned, unforeseen difficulties may be

FIGURE 8–4
System outage allocation



encountered in the field and lead to inadequate recovery. Because of the large number of variables involved and because the recovery function is so strongly related to all other components of maintenance, recovery deficiencies account for 35 percent of the downtime.

• *Procedural Errors*: Human error on the part of maintenance personnel or office administrators can also cause the system to go down. For example, someone in maintenance may mistakenly pull a circuit pack from the on-line processor while repairing a defective standby processor. Inadequate and incorrect documentation (for example, user's manuals) may also be classified as human error. Obviously, the number of manual operations must be reduced if procedural errors are to be minimized. Procedural errors account for 30 percent of the downtime.

The shortcomings and deficiencies of current systems are being continually corrected to improve system reliability.

DUPLEX ARCHITECTURE

When a fault occurs in a nonredundant single processor, the system will remain down until the processor is repaired. In order to meet reliability requirements, *redundancy* is included in the system design, and continuous, correct operation is maintained by duplicating all functional units within the processor. If one of the units fails, the duplicated unit is switched in, maintaining continuous operation. Meanwhile, the defective unit is repaired. Should a fault occur in the duplicated unit during the repair interval, the system will, of course, go down. If the repair interval is relatively short, the probability of simultaneous faults occurring in two identical units is quite small. This technique of redundancy has been used throughout each AT&T switching system.

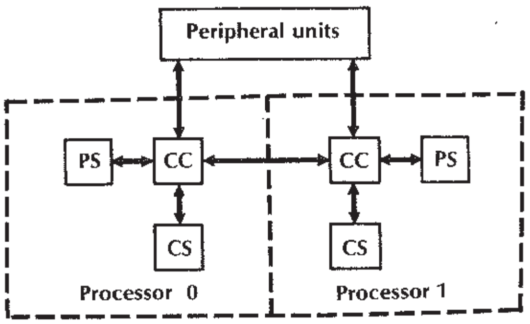
The first-generation electronic switching system (ESS) processor structure consists of two store communities: program store and call store. The program store is a read-only memory, containing the call processing, maintenance, and administration programs; it also contains long-term translation and system parameters. The call store contains the transient data related to telephone calls in progress. The memory is electrically alterable to allow its data to be changed frequently. In one particular arrangement, shown in Figure 8–5b, the complete processor is treated as a single functional block and is duplicated. This type of single-unit duplex system has two possible configurations: Either processor 0 or processor 1 can be assigned as the on-line working system, while the other unit serves as a standby backup. The mean time to failure (MTTF), a measure of reliability, is given by the following expression [Smith, 1972]:

$$\text{MTTF} = \frac{\mu}{2\lambda^2}$$

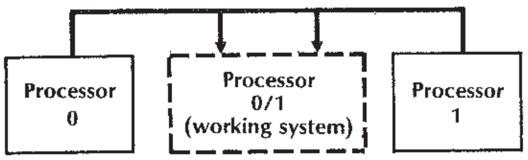
where μ = repair rate (reciprocal of the repair time)
λ = failure rate

The failure rate (λ) of one unit is the sum of the failure rates of all components within the unit. For medium and small ESS processors, Figure 8–5a shows a system structure containing several functional units that are treated as a single entity, with λ still sufficiently small to meet the reliability requirement. The single-unit duplex configuration has the advantage of being very simple in terms of the number of switching blocks in the system. This configuration simplifies not only the recovery program but also the hardware interconnection by eliminating the additional access required to make each duplicated block capable of switching independently into the on-line system configuration.

In the large 1ES switching system, which contains many components, the MTTF becomes



a. Processor structure



b. Two possible configurations

FIGURE 8–5
Single-unit duplex configuration

too low to meet the reliability requirement. In order to increase the MTTF, either the number of components (failure rate) or the repair time must be reduced. Alternatively, the single-unit duplex configuration can be partitioned into a multi-unit duplex configuration, as shown in Figure 8–6. In this arrangement, each subunit contains a small number of components and can be switched into a working system. The system will fail only if a fault occurs in the redundant subunit while the original is being repaired. Since each subunit contains fewer components, the probability of two simultaneous faults occurring in a duplicated pair of subunits is reduced. The MTTF of the multi-unit duplex configuration can be computed by considering the conditional probability of the failure of a duplicate subunit during the repair time of the original subunit.

An example of a multi-unit duplex configuration is shown in Figure 8–6. A working system is configured with a fault-free CC*x*-CS*x*-CSB*x*-PS*x*-PSB*x*-PUB*x* arrangement, where *x* is either subunit 0 or subunit 1. This arrangement means there are 2^6, or 64, possible combinations of system configurations. The MTTF is given by the following expression:

$$\text{MTTF} = \frac{r\mu}{2\lambda^2} \tag{1}$$

and

$$r = \frac{1}{(\lambda_{CC}/\lambda)^2 + (\lambda_{CS}/\lambda)^2 + (\lambda_{CSB}/\lambda)^2 + (\lambda_{PS}/\lambda)^2 + (\lambda_{PSB}/\lambda)^2 + (\lambda_{PUB}/\lambda)^2} \tag{2}$$

The factor *r* is at a maximum when the failure rate (λ_i) for each subunit is the same. In this case,

$$\lambda_{CC} = \lambda_{CS} = \lambda_{CSB} = \lambda_{PS} = \lambda_{PSB} = \lambda_{PUB} = \lambda_i \tag{3}$$

or

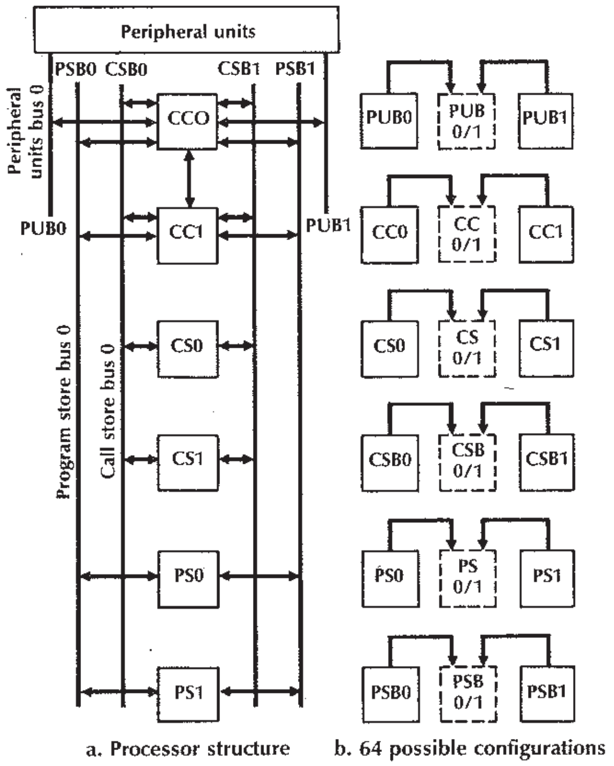
$$\lambda_i = \frac{\lambda}{s} \tag{4}$$

where s = number of subunits in Eq. (2)
$s = 6$
$r = s$

At best, the MTTF is improved by a factor corresponding to the number of partitioned subunits. This improvement is not fully realized, since equipment must be added to provide additional access and to select subunits. Partitioning the subsystem into subunits, as shown in Figure 8–6, results in subunits of different sizes. Again, the failure rate for each individual subunit will not be the same; hence, the r-factor will be smaller than 6. Because of the relatively large number of components used in implementing the 1ESS switch processor, the system is arranged in the multi-unit duplex configuration in order to meet the reliability requirement.

Reliability calculation is a process of predicting, from available failure rate data, the achievable reliability of a system and the probability of meeting the reliability objectives for telephone switching applications. These calculations are most useful and beneficial during the early stages of design in order to assess various types of redundancy and to determine the system's organi-

FIGURE 8–6
Multi-unit duplex configuration



zation. In the small and medium switches, the calculations support the use of single-unit duplex structures. For large systems, it is necessary to partition the system into a multi-unit duplex configuration.

FAULT SIMULATION TECHNIQUES

One of the more difficult tasks of maintenance design is fault diagnosis. The maintenance design's effectiveness in diagnostic resolution can be determined by simulation of the system's behavior in the presence of a specific fault. By means of simulation, design deficiencies can be identified and corrected prior to any system's deployment in the field. It is necessary to evaluate the system's ability to detect faults, to recover automatically back into a working system, and to provide diagnostics information when the fault is within a few replaceable circuit packs. Fault simulation, therefore, is an important aspect of maintenance design.

There are essentially two techniques used for simulating faults of digital systems: physical simulation and digital simulation. Physical simulation is a process of inserting faults into a physical working model. This method produces more realistic behavior under fault conditions than digital simulation does. A wider class of faults can be applied to the system, such as a blown fuse or shorted backplane interconnection. However, fault simulation cannot begin until the design has been completed and the equipment is fully operational. Also, it is not possible to insert faults that are internal to an integrated circuit.

Digital fault simulation is a means of predicting the behavior under failure of a processor modeled in a computer program. The computer used to execute the program (the host) is generally different from the processor that is being simulated (the object). Digital fault simulation gives a high degree of automation and excellent access to interior points of logic to monitor the signal flow. It allows diagnostic test development and evaluation to proceed well in advance of unit fabrication. The cost of computer simulation can be quite high for a large, complex system.

The physical fault simulation method was first employed to generate diagnostic data for the Morris Electronic Switching System [Tsiang and Ulrich, 1962]. Over 50,000 known faults were purposely introduced into the central control to be diagnosed by its diagnostic program. Test results associated with each fault were recorded. They were then sorted and printed in dictionary format to formulate a trouble-locating manual. Under trouble conditions, by consulting the manual, it was possible to determine a set of several suspected circuit packs that might contain the defective component. Use of the dictionary technique at the Morris system kept the average repair time low and made maintenance much easier.

The experience gained in the physical fault simulation was applied and extended in the 1ESS switch development [Downing, Nowak, and Tuomenoksa, 1964]. Each plug-in circuit pack was replaced by a fault simulator that introduced every possible type of single fault on the replaced package one at a time and then recorded the system reaction on magnetic tape. This procedure was followed for all circuit packs in the system. In addition to diagnostic data for dictionaries, additional data were collected to determine the adequacy of hardware and software in fault detection and system recovery. Deficiencies were corrected to improve the overall maintenance of the system.

A digital logic simulator, called LAMP [Chang, Smith, and Walford, 1974], was developed for the 1A system, and it played an important role in the hardware and diagnostics development of the 1A Processor. LAMP is capable of simulating a subsystem with as many as 65,000 logic gates. All classical faults for standard logic gates are simulatable with logic nodes stuck-at-0 or stuck-at-1. Before physical units are available, digital simulators can be very effective in verifying the design, evaluating diagnostic access, and developing tests. Physical fault simulation has been demonstrated in the System 1 processor to give a very realistic behavior under fault conditions. The integration of both techniques was employed in the development of the 1A processor to take advantage of both processes. The use of complementary simulation allows faults to be simulated physically (in the system laboratory) and logically (on a computer). Most of the deficiencies of one simulation process are compensated for by the other. The complementary method provides both a convenient method for validating the results and more extensive fault simulation data than is possible if either process is used individually. Figure 8–7 shows the complementary process of fault simulation used in the 1A Processor development [Bowman et al., 1977; Goetz, 1974]. Maximum diagnostic performance was achieved from an integrated use of both simulation methods.

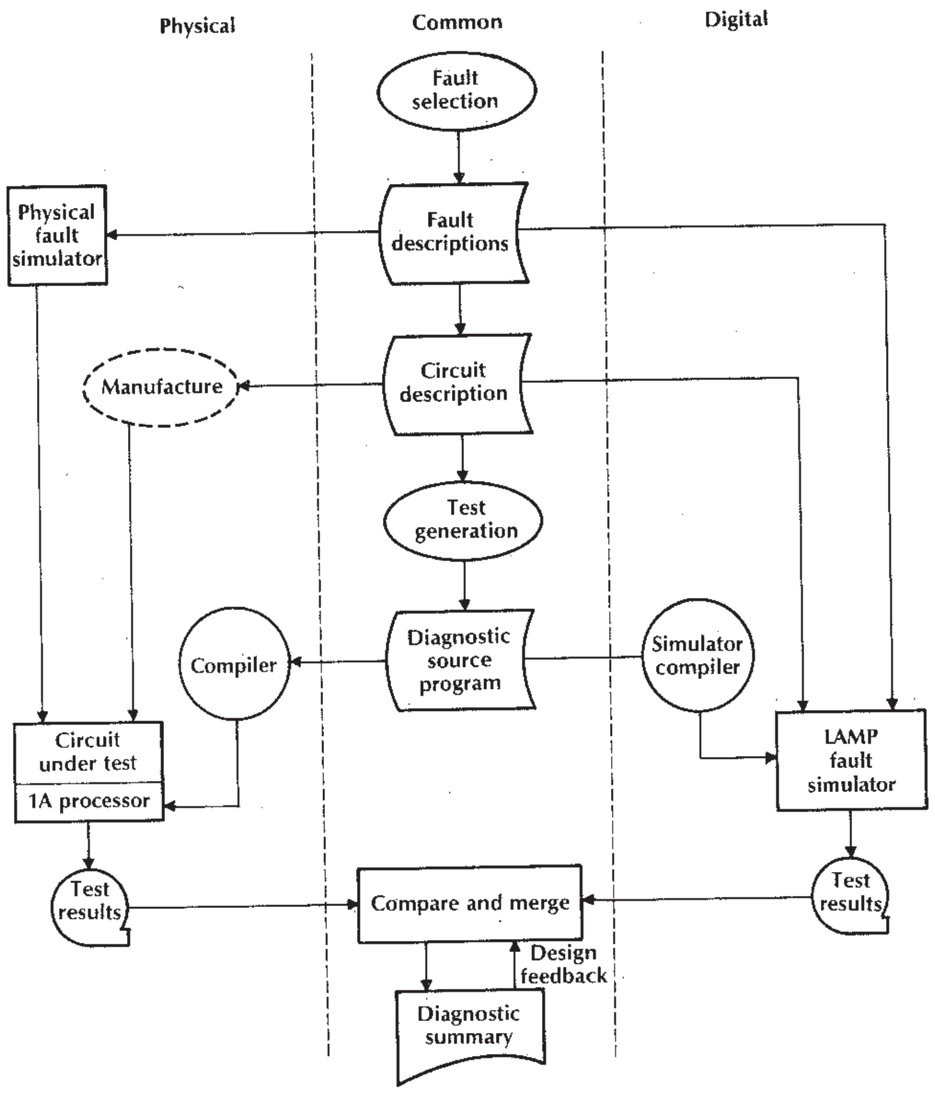


FIGURE 8–7 *Complementary fault simulation system*

FIRST-GENERATION ESS PROCESSORS

The world's first stored-program–controlled switching system provided commercial telephone service at Morris, Illinois, in 1959 for about a year on a field-trial basis [Keister, Ketchledge, and Lovell, 1960]. The system demonstrated the use of stored program control and the basic maintenance philosophy of providing continuous and reliable telephone service. The trial established valuable guides for designing a successor, the 1ESS switch.

1ESS Switch Processor (No. 1 Processor)

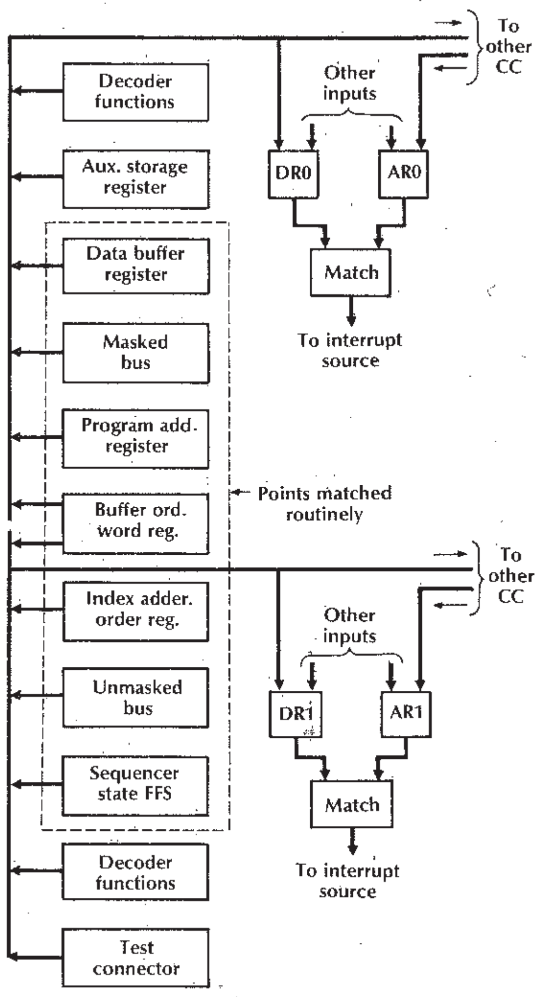
The 1ESS switching system was designed to serve large metropolitan telephone offices, ranging from several thousand to 65,000 lines [Keister, Ketchledge, and Vaughan, 1964]. As in most large switching systems, the processor represents only a small percentage of the total system cost. Therefore, performance and reliability were of primary importance in the design of the No. 1 processor; cost was secondary. In order to meet the reliability standards established by electromechanical systems, all units essential to proper operation of the office are duplicated (see Figure 8–6). The multi-unit duplex configuration was necessary to increase the MTTF of the processor because of the large number of components in each of the functional blocks.

Even with duplication, troubles must be found and corrected quickly to minimize exposure to system failure due to multiple troubles. All units are monitored continually so that troubles in the standby units are found just as quickly as those in the on-line units. Monitoring is accomplished by running the on-line and standby units in synchronous and match mode of operation [Downing, Nowak, and Tuomenoksa, 1964]. Synchronization requires that clock timing signals be in close tolerance so that every operation in both halves is performed in step, and key outputs are compared for error detection. The synchronization of duplicated units is accomplished by having the on-line oscillator output drive both clock circuits. There are two match circuits in each central control (CC). Each matcher compares 24 bits within one machine cycle of 5.5 microseconds. Figure 8–8 shows that each matcher has access to six sets of internal nodes (24 bits per node). In the routine match mode, the points matched in each cycle are dependent upon the instruction that is being executed. The selected match points are those most pertinent to the data processing steps that occur during a given machine cycle. The two matchers in each CC compare the same sets of selected test points. If a mismatch occurs, an interrupt is generated, causing the fault-recognition program to run. The basic function of this program is to determine which half of the system is faulty. The suspected unit is removed from service, and the appropriate diagnostic program is run to pinpoint the defective circuit pack. The capability of each CC to compare a number of internal nodes provides a highly effective means of detecting hardware errors.

The No. 1 Processor was designed during the discrete component era (early 1960s), using individual components to implement logic gates [Cagle et al., 1964]. The CC contains approximately 12,000 logic gates. Although this number appears small when compared to large-scale integration (LSI) technology, the No. 1 Processor was a physically large machine for its time. The match circuits capable of comparing internal nodes are the primary tools incorporated into the CC for diagnosing as well as detecting troubles. Specified information can be sampled by the matchers and retained in the match registers for examination. This mode of operation obtains critical data during the execution of diagnostic programs.

The early program store used permanent magnet twister (PMT) modules as basic storage elements [Ault et al., 1964]. PMTs are a form of ROM in which system failures cannot alter the information content. Experience gained from the Morris field test system, which used the less reliable flying spot store, indicated that Hamming correction code was highly effective in providing continuous operation. At the time of development, it was felt that PMT modules might not be

FIGURE 8–8
No. 1 Processor's CC match access



reliable enough. Consequently, the program store word included additional check bits for single-bit error correction (Hamming code). In addition, an overall parity check bit that covers both the data and their addresses is included in the word. The word size consists of 37 bits of information and seven check bits. When an error is corrected during normal operation, it is logged in an error counter. Also, detection of a single error in the address or a double error in the word will cause an automatic retry.

The call store is the temporary read and write memory for storing transient data associated with call processing. Ferrite sheet memory modules are the basic storage elements used in implementing the call store in the 1ESS switch [Genke, Harding, and Staehler, 1964]. The call store used in most No. 1 offices is smaller than the program store. (At the time of design, the cost per bit of call store was considerably higher than that of program store.) Also, ferrite sheet memory modules were considered to be very reliable devices. Consequently, single-bit error detection rather than Hamming correction code was provided in the call store.

There are two parity check bits: one over both the address and the data; and the other over the address only. Again, as in the program store, automatic retry is performed whenever an error is detected, and the event is logged in an error counter for diagnostic use.

Troubles are normally detected by fault-detection circuits, and error-free system operation is recovered by fault recognition programs [Downing, Nowak, and Tuomenoksa, 1964]. This requires the on-line processor to be capable of making a proper decision. If this is not possible, an emergency action timer will time out and activate special circuits to establish various combinations of subsystems into a system configuration. A special program that is used to determine whether or not the assembled processor is sane takes the processor through a series of tests arranged in a maze. Only one correct path through the maze exists. If the processor passes through successfully, the timer will be reset, and recovery is successful. If recovery is unsuccessful, the timer will time out again, and the rearrangement of subsystems will be tried one at a time (for example, combination of CC, program store, and program store bus systems). For each selected combination, the special sanity program is started and the sanity timer is activated. This procedure is repeated until a working configuration is found. The sanity program and sanity timer determine if the on-line CC is functioning properly. The active CC includes the program store and the program store bus.

2ESS Switch Processor (No. 2 Processor)

The No. 2 Processor was developed during the mid-1960s [Spencer and Vigilante, 1969]. The 2ESS switch was designed for medium-sized offices ranging from 1,000 to 10,000 lines. The processor's design was derived from experience with the common stored program of a private branch exchange (PBX), the No. 101 [Seley and Vigilante, 1964]. Since the capacity requirement of the 2ESS switch was to be less than that of the 1ESS switch, cost became one of the more important design considerations. (Reliability is equally important in all systems.) The 2ESS switch contains much less hardware than the 1ESS switch. Understandably, its component failure rate is also substantially lower. Its CC contains approximately 5000 gates (discrete components). To reduce cost and increase reliability, resistor-transistor logic (RTL) gates were chosen for the 2ESS processor, since resistors are less expensive and more reliable than diodes [the No. 1 Processor used diode-transistor logic (DTL)].

Because the No. 2 Processor's CC, program store, and call store are smaller, they are grouped together as a single switchable block in the single-unit duplex configuration shown in Figure 8–5. Calculations indicate that its MTTF is approximately the same as the No. 1 multi-unit duplex structure, with each of the functional blocks and associated store buses grouped together as a switchable block. The use of only two subsystem configurations considerably reduces the amount of hardware needed to provide gating paths and control for each functional unit. Moreover, the recovery program is simplified, and the reliability of the system is improved.

The No. 2 Processor runs in the synchronous and match mode of operation [Beuscher et al., 1969]. The on-line oscillator output drives both clock circuits in order to keep the timing synchronized. The match operation is not as extensive as it is in the No. 1 Processor. For simplicity, there is only one matcher in the No. 2 Processor; it is located in the nonduplicated maintenance center (see Figure 8–9). The matcher always compares the call store input register in the two CCs when call store operations are performed synchronously. A fault in almost any part of either CC quickly results in a call store input register mismatch. This mismatch occurs because almost all data manipulation performed in both the program control and the input-output control involves

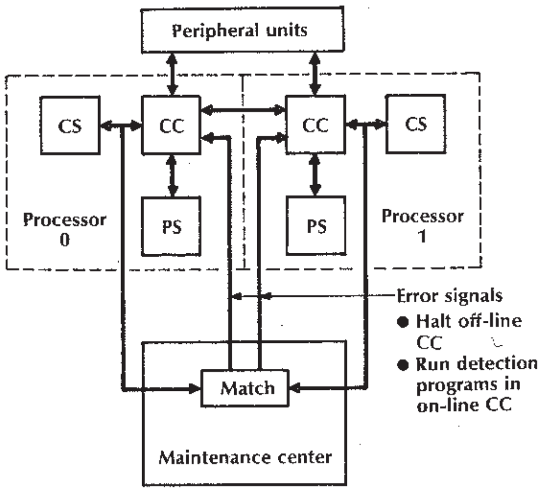


FIGURE 8–9
No. 2 Processor's CC match access

processed data returning to the call store. The call store input is the central point by which data eventually funnel through to the call store. By matching the call store inputs, an effective check of the system equipment is provided. Compared to the more complex matching of the No. 1 Processor, error detection in the No. 2 Processor may not be as instantaneous, since only one crucial node in the processor is matched. Certain faults in the No. 2 Processor will go undetected until the errors propagate into the call store. This interval is probably no more than tens or hundreds of microseconds. During such a short interval, the fault would affect only a single call.

The No. 2 Processor matcher is not used as a diagnostic tool as is the matcher in the No. 1 Processor. Therefore, additional detection hardware is designed into the No. 2 Processor to help diagnose as well as detect faults. When a mismatch occurs, the detection program is run in the on-line CC to determine if it contains the fault. This is done while the standby processor is disabled. If a solid fault in the on-line processor is detected by the mismatch detection program, the control is automatically passed to the standby processor, causing it to become the on-line processor. The faulty processor is disabled and diagnostic tests are called in to pinpoint the defective circuit pack.

The program store also uses PMT modules as basic storage elements, with a word size of 22 bits, half the width of the No. 1 Processor's word size. Experience gained in the design and operation of the No. 101 PBX showed that PMT stores are very reliable. The additional protection provided in the No. 1 Processor against memory faults by error correction was not considered to be as important in the No. 2 Processor. Thus, the need to keep the cost down led to the choice of error detection *only*, instead of the more sophisticated Hamming correction code.

Error detection works as follows: One of the 22 bits in a word is allocated as a parity check bit. The program store contains both program and translation data. Additional protection is provided by using odd parity for program words and even parity for translation data. This parity scheme detects the possibility of accessing the translation data area of memory as instruction words. For example, a software error may cause the program to branch into the data section of the memory and execute the data words as instruction words. The parity check would detect this problem immediately. The program store includes checking circuits to detect multiple-word access. Under program control, the sense amplifier threshold voltage can be varied in two discrete amounts from its nominal value to obtain a measure of the operating margin. The use of parity check was the proper choice for the No. 2 Processor in view of the high reliability of these memory devices.

The No. 2 Processor call store uses the same ferrite sheet memory modules as the No. 1 Processor. However, the No. 2 Processor's data word is 16 bits wide instead of 24. Fault detection depends heavily upon the matching of the call store inputs when the duplex processors run in the synchronous mode. Within the call store circuit, the access circuitry is checked to see that access currents flow in the right direction at the correct time and that only two access switches are selected in any store operation, ensuring that only one word is accessed in the memory operation. Similarly, threshold voltages of the sense amplifiers may be varied under program control to evaluate the operating margins of the store. No parity check bit is provided in the call store.

Each processor contains a program timer that is designed to back up other detection methods. Normally, the on-line processor clears the timer in both processors at prescribed intervals if the basic call processing program cycles correctly. If, however, a hardware or software trouble condition exists (for example, a program may go astray or a longer program loop may prevent the timer from being cleared), the timer will time out and automatically produce a switch. The new on-line processor is automatically forced to run an initialization restart program that attempts to establish a working system. System recovery is simplified by using two possible system configurations rather than the multi-unit duplex system.

SECOND-GENERATION PROCESSORS

The advent of silicon integrated circuits (ICs) in the mid-1960s provided the technological climate for dramatic miniaturization, improved performance, and cost-reduced hardware. The term *1A technology* refers to the standard set of IC devices, apparatus, and design tools that were used to design the No. 1A Processor and the No. 3A Processor [Becker et al., 1977]. The choice of technology and the scale of integration level were dictated by the technological advances made between 1968 and 1970. Small-scale integration (SSI), made possible by bipolar technology, was capable of high yield production. Because of the processor cycle time, high-speed logic gates with propagation delays from 5–10 nanoseconds were designed and developed concurrent with the No. 1A Processor.

No. 1A Processor

The No. 1A Processor, successor to the No. 1 Processor, was designed primarily for the control of large local and toll switches with high processing capabilities (the 1A ESS and 4ESS switches, respectively) [Budlong et al., 1977]. An important objective in developing the 1A ESS switch was to maintain commonality with the 1ESS switch. High capacity was achieved by implementing the new 1A integrated technology and a newly designed system structure. These changes made possible an instruction execution rate that is four to eight times faster than the No. 1 Processor. Compatibility with the 1ESS system also allows the No. 1A Processor to be retrofitted into an in-service 1ESS, replacing the No. 1 Processor when additional capacity is needed. The first 1A Processor was put into service in January, 1976, as control for a 4ESS toll switch in Chicago. Less than one year later, the first 1A ESS system was put into commercial operation. By 1988, about 2000 systems were in service.

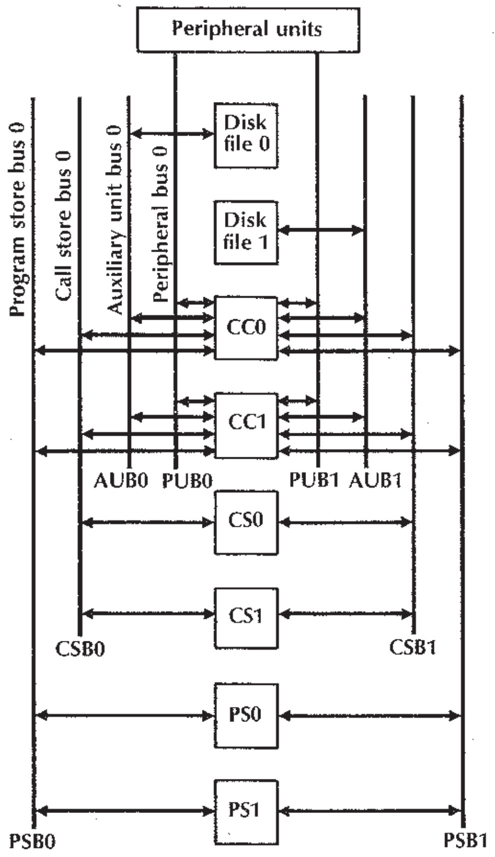
The No. 1A Processor architecture is similar to its predecessor in that all of its subsystems have redundant units and are connected to the basic CC via redundant bus systems [Bowman et al., 1977]. One of the No. 1A Processor's major architectural differences is its program store

[Ault et al., 1977]. It has a writable RAM instead of PMT ROM. By combining disk memory and RAM, the system has the same amount of memory as a system with PMT, but at a lower cost. Backup copy of program and translation data is kept on disk. Other programs (e.g., diagnostics) are brought to RAM as needed; the same RAM spare is shared among different programs. More important is the system's ability to change the content of the store quickly and automatically. This ability considerably simplifies the administration and updating of program and translation information in working offices.

The additional disk (file store) subsystem adds flexibility to the No. 1A Processor [Ault et al., 1977], but it also increases the complexity of system recovery. Figure 8–10 shows the multi-unit duplex No. 1A Processor. This configuration is similar to the No. 1 Processor arrangement (see Figure 8–6) with a duplicated file store included. The file store communicates with the program store or call store via the CC and the auxiliary unit bus. This communication allows direct memory access between the file store and the program store or the call store. The disk file and the auxiliary unit bus are grouped together as a switchable entity.

Error detection is achieved by the duplicated and matched synchronous mode of operation, as in the No. 1 Processor. Both CCs operate in step and perform identical operations. The

FIGURE 8–10
No. 1A Processor configuration



matching is done more extensively in the 1A to obtain as complete a check as possible. There are two match circuits in each processor. Each matcher has the ability to compare 24 internal bits to 24 bits in its mate once every machine cycle. (A machine cycle is 700 nanoseconds.) Any one of 16 different 24-bit internal nodes can be selected for comparison. The choice is determined by the type of instruction that is being executed. Rather than compare the same nodes in both CCs, the on-line and the standby CCs are arranged to match different sets of data. Four distinct internal groups are matched in the same machine cycle to ensure the correct execution of any instruction.

The No. 1A Processor design is an improvement of the No. 1 Processor design. The No. 1A Processor incorporates much more checking hardware throughout various functional units, in addition to matching hardware. Checking hardware speeds up fault detection and also aids the fault recovery process by providing indications that help isolate the faulty unit. The matching is used in various modes for maintenance purposes. This capability provides powerful diagnostic tools in isolating faults.

The program store and call store use the same hardware technology as in the No. 1 Processor. The CC contains approximately 50,000 logic gates. While the initial design of the stores called for core memories, they have been replaced with semiconductor dynamic MOS memories. The word size is 26 bits (24 data bits and 2 parity check bits). In the No. 1 Processor, the program store and the call store are fully duplicated. Because of their size, duplication requires a considerable amount of hardware, resulting in higher cost and increased component failures. To reduce the amount of hardware in the No. 1A Processor's store community, the memory is partitioned into blocks of 64K words, as shown in Figure 8–11. Two additional store blocks are provided as roving spares. If one of the program stores fails, a roving program store spare is substituted, and a copy of the program in the file store is transferred to the program store replacement. This type of redundancy has been made possible by the ability to regenerate data stored in a failing unit. Since a program store can be reloaded from the file store in less than a second, a roving spare redundancy plan is sufficient to meet the reliability requirement. As a result, Hamming correction code was not adopted in the No. 1A program store. However, it is essential that an error be detected quickly. Two parity check bits are generated over a partially overlapped, interleaved set of data bits and address. This overlapping is arranged to cope with particular memory circuit failures that may affect more than one bit of a word.

The 1A call stores contain both translation data backed up on the file stores and call-related transient data that are difficult to regenerate. The roving spare concept is expanded for the call stores to include sufficient spares to provide full duplication of transient data. If a fault occurs in a store that contains translation data, one of the duplicated stores containing transient call data is preempted and loaded with the necessary translation data from the duplicate in the file store. A parity check is done in the same manner as in the program store, using two check bits.

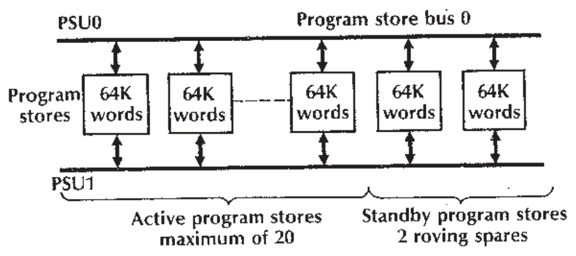


FIGURE 8–11 *No. 1A Processor's program store structure*

The combination of writable program store and file store provides a very effective and flexible system architecture for administrating and implementing a wide variety of features that are difficult to obtain in the 1ESS system. However, this architecture also complicates the process of fault recognition and recovery. Reconfiguration into a working system under trouble conditions is an extensive task, depending on the severity of the fault. (For example, it is possible for the processor to lose its sanity or ability to make proper decisions.) An autonomous hardware processor configuration (PC) circuit is provided in each CC to assist in assembling a working system. The PC circuit consists of various timers that ensure that the operational, fault recovery, and configuration programs are successfully executed. If these programs *are not* executed, the PC circuit controls the CC-to-program memory configuration, reloading program memory from file store when required, and isolating various subsystems from the CC until a working system is obtained.

No. 3A Processor

The No. 3A Processor was designed to control the small 3ESS switch [Irland and Stagg, 1974], which can handle from 500 to 5,000 lines. One of the major concerns in the design of this ESS was the cost of its processor. The low cost and high speed of integrated logic circuitry made it possible to design a cost-effective processor that performed better than its discrete component predecessor, the No. 2 Processor. The No. 3A project was started in early 1971. The first system cut into commercial service in late 1975.

Because the number of components in the No. 3A Processor is considerably fewer than in the No. 1A Processor, all subsystems are fully duplicated, including the main store. The CC, the store bus, and the store are treated as a single switchable entity, rather than individual switchable units as in the No. 1A Processor. The system structure is similar to the 2ESS switch. Experience gained in the design and operation of the No. 2 provided valuable input for the No. 3 Processor design.

The 3A design makes one major departure from previous processor designs: It operates in the nonmatched mode of duplex operation. The primary purpose of matching is to detect errors. A mismatch, however, does not indicate *where* (in which one of the processors) the fault has occurred. A diagnostic fault-location program must be run to localize the trouble so that the defective unit can be taken off line. For this reason, the No. 3A Processor was designed to be self-checking, with detection circuitry incorporated as an integral part of the processor. Faults occurring during normal operation are discovered quickly by detecting hardware. Detection circuitry eliminates the need to run the standby system in the synchronous and match mode of operation or the need to run the fault recognition program to identify the defective unit when a mismatch occurs.

The synchronous and match mode arrangement of the No. 1 Processor and the No. 2 Processor provides excellent detection and coverage of faults. However, there are many instances (for example, periodic diagnostics, administration changes, recent change updates, and so on) when the system is not run in the normal match mode. Consequently, during these periods, the system is vulnerable to faults that may go undetected. The rapid advances in integrated circuit technology make possible the implementation of self-checking circuits in a cost-effective manner. Self-checking circuits eliminate the need for the synchronous and match mode of operation.

Another new feature in switching system processor design is the application of the micro-program technique in the No. 3A [Storey, 1976]. This technique provides a regular procedure of implementing the control logic. Standard error detection is made part of the hardware to achieve a high degree of checkability. Sequential logic, which is difficult to check, is easily implemented as a sequence of microprogram steps. Microprogramming offers many attractive features: It is simple, flexible, easy to maintain, and easy to expand.

The No. 3A Processor paralleled the design of the No. 1A Processor in its use of an electrically alterable (writable) memory. However, great strides in semiconductor memory technology after the No. 1A became operational permitted the use of semiconductor memory, rather than the core memory, in the 3A.

The 3A's call store and program store are consolidated into a single store system. This consolidation reduces cost by eliminating buses, drivers, registers, and controls. A single store system no longer allows concurrent access of call store and program store. However, this disadvantage is more than compensated for by the much faster semiconductor memory. Its access time is 1 microsecond (the earlier PMT stores had an access time of 6 microseconds).

Normal operation requires the on-line processor to run and process calls while the standby processor is in the halt state, with its memory updated for each write operation. For the read operation, only the on-line memory is read, *except* when a parity error occurs during a memory read. A parity error results in a microprogram interrupt, which reads the word from the standby store in an attempt to bypass the error.

As discussed previously, the No. 2 Processor (first generation) is used in the 2ESS switch for medium-sized offices. It covers approximately 4,000 to 12,000 lines with a call handling capability of 19,000 busy-hour calls. (The number of calls is related to the calling rate of lines during the busy hour.) The microprogram technique used in the No. 3A Processor design allows the No. 2 Processor's instruction set to be emulated. This emulation enables programs written in the No. 2 assembly language to be directly portable to the No. 3A Processor. The ability to preserve the call processing programs permits the 2ESS system to be updated with the No. 3A Processor without having to undergo a complete new program development.

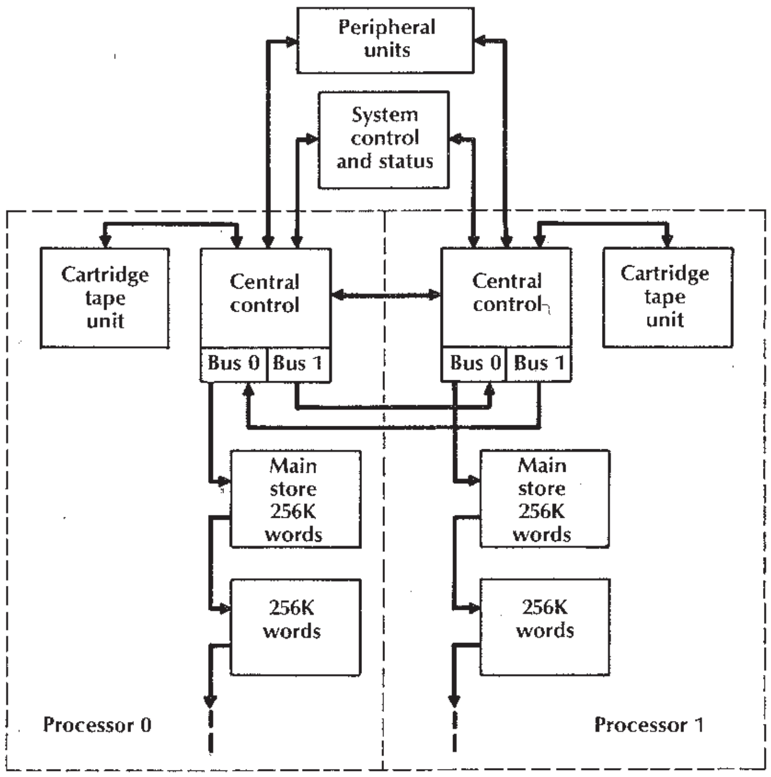
The combination of the No. 3A Processor and the peripheral equipment of the 2ESS system is designated as the 2B ESS switch. It is capable of handling 38,000 busy-hour calls, twice the capability of the 2ESS switch [Mandigo, 1976], and can be expanded to cover about 20,000 lines. Furthermore, when an existing 2ESS system in the field exceeds its real-time capacity, the No. 2 Processor can be taken out and replaced with the No. 3A Processor. The retrofit operation has been carried out successfully in working offices without disturbing telephone service.

Self-checking hardware has been integrated into the design to detect faults during normal system operation. This simplified fault recognition technique is required to identify a subsystem unit when it becomes defective. Reconfiguration into a working system is immediate, without extensive diagnostic programs to determine which subsystem unit contains the fault. The problem of synchronization, in a much shorter machine cycle (150 nanoseconds), is eliminated by not having to run both processors in step. The No. 3A Processor uses low-cost ICs to realize its highly reliable and flexible design.

General Systems Description. The general system block diagram of the No. 3A Processor is shown in Figure 8–12. The CC, the main store, and the cartridge tape unit are duplicated for reliability. These units are grouped as a single switchable entity rather than individual switchable units. The quantity of equipment within the switchable block is small enough to meet the reliability requirements; therefore, the expense and complexity of providing communication paths and control for switchable units within the system are avoided. Each functional unit was designed to be as autonomous as possible, with a minimum number of output signal leads. Such autonomy provides the flexibility necessary to expand the system and make changes easily.

As shown in Figure 8–12, the standard program store and call store are combined as a single storage unit to reduce cost. Although the processors are not run in the synchronous and match

FIGURE 8–12
No. 3A Processor organization



mode of operation, both stores (on-line and standby) are kept up to date by having the on-line processor write into both stores simultaneously when call store data are written or changed. Because of the volatile nature of a writable memory, low-cost bulk storage backup (cartridge tape) is required to reload the program and translation data when the data are lost due to a store failure. The pump-up mechanism, or store loader, uses the microprogram control in conjunction with an I/O serial channel to transfer data between the cartridge tape unit and the main store. Other deferrable, infrequently used programs (that is, diagnostics or growth programs) are stored on tape and paged in as needed.

The system control and status panel, a nonduplicated block, provides a common point for the display of overall system status and alarms. Included in this unit is the emergency action circuitry that allows the maintenance personnel to initialize the system or force and lock the system into a fixed configuration. Communication with the processor takes place via the I/O serial channel.

General Processor Description. Figure 8–13 shows a detailed block diagram of the CC. It is organized to process input data and handle call processing functions efficiently. The processor's design is based on the register type of architecture. Fast-access storage in the form of flip-flop registers provides short-term storage for information that is being used in current data processing operations. Sixteen general-purpose registers are provided as integral parts of the structure.

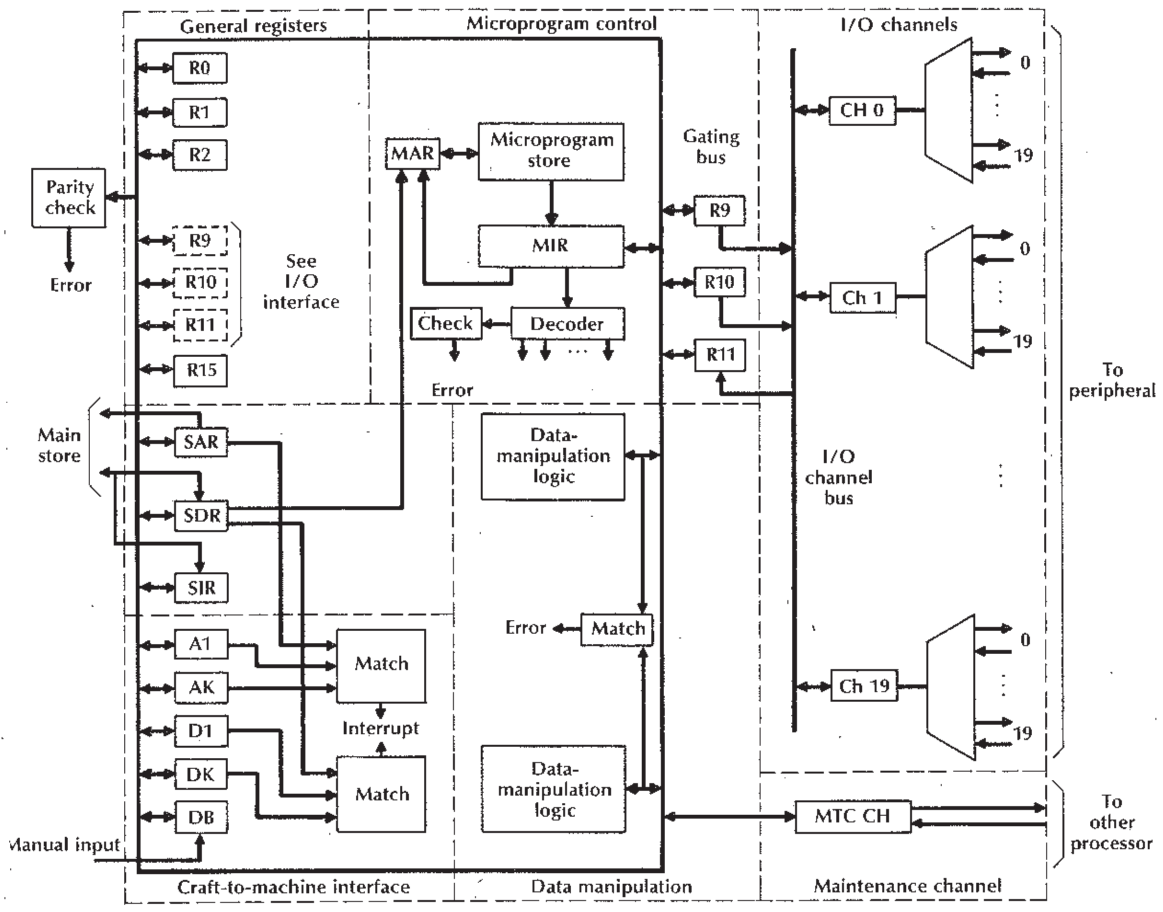


FIGURE 8–13 *No. 3A Processor's central control*

Microprogram control is the heart of the No. 3A Processor. It provides nearly all of the complex control and sequencing operations required for implementing the instruction set. Other complicated sequencing functions are also stored in the microprogram memory; for example, the bootstrap operation of reloading the program from the backup tape unit, the initializing sequence to restart the system under trouble conditions, the interrupt priority control and saving of essential registers, the emergency action timer and processor switching operation, and the craft-to-machine functions. The regular structure of the microprogram memory makes error detection easier. The microprogram method of implementation also offers flexibility in changing control functions.

The data manipulation instructions are designed specifically for implementing the call processing programs. These instructions are concerned with logical and bit manipulation rather than with arithmetical operations. However, a binary ADD is included in the instruction repertoire for adding two binary numbers and for indexing. This instruction allows other arithmetical operations to be implemented conveniently by the software combinations of addition and logical operations, or by a microprogram sequence if higher speed is essential. The data manipulation logic contains rotation, Boolean function of two variables, first zero detection, and fast binary ADD.

The remaining function blocks in Figure 8–13 deal with external interfaces. The 20 main I/O channels, each with 20 subchannels, allow the processor to control and access up to 400 peripheral units by means of 21-bit (16 data, 2 parity, and 3 start code bits) serial 6.67-MHz messages. The system is expandable in modules of one main channel (20 subchannels). The I/O structure allows up to 20 subchannels (one from each main channel) to be active simultaneously. In addition, the craft-to-machine interface, with displays and manual inputs, is integrated into the processor. This interface contains many of the manual functions that will assist in hardware and software debugging. The control logic associated with this part of the processor is incorporated as part of the microprogram control. Lastly, the maintenance channel enables the online processor to control and diagnose the standby processor. The use of a serial channel reduces the number of leads interconnecting the two processors and causes them to be loosely coupled. This loose coupling facilitates the split mode or stand-alone configuration for factory test or system test.

Hardware Implementation. Maintenance has been made an integral part of the 3A CC design. It uses the standard 1A ESS logic family with its associated packaging technology [Becker et al., 1977]. Up to 52 silicon integrated circuit chips (SICs), each containing from 4 to 10 logic gates, can be packed on a 3.25" × 4.00" 1A ceramic substrate. The substrate is mounted on a 3.67" × 7" circuit board with an 82-pin connector for backplane interconnections. In the 3A CC, the 53 1A logic circuit packs average about 44 SICs, resulting in an average of 308 gates per circuit pack, or a total of 16,482 gates. Figure 8–14 shows a detailed functional diagram of the 3A CC and the percentage of logic gates used in each function unit.

Another insight into how the gates are used in the 3A is shown in Figure 8–15. The figure shows the relationship between working gates, maintenance access gates, and self-checking logic. The working gates are the portion that contributes to the data processing functions, while the maintenance access gates provide the necessary access to make the CC maintainable (that is, maintenance channel and control panel). The self-checking gates are required to implement the parity bits, the check circuits, and the duplicate circuits that make the CC self-checking. As indicated, about 30 percent of the logic is used for checking. The design covers a high degree of component failures. It is estimated that about 90–95 percent of the faults would be detected by hardware error detection logic. Certain portions of the checkers, timers, and interrupt logic are not checked. These circuits are periodically exercised under program control to ensure that they are fault-free.

THIRD-GENERATION 3B20D PROCESSOR

The 3B20D Processor is the first designed for a broad range of AT&T applications. Its development is a natural outgrowth of the continuing need for high availability, real-time control of electronic switching systems for the telecommunications industry. The 3B20D architecture takes advantage of the increased efficiency and storage capabilities of the latest integrated-circuit technology to significantly reduce its maintenance and software development costs.

Figure 8–16 shows the trend of processors for AT&T switching systems for the past three decades. The first-generation processors, the No. 1 and the No. 2, were designed specifically for controlling large (several thousand to 65,000 lines) and medium (1,000–10,000 lines) telephone offices. The predominant cost of these systems, as in most early systems, was the cost of the hardware. The advent of silicon integrated circuits in the mid-1960s was the technological advance needed for dramatic performance improvements and cost reductions in hardware. Integrated circuits led to the development of the second generation of processors (the No. 1A and the No.

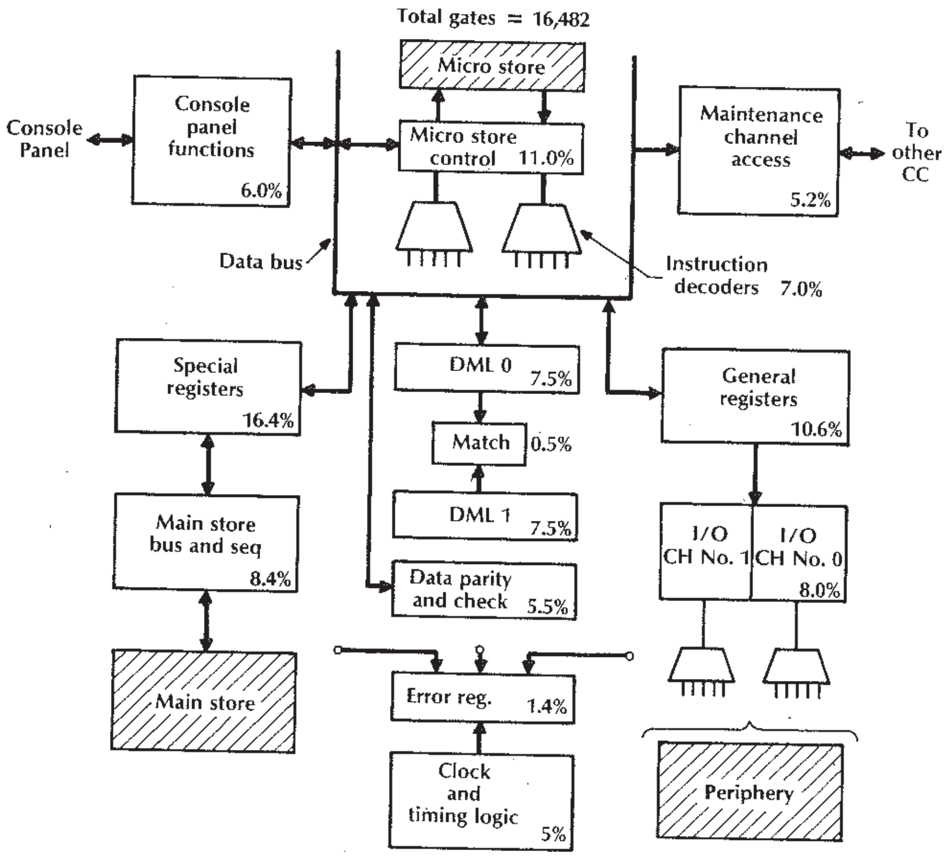


FIGURE 8–14
No. 3A Processor's CC gate count

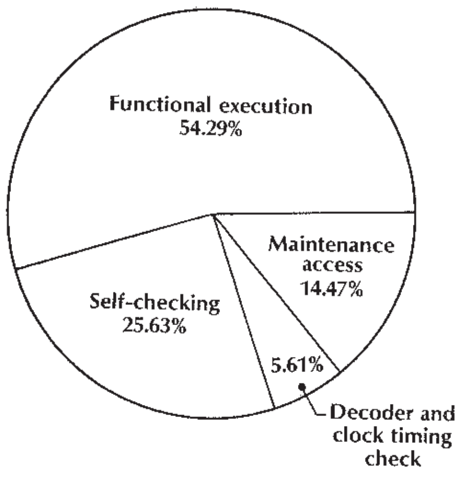


FIGURE 8–15
Logic gates in No. 3A Processor's CC (total gates = 16,482)

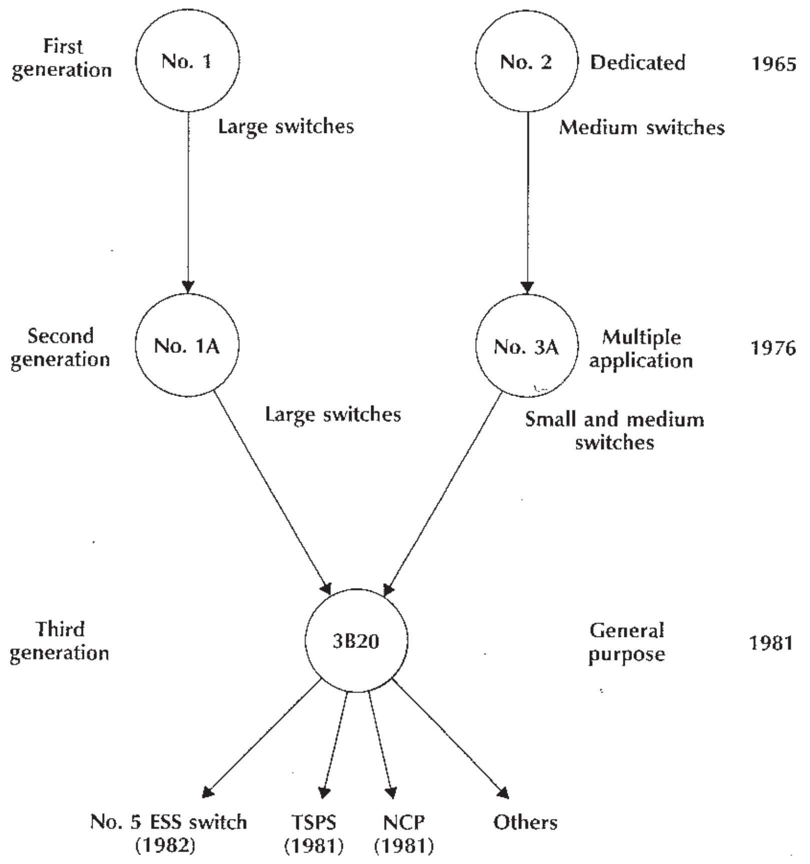


FIGURE 8–16 *Processor trends for AT&T switching systems*

3A). These processors, unlike the first-generation machines, were designed for multiple applications; the third-generation machines have even greater capabilities.

The 3B20D Processor, the first member of the third generation, is a general-purpose system. Its versatile processing base fulfills the varied needs of telecommunications systems. Several thousand 3B20D sites are currently providing real-time data base processing for enhanced 800 service, network control point systems, high-capacity processors for the traffic service position system, the central processor in the administration module for the 5ESS systems, and support processors for the 1A ESS and 4ESS systems.

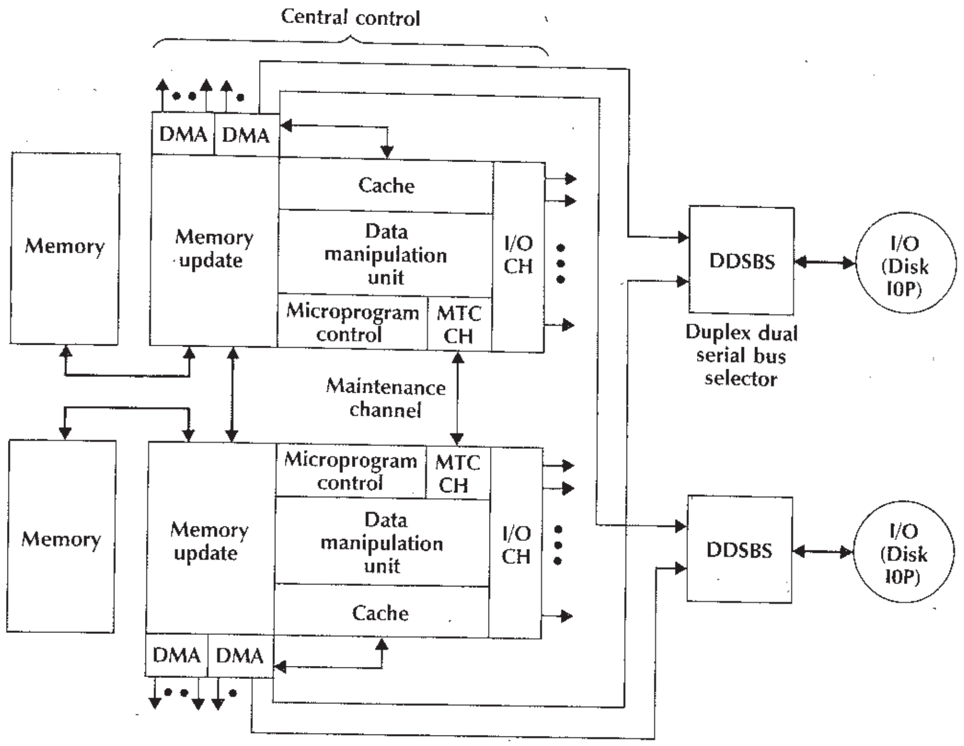
Overview of 3B20D Processor Architecture

The successful deployment and field operation of many electronic switching systems and processors (notably the No. 3A) have contributed to the design of the 3B20D. Previous systems have demonstrated the simplicity and robustness of duplex configurations in meeting stringent reliability requirements [Toy, 1978; Storey, 1976]. Hence, a duplex configuration forms the basic structure for both the hardware and software architecture for the 3B20D. The 3B20D processor also has a concurrent, self-checking design [Toy and Gallaher, 1983]. Extensive checking hardware is an integral part of the processor. Faults that occur during normal operation are quickly discovered by detection hardware. Self-checking eliminates the need for fault-recognition programs to identify the defective unit when a mismatch occurs; therefore, the standby processor is not required to run synchronously. System maintenance is simplified because reconfiguration into a working system is immediate. Another advantage of the self-checking design is that it permits more straightforward expansion from simplex to duplex or to multiple processor arrangements.

As opposed to the hardware-dominated costs of the first- and second-generation processors, the costs of the 3B20D, as is typical of current systems, are dominated by software design, updating, and maintenance expenditures. To reduce these costs as much as possible, the 3B20D supports a high-level language, a customized operating system, and software test facilities. By combining the software and hardware development efforts, an integrated and cost-effective system has evolved.

Figure 8–17 shows the general block diagram of the 3B20D Processor. The CC, the memory, and the I/O disk system are duplicated and grouped as a switchable entity, although each CC may access each disk system. The quantity of equipment within the switchable block is small enough to meet stringent reliability requirements, thus avoiding the need for complex recovery programs. Each CC has direct access to both disk systems; however, this capability mainly provides a valid data source for memory reloading under trouble conditions. The processors are not run in the synchronous and match mode of operations as is done in early systems. However, both stores (on-line and standby) are kept current by memory update hardware that acts concurrently

FIGURE 8–17
3B20D Processor general block diagram



with instruction execution. When memory data is written by the CC, the on-line memory update circuit writes into both memories simultaneously. Under trouble conditions, the memory of the standby processor contains up-to-date information; complete transfer of memory from one processor to another is not necessary.

The direct memory access (DMA) circuits interface directly with the memory update circuit 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. The duplex dual serial bus selector allows both of the processors to access a single I/O device. For maintenance purposes, the duplex 3B20D CCs are interconnected by the maintenance channel. This high-speed serial path provides diagnostic access at the microcode level. It transmits streams of microinstructions from the on-line processor to exercise the standby processor. Other microinstructions from an external unit help diagnose problems.

The 3B20D Processor

The 3B20D Processor performs all the functions normally associated with a CPU and other functions, including duplex operation, efficient emulation of other machines, and communication with a flexible and intelligent periphery [Rolund, Beckett, and Harms, 1983]. The microprograms in the processor minimize the amount of hardware decoding and simplify the control structure. There is substantial flexibility in the choice of instruction formats that may be interpreted.

The CPU is a 32-bit machine with a 24-bit address scheme. Most of the data paths in the CC are 32 bits wide and have an additional 4 parity check bits. The CC architecture is based on registers; multiple buses allow concurrent data transfers. Separate I/O and store buses allow concurrent memory access and I/O operations. A block diagram of the central control is shown in Figure 8–18. These functions and subsystems control the CC and all interactions with it.

The microprogram control subsystem provides nearly all the complex control and sequencing operations required for implementing the instruction set. The microcode supports up to three different emulations in addition to its native instruction set. Other complicated sequencing functions are stored in the microinstruction store, or microstore. The microcontrol unit sequences the microstore and interprets each of its words to generate the control signals specified by the microinstruction. Execution time depends on the complexity of the microinstruction. Each microinstruction is allocated execution times of 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 frequently used instructions are implemented with a single microinstruction.

The data manipulation unit (DMU) contains the rotate mask unit (RMU) and the arithmetic logic unit (ALU), as shown in Figure 8–19. These units perform the arithmetic and logic operations of the system. The RMU rotates or shifts any number of bits from positions 0 through 31 through a two-stage, barrel-shift network. In addition, the RMU performs AND or OR operations on bits, nibbles, bytes, half words, full words, and miscellaneous predefined patterns. The RMU outputs go directly into the ALU. The ability of the RMU to manipulate and process any bit fields within a word greatly enhances the power of the microcode.

The other component of the DMU is the ALU, which is implemented using AMD Company's 2901 ALU slices. The 2901s are bipolar 4-bit ALUs (see Figure 8–20) [AMD, 1979]. Eight 2901 chips provide two key elements: the 2-port, 16-word RAM and the high-speed ALU. Data in any of the 16 words addressed by the 4-bit A-address input can be used as an operand to the ALU. Similarly, data in any of the 16 words defined by the 4-bit B-address input can be simultaneously read and used as a second operand to the ALU. Because the internal 16-word RAM is dedicated as general

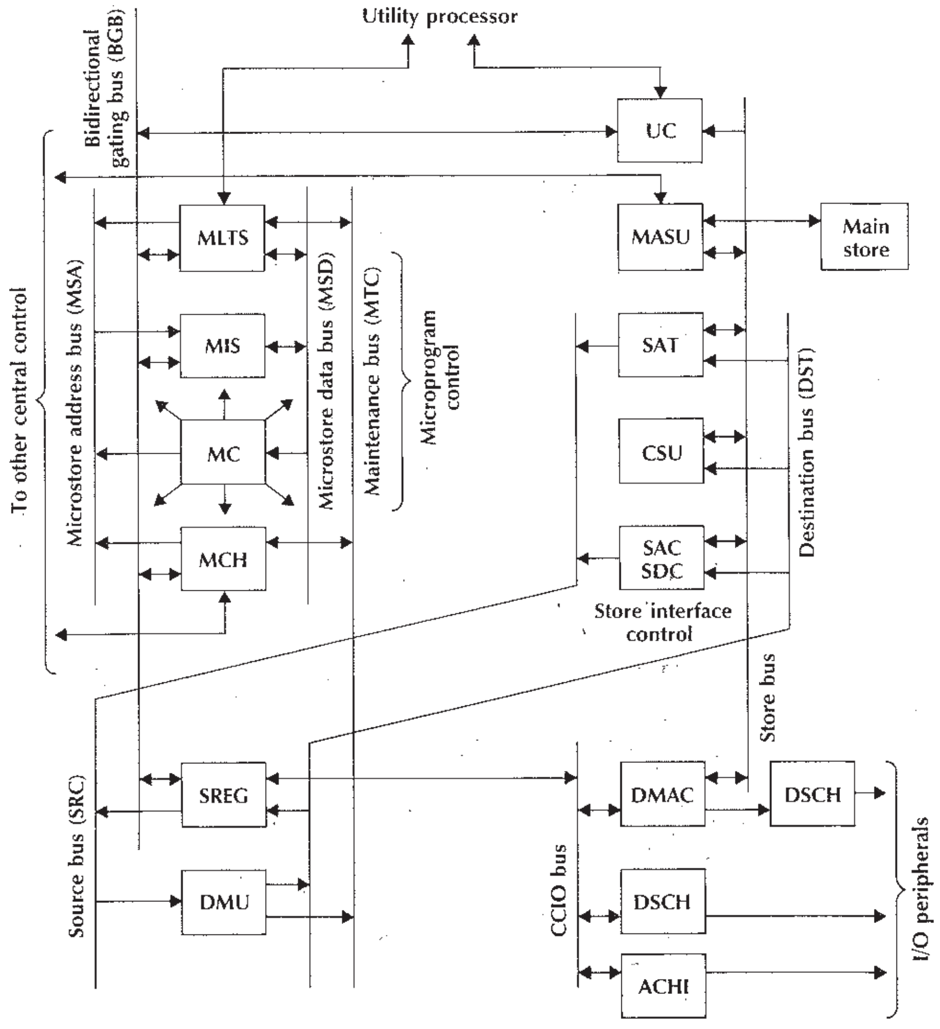
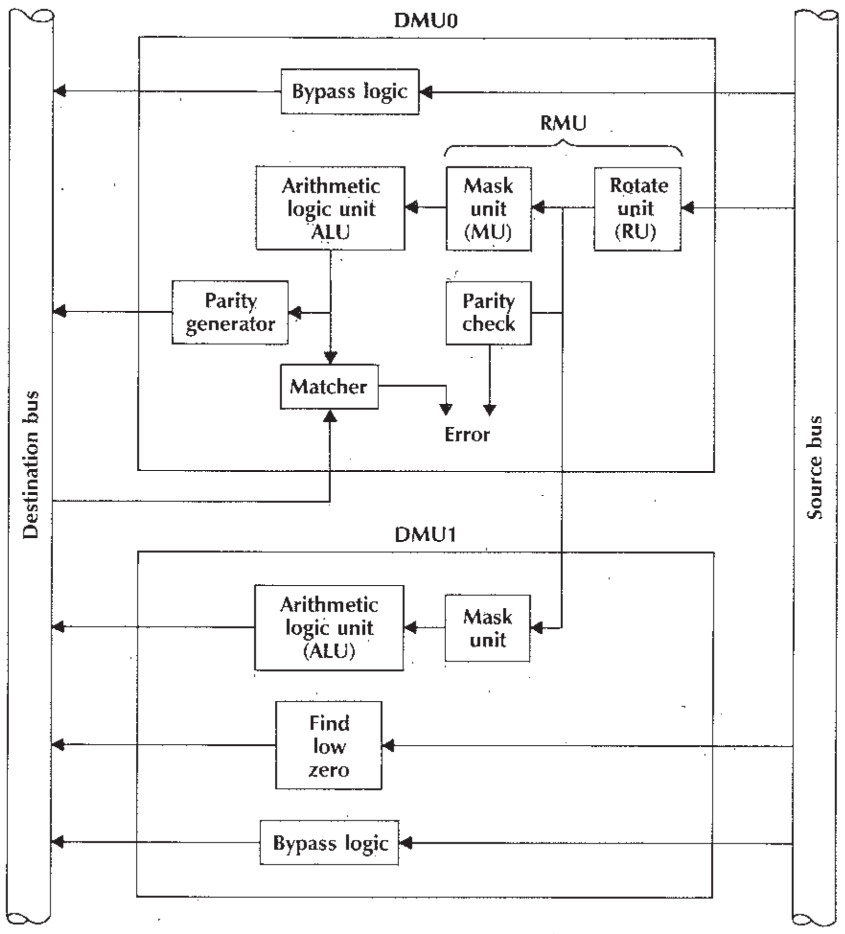


FIGURE 8–18
3B20D Processor's central control

MLTS:	Microlevel test set
MCH:	Maintenance channel
MIS:	Microinstruction store
MC:	Microcontrol unit
DMU:	Data manipulation unit
SREG:	Special registers
SAC:	Store address controller
SDC:	Store data controller

SAT:	Store address translator
UC:	Utility circuit
MASU:	Main store update
CSU:	Cache store unit
DMAC:	Direct memory access controller
DSCH:	Dual serial channel
ACHI:	Application channel interface

FIGURE 8–19
3B20D Processor's data manipulation unit



registers, the result can be directed to the RAM word specified by the B-address, thus optimizing the performance speed of the arithmetic and logical operations involving general registers and the output of the RMU.

The logic blocks of Figure 8–19 depict the self-checking capability of the RMU and ALU. The first-stage byte rotate unit of the RMU is checked for byte parity, which it preserves; the mask unit, including the second-stage bit rotate, is checked by duplication. The ALU is also checked by duplication. The data is taken from one ALU, and parity is generated from the other. The data from one ALU is also matched with that from the duplicate. The underlying self-checking strategy, illustrated here and used throughout the CPU, is to use parity checking where parity is preserved and duplication of logic where parity is not preserved.

The special registers (SREG) associated with the operation of the CC are external to the DMU, unlike the 16 general registers inside the DMU that are available to the programmer. Most of the special registers are not explicitly specified by the 3B20D instruction set. They are characterized by their special dedicated functions and receive their inputs from sources other than

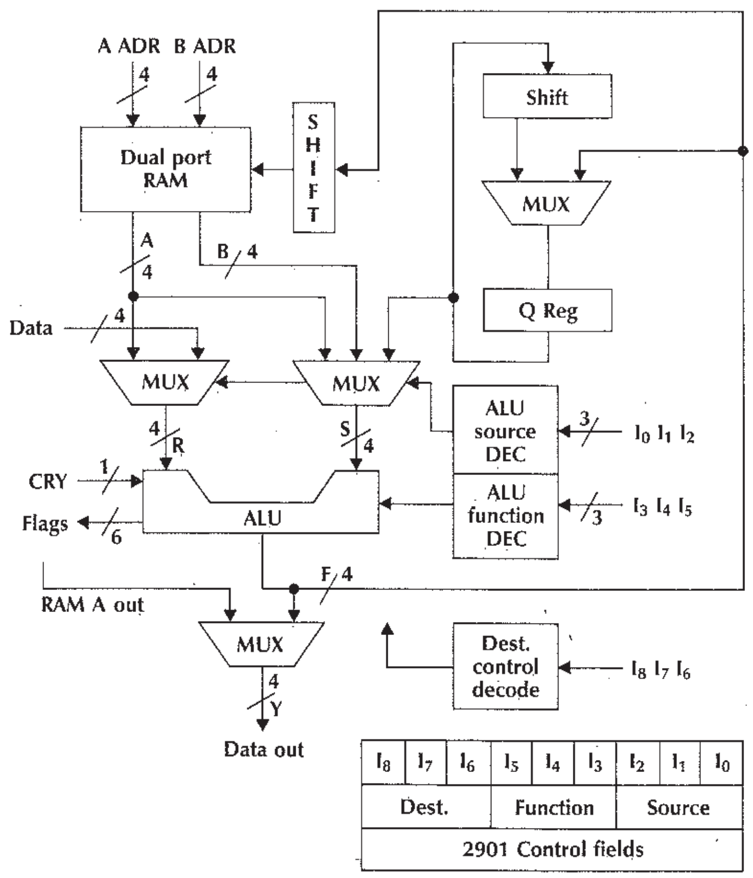


FIGURE 8–20 *AMD's 2901 internal architecture*

the internal data bus. They control and direct the operation of the processor. Some of the special registers are (1) error register, (2) program status word, (3) hardware status register, (4) system status register, (5) interrupt register, and (6) timers. In addition to these registers, a 32-word RAM that is available only at the microcode level is provided within the SREG block. It is used for scratch-pad space, and it is preassigned registers, such as those that support memory management, to facilitate and enhance the power of microprogram sequences.

The *store interface circuit* controls the transfer of data and instructions from system memory to the CC. Two controls, the store address control and the store data control, handle memory addressing, update the program counter, and fetch instructions. Associated with the SAC are the program address, the store address register, and the store control register. Associated with the store data control are the store data register, the store instruction register, and the instruction buffer. These circuits ensure a continuous flow of instructions to the microcontrol unit.

Memory mapping is required in the implementation of a virtual address multiprogramming system. The store address translation (SAT) facility is the mechanism that provides memory mapping between a program-specific virtual address and its corresponding physical address.

Address translation hardware is included in the SAT by the address translation buffer to facilitate memory management [Hetherington and Kusulas, 1983].

The cache store unit is an optional circuit that improves overall system performance by reducing the effective memory access time. The cache is a four-way, set-associated memory containing 8K bytes.

The main store update unit provides a multiport interface to the memories as both DMA and CC circuit attempt to use the memory. The update circuit arbitrates asynchronous requests from the on-line CC and the on-line DMA. The cross-coupling between the memory update units permits the on-line CPU to access either memory, or both memories, for concurrent write operations.

The main store uses dynamic memory devices and high-speed, TTL-compatible, gate-array integrated circuits. It consists of a single circuit board main store controller and up to 16 megabytes may be equipped within the central control frame. Throughout the central control, byte parity is maintained over each byte of the data word. By adding four error-correction code bits in a modified form of Hamming code (in addition to the byte parity bits), the main store performs single-bit error correction and double-bit error detection.

Input/output interfacing is done in several ways in the CC. The communication path between the CC and the I/O channels is through the central control input/output (CCIO) bus, which 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 (DSCHs). I/O channels, including user-specific interfaces, can be connected directly to the CC by means of the CCIO bus. Two standard interfaces are the DSCH, a high-speed, multiport serial interface, and the application channel interface (ACHI). The ACHI is a high-throughput, parallel bus, peripheral communication path.

The maintenance channel circuit provides diagnostic access to the CC at the microinstruction level. It also controls basic fault recovery and system sanity functions in the off-line processor.

Craft Interface

The maintenance interface is commonly referred to as the craft interface [Barton and Schmitt, 1983] in the telecommunications industry. The craft interface of the 3B20D is markedly different from previous systems developed at Bell Laboratories because it relies almost exclusively on video displays and keyboard controls. The earlier systems have key-lamp panels and teletypewriters in their master control centers.

The craft interface includes hardware, firmware, and software that enable maintenance personnel to obtain the status of, and exert control over, the system. Status information is presented visually as graphical displays and text messages on various terminals and printers; audible alarm circuits can also be connected to the 3B20D. System control is exerted primarily through a keyboard attached to the video display terminal. System control is also possible from remote locations, called switching control centers. The data links to the remote sites use the international standard message protocol (X.25) because of its low vulnerability to noise and other data communication failures. The adoption of the X.25 message protocol standardizes remote access to the 3B20D processor for packet switching networks.

Figure 8–21 is a functional block diagram of the craft interface. Each of the duplex processors is connected to both input/output processors (IOPs), which, as mentioned previously, support up to 16 peripheral controllers. The IOP software driver contains handlers that deal with the specialized functions of the peripheral controllers. Maintenance personnel use the read-only

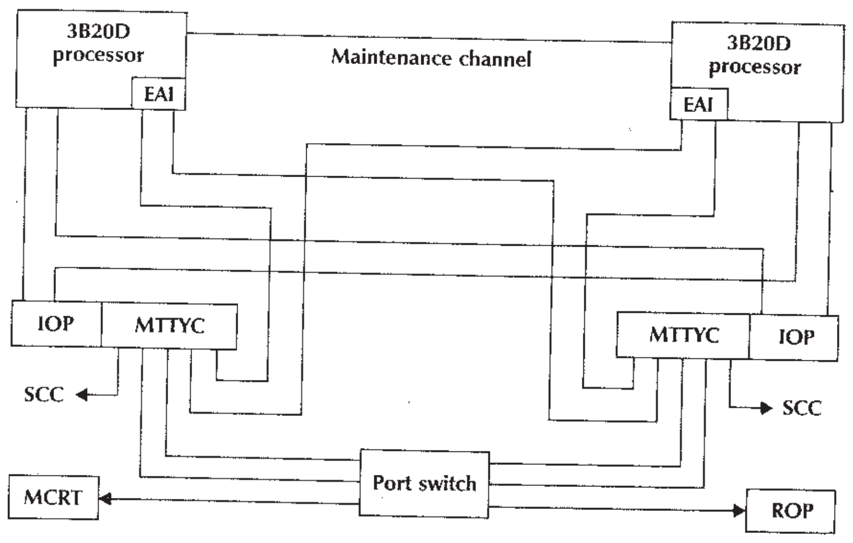


FIGURE 8–21 *Craft interface hardware overview*

printer (ROP) and the maintenance CRT (MCRT). The ROP logs all important status messages. The MCRT is a keyboard display terminal. The system contains only one ROP and one MCRT because the port switch keeps the ROP and the MCRT connected to the active on-line processor.

All capabilities of the craft interface are accessible from a remote switching control center by means of a dedicated data link. The data link is duplicated; it includes a primary link and a backup link. Both links use the CCITT X.25 communication protocol. The MCRT, ROP, and X.25 links are attached to a peripheral controller known as the maintenance teletype controller (MTTYC). The craft interface handler controls the transfer of data to and from the peripheral devices associated with the MTTYC. The MTTYC is connected directly to the emergency action interface (EAI) in the central processor. The EAI menu on the MCRT gives basic status information and manual control of the processor regardless of DMERT (operating system, see the following section) software sanity; this access is controlled totally by the firmware in the MTTYC. This reliable, high-capacity data link for remote maintenance makes the 3B20D well suited for unattended operation.

DMERT: The UNIX RTR Operating System

The operating system used in the 3B20D is the duplex multi-environment real-time (DMERT) operating system, which is now called the UNIX RTR operating system [Kane, Anderson, and McCabe, 1983; Grzelakowski, Campbell, and Dubman, 1983]. It has a process-oriented structure that emphasizes high data availability. It is designed for both real-time and time-shared operations. The basic architecture of the DMERT operating system is based on an earlier system named MERT [Lycklama and Bayer, 1978] and the UNIX operating system [Ritchie and Thompson, 1978]. Both the UNIX and the MERT operating systems were developed to execute on commercial equipment. Currently, the UNIX operating system is widely used, and the MERT operating system has been replaced by its duplex successor, the DMERT operating system. Experience gained from the operating system of the earlier No. 3A Processor, a real-time monitor known as the extended operating system (EOS) [Elmendorf, 1980], also benefited the designers of the DMERT system.

The DMERT operating system has a sophisticated architecture that draws on the proven design concepts of the EOS, MERT, and UNIX operating systems (see Figure 8–22).

Application programmers may add code at the kernel process, supervisor process, and user levels. The multilevel structure makes the DMERT operating system flexible and efficient in its use of real time. The structure of the virtual machines permits the management of both real-time applications and time-shared background tasks. For example, Figure 8–23 shows how telephone switching software is allocated to the different levels. The operating system maintains a process hierarchy based on 16 execution levels. Time-critical functions such as the I/O drivers, fault recovery, and call processing are implemented at the kernel process level. A kernel process may belong to levels 3 through 15 (levels 0 through 2 are reserved for the time-sharing environment). By means of this hierarchical execution-level structure, applications are able to customize their control and distribution of real time.

The portion of real time that is not used by the kernel or kernel processes is time shared among supervisor and user processes. Deferrable jobs such as traffic reports, recent changes, and diagnostics are implemented at the highest user level. Processes supporting the time-sharing environment are run at execution level 2. These processes are run just beneath the real-time hierarchy; they gain control of the processor only after all the real-time work is completed. By supporting both real-time and time-sharing environments, the DMERT operating system makes efficient use of its physical resources.

Fault Recovery

When any of the unique fault detection circuits detects an error condition, an error interrupt (or error report in the case of certain peripherals) is registered in the processor. The most severe error interrupts result in automatic hardware sequences that switch the processing activity between the processors (hard switch). Less severe errors result in micro-interrupts that activate the microcode and software to recover the system. This layered approach that constitutes the recovery

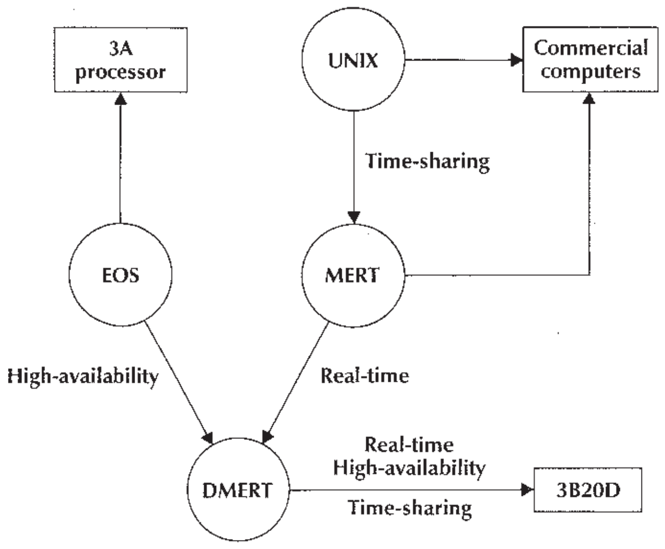


FIGURE 8–22
Bases of DMERT architecture

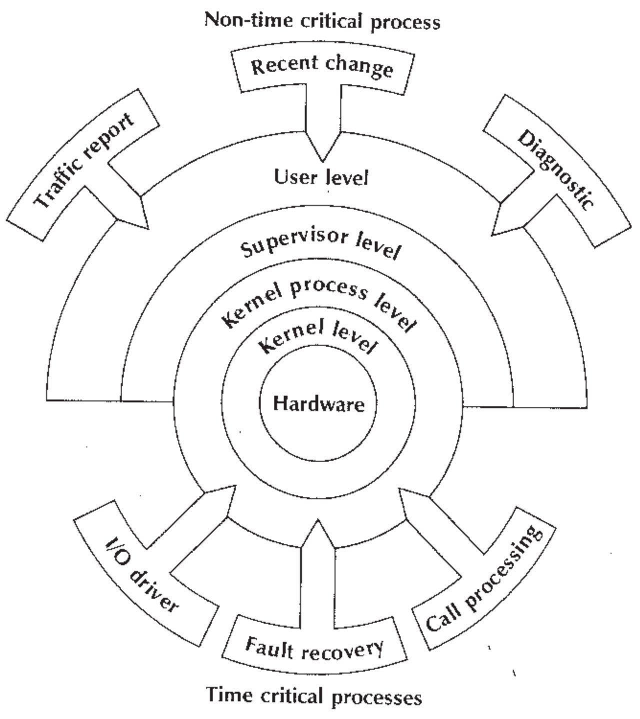


FIGURE 8–23 *Example of DMERT multi-environment structure*

architecture is depicted in Figure 8–24 [Hansen, Peterson, and Whittington, 1983]. Microcode provides low-level access to the hardware, and the recovery software provides the high-level control mechanisms and decision making.

Figure 8–25 illustrates the principal architecture and features of the recovery software. The bootstrap and initialization routines contain a fundamental set of microcode and software algorithms that control initializations and recoveries. These actions are stimulated by a maintenance restart function (MRF), which represents the highest-priority microinterrupt in the system. An MRF sequence may 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 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 whether an error is exceeding the 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 for both hardware and software errors.

Hardware Fault Recovery. The 3B20D Processor has built in self-checking circuitry that detects hardware faults as soon as they occur. This circuitry simplifies recovery, since early detection limits the damage done by the fault. Faults in this category indicate that the processor is no longer capable of proper operation and result in an immediate termination of the currently running processor and a switch to the standby processor. Since the standby processor does not

FIGURE 8–24
Recovery software structure

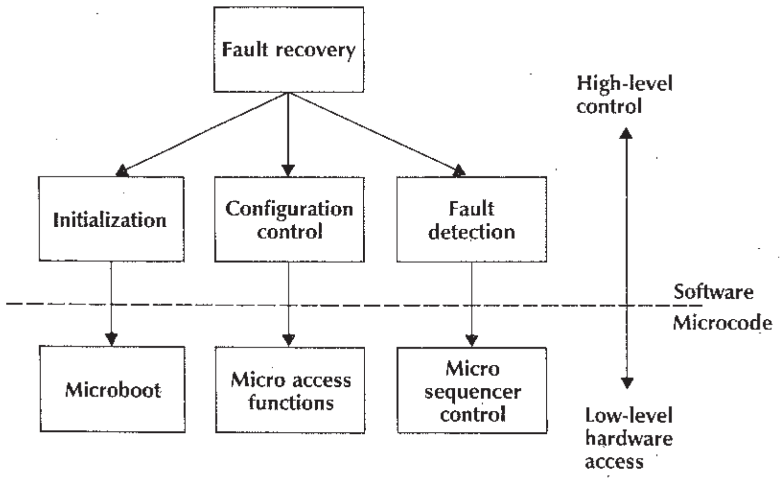
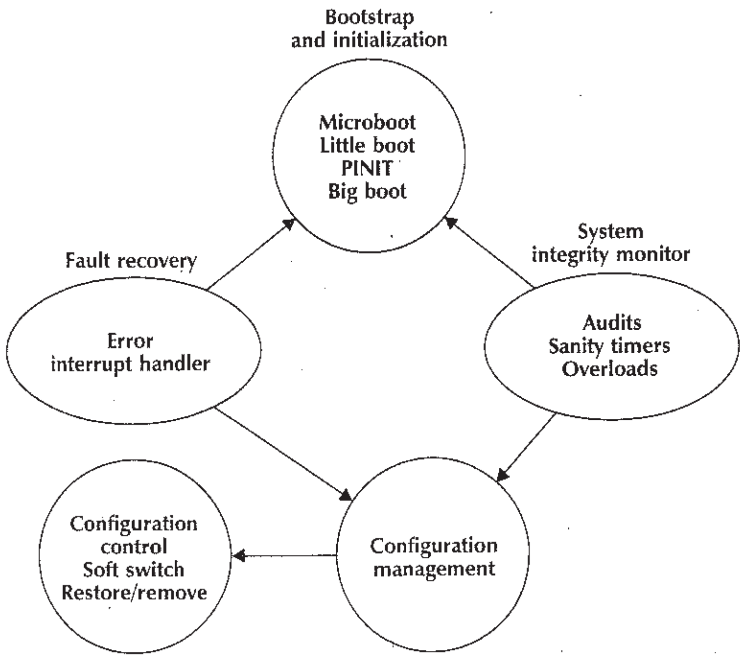


FIGURE 8–25
Fault recovery architecture



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 termination and switch of the processors. Examples of errors of this kind are hardware faults detected in the standby processor memory and software errors such as write-protection violations. Other errors in this category are the hardware faults that are handled by the self-correcting circuitry. Although most units have self-checking circuits, some units (such as main memories) have fault rates that justify the addition of 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.

All error interrupts are reported to CONFIG. 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 based on several factors, including the importance of the faulty unit and whether a mate exists for it. If the faulty unit is essential to the system and a mate unit is available, the faulty unit is removed from service and scheduled for diagnostic testing. If there is no available mate unit, the faulty unit is initialized and returned to service until the mate is restored. When the mate is restored, it is switched on line and the faulty unit is scheduled for diagnostic testing. In the case of essential units, it is better to have a faulty unit than no unit. Unessential units are removed from service and scheduled for diagnostic testing whenever their error thresholds are exceeded.

Each processor has a sanity timer that causes an initialization if it expires. The active processor maintains both its own timer and the timer of the standby unit. If the active processor cannot recover from a fault, the sanity timer triggers the initialization of the standby processor.

Special Microcode for Recovery. A large fraction of the microcode in the central control handles system recoveries. Most of this recovery microcode is in ROM because most of the recovery functions are required, regardless of the past history of the CC or its boot devices. Functions that are required even if the CC is not ready to execute its instruction set include microinterrupt processing, maintenance channel assists, and microcode to initialize hardware systems.

Microinterrupt processing handles errors in the address translation buffer to micro-instruction store sequence. Maintenance channel assists allow one processor to access the other processor. Microcode initializes the hardware systems; 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 made available. Arbitrary microsequences loaded into the scratch area are then executed as special tests or functions. Before the software can run, however, the WMS must be loaded from disk. The WMS is loaded as part of the processing of the MRF microinterrupt.

Software Fault Recovery. Software fault recovery is architecturally similar to hardware fault recovery. Each major unit of software has associated with it error detection mechanisms (defensive checks and audits), error thresholds, and error recovery mechanisms (failure returns, audits, and data correction and initialization techniques). Both the system integrity monitor (SIM) and the error interrupt handler (EIH) oversee the proper execution of the process. An error threshold in SIM ensures that a process does not put itself into an infinite execution loop or excessively consume a system resource (for example, message buffers). The EIH, through the use of hardware and microcode detectors, ensures that processes do not try to access memory outside defined limits or execute restricted instructions. Each process has initialization and recovery controls (analogous to hardware) to effect recoveries. Figure 8–26 illustrates this software recovery architecture.

If recovery actions result in the removal of hardware units, diagnostics are dispatched automatically to analyze the specific problem. Audits are the software counterparts for hardware diagnostics; the major difference is that routine audits run more frequently than diagnostics, and they correct certain errors.

Software Audits. 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, the audit manager.

Audits in the DMERT operating system 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 reside in the kernel, additional data reside in critical processes such as the file manager and device drivers. Smaller amounts of auditable data reside in supervisor processes, such as 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. Audits within the DMERT operating system include the following:

- The *message buffers audit* finds and frees lost message buffers; that is, messages that have been on the queue of a process for extended periods of time.

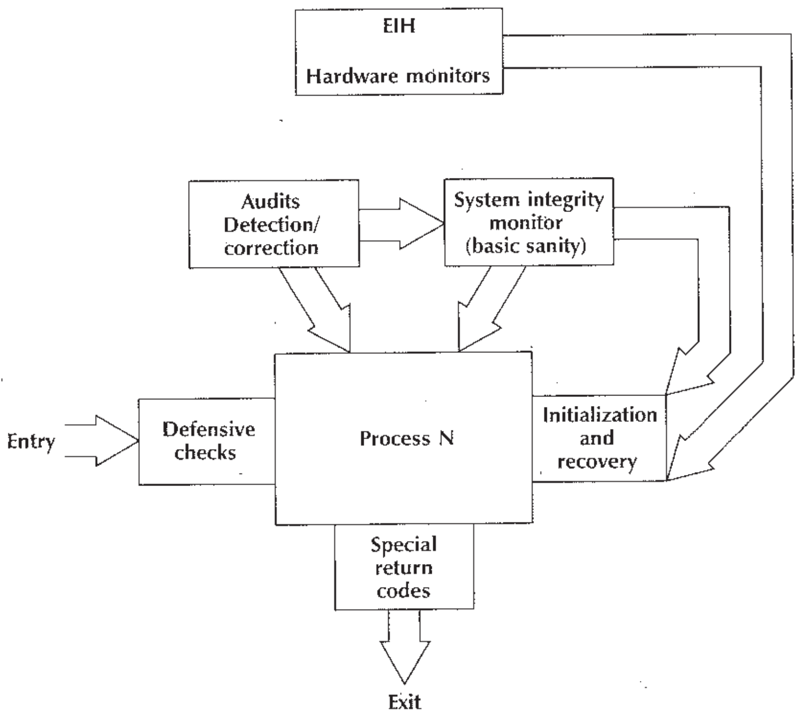


FIGURE 8–26
Software fault recovery architecture

- The *scheduler audit* checks for linkage errors in the ready and not-ready lists of the scheduler.
- The *memory manager audit* recovers lost swap space and corrects any overlap of swap space.
- The *file manager audit* checks all internal file manager structures, including task blocks, buffers, and the mount table. It corrects the information and can back out aborted tasks to free their resources.
- The *file system audit* is demanded by the file manager whenever a file system is mounted in the read/write mode. It checks and corrects the file system's super block, free list and its free-block bit map. The audit verifies the integrity of the mounted file systems concurrent with their use.

System Initialization. When a maintenance restart interrupt occurs, a long sequence of microsteps begins to establish system sanity. Both processors may be in their maintenance restart function (MRF) sequence at the same time, and each may try to become the active processor. The MRF code decides which processor should become active and 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 it is forced on line by an operator command.

A number of tests are made on data in the operating status register to select one of four possible actions: (1) processor initialization, (2) stop and switch, (3) microboot, or (4) tapeboot. The simplest actions are initializing a processor and stopping and switching to the other processor. Switching to another processor is accomplished by sending a switch command over the maintenance channel to the other processor. If an initialization does not recover the system to an operational state, another and more severe initialization is triggered automatically. The initialization interval determines whether escalating is necessary. Any initialization that occurs within the initialization interval (that is, within a specified time interval after an initialization) escalates to the next higher level. The length of the initialization interval is a system generation parameter that is established by the application.

The *microboot program* 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 the boot device. If the craft interface has forced either the primary or the secondary boot device active, it uses that device. Otherwise, the microboot program selects a disk drive based on the state of the initialization status control bits in the system status register. Alternate boots use alternate devices. Microcode is read from the disk and then copied to the writable microstore. Finally, the little boot program is read from the boot partition and given control.

The *tapeboot program* is a complex sequence of microcode that is used only when requested manually from the craft interface. Its function is to create a new system disk from tape. Tapeboot initializes the tape device and disk drive selected by the craft interface and initializes the writable microstore from tape. The load disk tape program is read from tape into main store; memory management tables are created to allow it to run the hardware complex without the operating system present. The load disk tape program then reads the tape to make a DMERT disk image.

Emergency Mode. The emergency mode on the 3B20D refers to the facilities and procedures that prevent the system from experiencing a total outage. For example, emergency facilities are applied when the system is unable to recover automatically. The most frequent emergencies encountered include duplex failures of the control unit, duplex failures of the system disks, duplex failures of the essential I/O devices, and failures of fault recovery to find a working configuration of the hardware. Other problems that require the emergency mode include software faults that do 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 problems that may cause the failure of the 3B20D as a system. The emergency action interface (EAI) on the 3B20D provides manual initialization capabilities that can recover the system from several of the conditions mentioned above. The EAI allows the maintenance personnel to demand a specific processor and disk configuration if a certain configuration is causing problems. The EAI also allows the craft to reconfigure the system to handle maintenance hardware failures. For example, the craft may 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 the loss of subsystem capabilities.

The 3B20D provides other emergency mode capabilities through the port switch select, the disk power inverter select, and the unit power switches. These devices are used by maintenance personnel to reconfigure the system manually to handle certain problems. Under unstable bootstrap conditions, the 3B20D outputs diagnostics information called *processor recovery messages.* These messages provide a general set of diagnostics in the event of a complete system outage.

The final backup repair procedure consists of the dead start diagnostics. Primarily used as installation tools, the dead start diagnostics allow a nonworking processor to be repaired from a remote host processor.

Fault Diagnostics

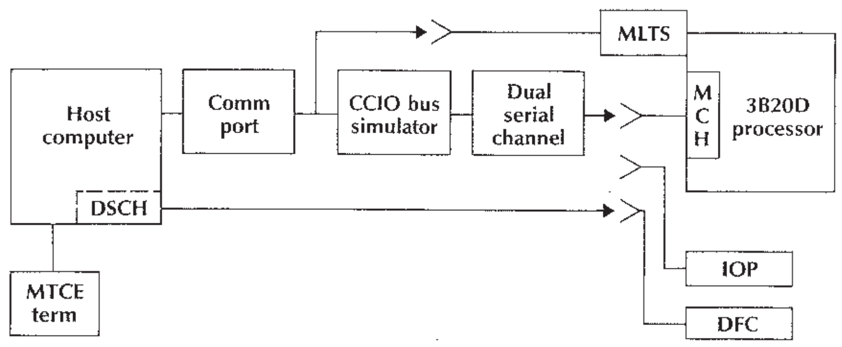
As with earlier processor designs, 3B20D processor diagnostics detect faults efficiently and effectively, provide consistent test results, protect the contents of memory, do not interfere with normal system operation, allow automatic trouble location, and are easy to maintain and update. To meet these design objectives, the diagnostic control structure is an integral part of the DMERT operating system and supports the evolutionary stages of development [Quinn and Goetz, 1983].

Diagnostic Environments. As shown in Figure 8–27, the 3B20D Processor may be diagnosed from several execution environments. During the early phase of its development, a local host computer was used to support hardware, software, and diagnostic design. This arrangement continues to be used in factory testing. Later in the development phase, more efficient use was made of the host computer by providing access to a remote 3B20D Processor over a dial-up telephone line. In the final development stage (the standard duplex system configuration), the active control unit was capable of diagnosing its own peripheral controllers and the standby control unit. Each of these access arrangements is discussed below.

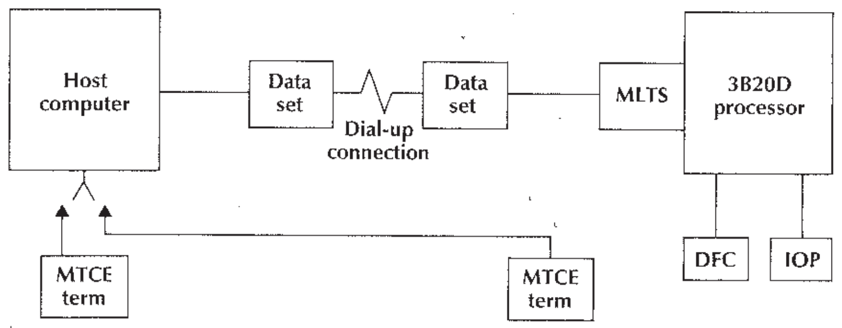
Figure 8–27a shows three local host access arrangements. In the first arrangement, diagnostic programs executing in a host computer send test inputs and receive test results through a standard communication port to a microlevel test set (MLTS). The MLTS connects directly to the 3B20D control unit backplane and provides complete access to and control of the processor's microprogram control circuitry. The second access arrangement uses a circuit 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. Only the MCH, however, is used in the duplex configuration; it communicates with either another MCH or a DSCH. As shown, the CCIO/BS-DSCH access path is also used to diagnose the IOP and the disk file controller (DFC). The third access arrangement is used when the local host is a 3B20D processor. The path in this case is from the DSCH of the host processor to the MCH, IOP, or DFC of the target machine.

The DSCH communicates over distances of approximately 100 feet. Remote host (Figure

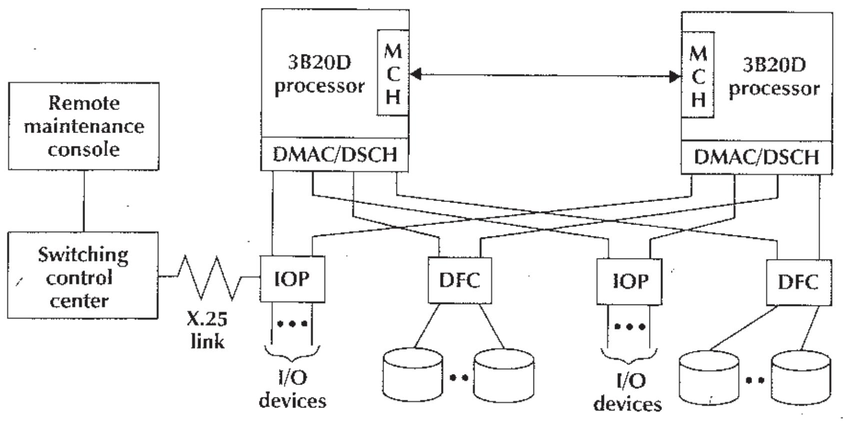
FIGURE 8–27
3B20D Processor's diagnostic environments



a. Local host



b. Remote host



c. Duplex mode

8–27b) access arrangements are used for diagnosing over longer distances. Using data sets and a telephone line, tests stored and executed on a remote computer are applied through the MLTS to the control unit. Peripheral controllers (IOP and DFC) may also be diagnosed by downloading tests into the control unit and executing them. Although remote host diagnostics are useful when a local host is unavailable, execution performance is limited by the transmission facilities used.

The primary diagnostic execution environment is the duplex mode of the 3B20D (Figure 8–27c). The active (on-line) processor acts as a local host for diagnosing the standby (off-line) processor. A link between maintenance channels 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, which is a DSCH attached to the direct memory access controller. Tests of the links from the off-line processor to the peripherals may also be run under the control of the active processor. As shown in Figure 8–27c, the duplex system configuration also supports remote monitoring and control of diagnostics over a dedicated link to a switching control center.

Diagnostic Control Structure. The diagnostic control structure is depicted in Figure 8–28. The modules that provide access to the equipment configuration data base (ECD) are at the kernel

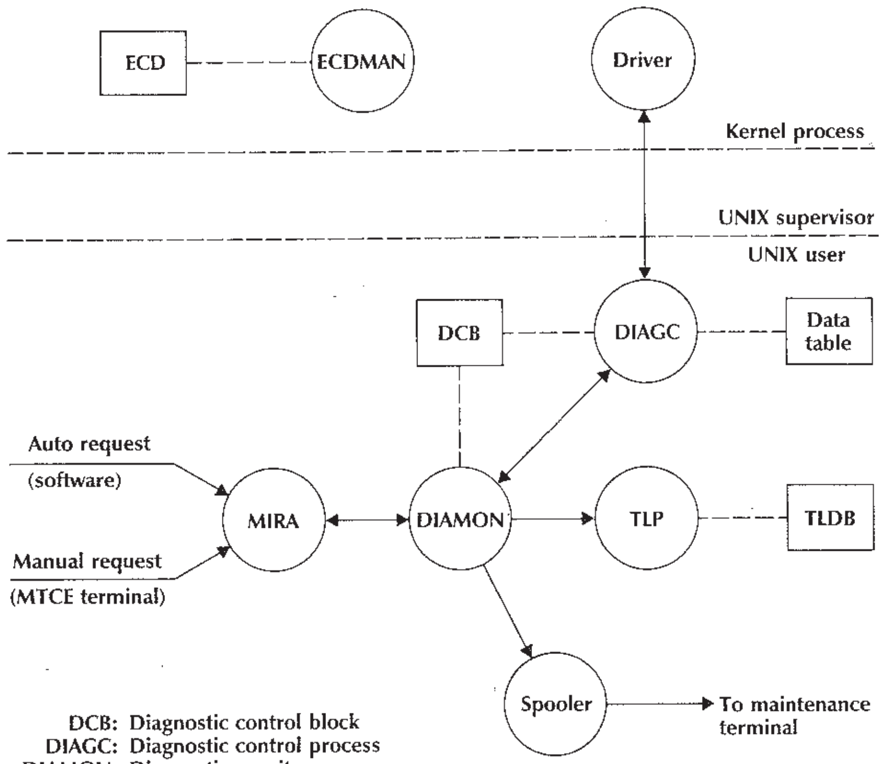


FIGURE 8–28
Diagnostic control structure

DCB:	Diagnostic control block
DIAGC:	Diagnostic control process
DIAMON:	Diagnostic monitor
ECDMAN:	Equipment configuration data manager
MIRA:	Maintenance input request administrator
TLP:	Trouble location process
TLDB:	Trouble location database
Data table:	Diagnostic data table files

process level. All the information relevant to the diagnostic tests that should be applied to each hardware unit is contained in the ECD. This information includes the name of each hardware unit; its 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 made.

The UNIX operating system supervisor resides at the supervisor level and provides a protected environment and operating system service 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 messages from the diagnostic structure are sent to the system spooler for printing.

The MIRA schedules and dispatches all the maintenance requests. MIRA has two queues, a waiting queue and an active queue, to administer maintenance requests. Requests are serviced according to their priorities and the availability of resources. Manual requests have higher priorities than requests initiated automatically. For each service request, MIRA creates 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 (EIH). Execution of each diagnostic is directed from start to finish by DIAMON.

DIAGC is a generic name that refers to a class of diagnostic control processes. The DIAGC is a unit or application-dependent module that controls the execution of tests. DIAGC contains all the application-dependent task routines, 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 may interface with it.

If the diagnostic request specifies the TLP option, the trouble-locating process is invoked after the diagnostic testing is completed. The TLP compares characteristics of the failures found by the diagnostics with a resident data base of fault signatures. In each data table, the tests are partitioned into groups. A test failure in a group sets a flag bit, called a key, which is permanently assigned to the group. The TLP searches the results of the diagnostics and, based on the phase and key information, creates an ordered list of the closest signatures and, ultimately, of the suspected faulty equipment. This approach makes the data base and sorting processes less sensitive than earlier methods of testing changes to circuits and marginal failures. During the development of the 3B20D, the trouble-locating data base (TLDB) was generated by physically inserting faults into units in a test laboratory. The TLDB of operational systems can be modified directly by inserting information into the test data table.

Diagnostic 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 type of circuit under diagnosis. Requests may diagnose an entire unit, a particular subunit, or all the subunits in a specified community. Individual test phases or ranges of phases may be executed and the results printed with optional amounts of detail. Some diagnostic test phases, because of their long execution time requirements or their dependence on the availability of other system hardware resources, are restricted to manual initializations. Interactive features, such as stepping, pausing, and looping, facilitate difficult repairs. Units are restored to service automatically if they pass all tests. Several host computer versions of the software are supported, along with application-dependent interfaces.

Diagnostics are initiated manually or automatically. Manual requests may be entered from a local maintenance terminal or through a work station at a switching control center that is connected to the processor with a synchronous data link. Automatic requests originate from other software modules, including the error interrupt handler, the routine exercise scheduler, and the application software modules.

Evaluation. The stringent availability requirements of AT&T applications using the 3B20D Processors have a significant effect on all the aspects of the system design. The diagnostic and maintenance designers were actively involved in meeting these requirements from the initial architectural planning and requirements generation. Many hardware features monitor system integrity, detect errors, reconfigure the system, and facilitate repairs. Although some of the features isolate faults during pack repairs, most are used at the system level to effect repairs through circuit pack replacement. Diagnostics, the primary repair capability for the system, makes extensive use of these hardware features for control and observation of the circuitry.

During the development of the processor, diagnostic tests were generated manually and with the aid of hardware logic simulators. To ensure that the diagnostics met the objective of detecting 90 percent of the simulated faults, an extensive evaluation process was carried out. Thousands of faults were inserted at the dual-in-line (DIP) package terminals (pins). These faults provided timely and effective feedback on the design of the diagnostic tests and the development of the trouble-locating data base.

Operational Results of 3B20D Processor

The 3B20D Processors have been in commercial operation since September, 1981. The performance of the 3B20D improved tremendously during the first two years of operation. Figure 8–29 shows the results of field data accumulated over many machine operating hours during early

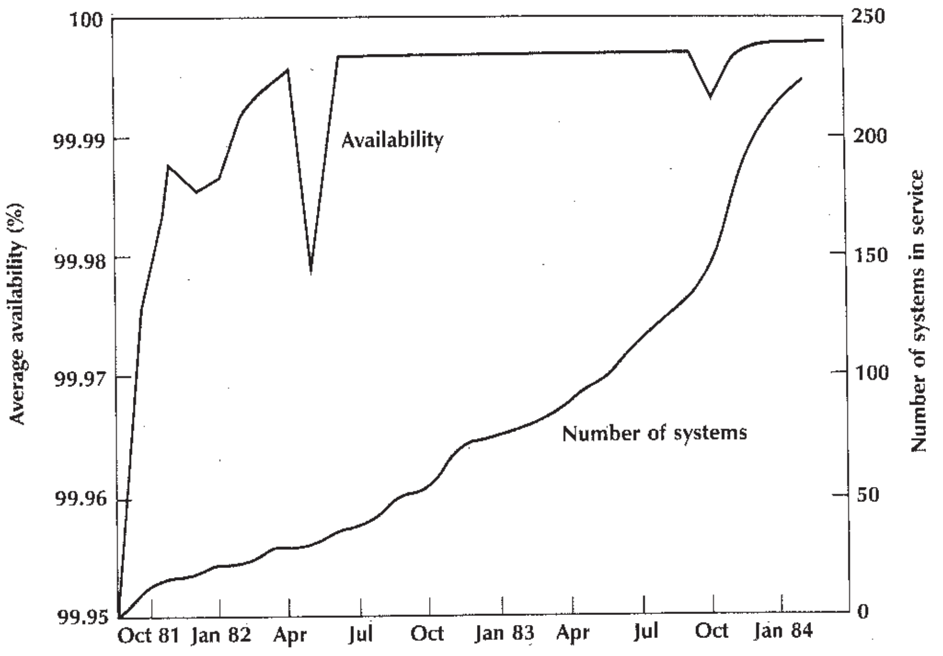


FIGURE 8–29 *Observed availability and number in service of 3B20D Processors*

years of operations [Wallace and Barnes, 1983]. When the first system began commercial service, outages occurred because of software and hardware faults that could only be corrected with field experience. The availability factor improved as the processor design matured and the operating personnel gained experience.

Figure 8–30 shows downtime data for three AT&T processors, including the 3B20D. The experience gained in the design and field operation of earlier electronic switching systems (notably the No. 1A and the No. 3A Processors) has contributed to the design of the 3B20D. The reliability (downtime) curves show that each processor approached its downtime objective more quickly than its predecessor [Wallace and Barnes, 1983]. The data has been smoothed and fit to an exponential decay function for the comparison.

SUMMARY

In order to achieve the reliability requirements, all AT&T switch subsystem units are duplicated. When a hardware failure occurs in any of the subunits, the processor is reconfigured into a working system around the defective unit. The partitioning of subsystem units into switching blocks varies with the size of the processor. For the medium- or small-sized processors, such as the No. 2 or the No. 3, the central control, the main memory, the bulk memory, and the store bus are grouped as a single switchable entity. A failure in one of the subunits is considered a failure in the switchable block. Since the number of components within a switchable block is sufficiently small, this type of single-unit duplex configuration meets the reliability requirement. For larger processors, such as the No. 1 or the No. 1A, the central control, the program store, the call store, the store buses, and the bulk file store are treated individually as switchable blocks.

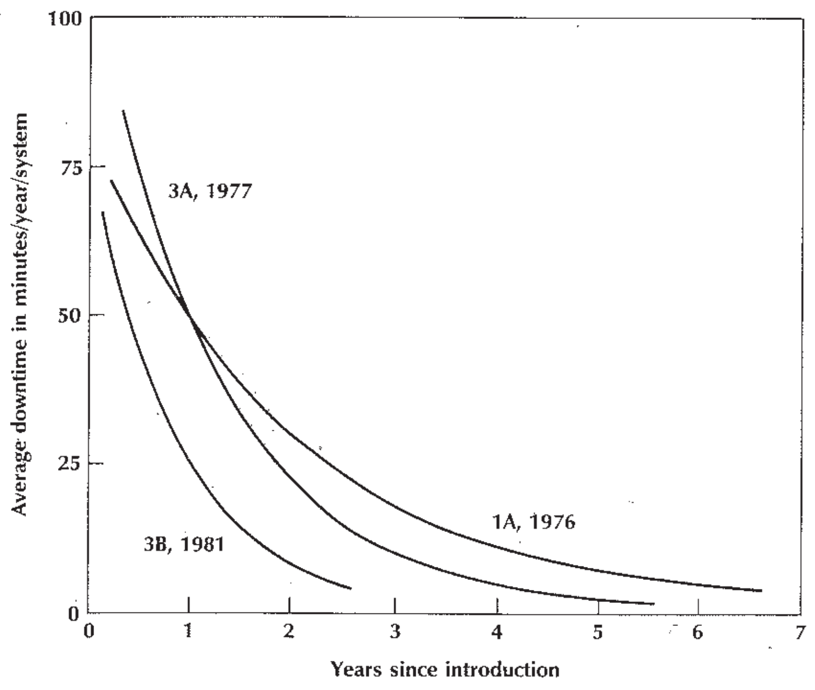


FIGURE 8–30 *Downtime versus time since introduction for three high-availability processors*

This multi-unit duplex configuration allows a considerable number of combinations in which a working system can be assembled. The system is down only when two simultaneous failures occur, one in the subunit and the other in the duplicate subunit. A greater fault tolerance is possible with this configuration. This type of configuration is necessary for the large processor because each subunit contains a larger number of components.

The first generation of processors, which includes the No. 1 and the No. 2, have provided commercial service since 1965 and 1969, respectively. The No. 1 Processor serves large telephone offices (metropolitan); the No. 2 Processor is used in medium-sized offices (suburban). Their reliability requirements are the same. Both processors depend on integrated maintenance software, with hardware that must (1) quickly detect a system failure condition, (2) isolate and configure a working system around the faulty subunit, (3) diagnose the faulty unit, and (4) assist the maintenance personnel in repairing the unit. The primary detection technique is the synchronous and match mode of operation of both central controls. Matching is done more extensively in the No. 1 than in the No. 2, since cost is one of major considerations in the design of the No. 2 Processor. In addition to matching, coding techniques, diagnostic access, and other check logic have been incorporated into the basic design of these processors to realize the reliability objectives.

The widespread acceptance of the 1ESS and 2ESS switching systems has created the need for a second generation of processors: the No. 1A and the No. 3A. They offer greater capability and are also more cost effective. Both processors use the same integrated circuit technology. The 1A Processor extends its performance range by a factor of 4 to 8 times over the No. 1 Processor by using faster logic and faster memory. The 1A design takes advantage of the experience gained in the design and operation of the No. 1 Processor. The No. 1A Processor provides considerably more hardware for error detection and more extensive matching of a large number of internal nodes within the central control. The design of the No. 3A Processor has benefited by the experience gained from the No. 2 Processor. A major departure in the design of the 3A Processor from the design of earlier processors is the nonsynchronous and the nonmatch mode of operation. The No. 3A Processor uses self-checking as the primary means of error detection. Another departure is in the design of the No. 3A Processor's control section: It is microprogrammed. The No. 3A Processor's flexibility permits emulation of the No. 2 Processor quite easily.

The third-generation systems are dominated by software design, updating, and maintenance expenditures. The 3B20D Processor is a general-purpose, high-availability machine that supports many types of applications. A comprehensive set of software tools and facilities improves programming productivity and reduces the cost of software development and maintenance. The hardware architecture efficiently supports high-level languages, particularly the C language. The UNIX RTR operating system was designed concurrently with the hardware to meet the needs of switching and telecommunication systems. Its architecture permits time-critical, real-time code to coexist with time-shared 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 control logic is devoted to self-checking. Self-checking allows concurrent error detection and immediate recovery. In March, 1988, over 1000 5ESS systems and more than 20 million telephone lines were in commercial use, serving the smallest remote switching module of hundreds of lines to the largest office of 85,000 lines with high-quality services.

REFERENCES

AMD, 1979; Ault et al., 1964, 1977; Barton and Schmitt, 1983; Becker et al., 1977; Beuscher et al., 1969; Bowman et al., 1977; Browne et al., 1969; Budlong et al., 1977; Cagle et al., 1964; Chang, Smith, and Walford, 1974.

Downing, Nowak and Tuomenoksa, 1964; Elmendorf, 1980; Genke, Harding, and Staehler, 1964; Goetz, 1974; Grzelakowski, Campbell, and Dubman, 1983; Hansen, Peterson, and Whittington, 1983; Harr, Taylor, and Ulrich, 1969; Hetherington and Kusulas, 1983.

Irland and Stagg, 1974; Kane, Anderson, and McCabe, 1983; Keister, Ketchledge, and Lovell, 1960; Keister, Ketchledge, and Vaughan, 1964; Lycklama and Bayer, 1978; Mandigo, 1976; Quinn and Goetz, 1983; Ritchie and Thompson, 1978; Rolund, Beckett, and Harms, 1983.

Seley and Vigilante, 1964; Smith, 1972; Spencer and Vigilante, 1969; Staehler, 1977; Staehler and Watters, 1976; Storey, 1976; Toy, 1978; Toy and Gallaher, 1983; Tsiang and Ulrich, 1962; Wallace and Barnes, 1983.

THE AT&T CASE

Part II: Large-Scale Real-Time Program Retrofit Methodology in AT&T 5ESS® Switch

L.C. TOY*

Modern telephone systems are continuously undergoing changes to take advantage of rapid advances in hardware technology. In addition, new features are continuously being developed and incorporated as integral parts of the system. The additions and changes must be implemented in real time without disrupting the customer's telephone service. The procedures and methodology utilize the distributed and redundant architecture of the 5ESS Switching System to update and/or grow the system, while at the same time providing continuous service.

In this case study, the architecture of the 5ESS electronic switching system is described briefly, and then considerations for replacement of the resident software in the 5ESS system and how the process is implemented are discussed. The focus is on major software replacement.

5ESS SWITCH ARCHITECTURE OVERVIEW

The 5ESS switch is a fully digital switching system with a distributed processing and switching architecture [5ESS, 1986; Smith and Andrews, 1981; Davis et al., 1981]. It is comprised of an administrative module, a communications module, and one or more switching modules and/or remote switching modules (Figure 8–31). These three basic elements can be configured to cover the complete range of applications, from remote switching modules to large-capacity telephone exchanges [Byrne and O'Reilly, 1985]. The system is highly reliable: Because of the fault-tolerant architecture, the administrative module and switching modules experience only a few minutes of downtime per year [Allers et al., 1983].

• **Switching Module (SM):** Over 95 percent of call processing is handled by the switching module, which is the basic growth unit of the system. In addition to providing circuit and packet switching functions, the switching module connects all external lines, trunks, and special services circuits. Remote switching modules can be located at a considerable distance from the main body of the switch and can be used singly or grouped in clusters to serve groups of up to 10,000 subscribers. Time-division switching is performed in the time slot interchange unit, while most of the call processing is done in the switching module processor unit. Each switching module

* The author acknowledges M.A. Gauldin and F.B. Strebendt, for their assistance and encouragement, and E.E. Haselrick and M.G. D'Souza, for providing reference materials. Development of the retrofit capability resulted from the contributions of many people within AT&T Bell Laboratories and AT&T Technologies.

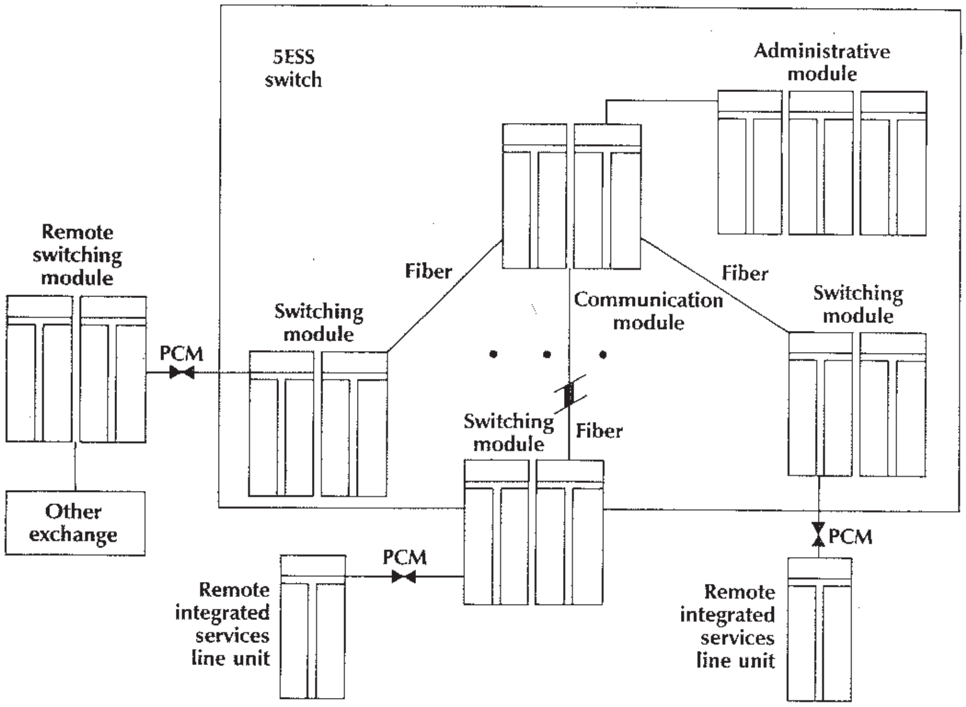


FIGURE 8–31 *5ESS remote switching module capabilities*

has a time slot interchange unit and a switching module processor unit but may differ from other switching modules in the types and quantities of peripheral equipment (line unit, trunk unit, digital carrier line unit, and digital line trunk unit). All common equipment in the switching module is duplicated for reliability purposes. Optical fiber network, control, and timing links connect the switching modules to the communications module.

• **Communications Module (CM):** The control-message communication facilities between the switching modules and the administrative module and between any two switching modules are provided by the communication module. The communications module contains a time-multiplexed switch and a message switch. Digital paths for switched connections between the switching modules are provided by the time-multiplexed switch. The message switch passes control messages between any two switching modules and between the switching modules and the administrative module. For reliability, both the time-multiplexed switch and the message switch are duplicated. The initial design of the communications module supported one switching module. A new communications module that supported 30 switching modules was then followed by a communications module with the capacity of 48 switching modules [Anderson et al., 1987]. A large communications module that supports 192 switching modules is now available. Call routing functions now exist in the communications module.

• **Administrative Module (AM):** The administrative module collects traffic and billing data. Many functions not related to call processing, such as fault detection, diagnostics, and fault recovery, are performed by the administrative module, which uses a 3B20D computer [Toy and Gallaher, 1983]. The administrative module consists of a duplicated administrative processor, an input/output processor to interface with terminals, printers, data links and tape units, and a duplicated disk file controller to which the disks are connected. Disk interfaces are SCSI (small computer system interface) standard.The duplicated administrative processors in the administrative module work in active/standby configuration so that if a fault occurs in the active unit, the standby unit can be switched into service. The disk memory provides mass storage for programs and data [Haglund and Peterson, 1983]. The input/output processor allows technicians to interface with the system via video display units and a master control center.

SOFTWARE REPLACEMENT

With the 5ESS switch, replacement of the resident software and/or hardware is done in real time to minimize the impact on customer service and call processing. A conversion process occurs when one generation of the switch's features is upgraded to accept new hardware or software. The generation change marks a change in the *generic*. The conversion process is referred to as a *generic retrofit* [Bauer, Croxall, and Davis, 1985]. The generic retrofit capability in the 5ESS switch provides the method and support software for replacing the resident software, the associated databases, and the read-only memory firmware for the switching modules, while maintaining adequate service and reliability.

Several factors must be taken into consideration before a 5ESS switch is retrofitted. First, to minimize impact on customers, retrofitting of software and most hardware occurs at separate times in the 5ESS switch. Consequently, either the old hardware must be compatible with the new software or the new hardware must be compatible with the old software.

Second, due to the distributed processor architecture of the 5ESS system, there are periods of time when the software release in the administrative module is different from that in one or more switching modules. The messages that pass between processors fall into three categories: (1) messages that must pass through regardless of software release, (2) messages that involve the switching module read-only memory software, and (3) all other messages. Messages in the first category establish the links between the administrative module and switching modules and are needed so that other messages can be exchanged. The second category consists of those messages related to initializing the switching modules with data.

A third consideration is evolution of the two databases that exist in the 5ESS switch. The Office Dependent Data contains such information as routing information and subscriber information. The Equipment Configuration Data contains information about the 3B20D hardware and peripheral equipment, such as the video display units and disk drives. Each exchange customizes these two databases to fit the particular configuration. Due to structural changes that are needed to support new features, the databases must be evolved from a format compatible with the old software release to one compatible with the new software release. The 5ESS system's generic retrofit capability supports the evolution of these customized databases.

Software Replacement Processes

The 3B20D/UNIX* RTR system provides system update facilities to support the introduction of new versions of UNIX RTR and application software [Wallace and Barnes, 1984]. The system update software provides a flexible mechanism that can be used by a variety of applications, including the 5ESS switch and allows the inclusion of application-dependent processing. The system update software places all new generic data into the system and makes final preparations prior to initializing the system from the new generic. After initialization, complete propagation of the new generic into the system occurs.

During each step of the update process, the system update software provides an opportunity

* UNIX is a registered trademark of UNIX System Laboratories, Inc.

for application-dependent processing by transferring control to an application process. To regain control, the system update process monitors the application-dependent process. Each time the application-dependent process is invoked, the system update process waits a specified time for its completion. The application process sends an acknowledgment message to the system update process that indicates the amount of time it requires for processing. If the application-dependent process does not finish within the allocated time, the system update process sends a software termination signal to the application process, informs the office technicians of any error, and stops the stage.

A separate process exists for each step of the update procedure. Communication between the system update processes is accomplished through use of a binary log file. The first entry in the log file specifies the number of entries in the file. Each system update process will write beginning, ending, and application-dependent entries in the log file. Information contained in the beginning and ending entries for each process includes the date and time of the entry, the process identification, and the type of update process that is making the entry. In addition, the ending entry contains the completion code of the stage, which indicates success or failure, and error codes that provide more specific information about what errors occurred.

The application-dependent process maintains an ASCII log file. An entry for each stage of the retrofit exists in the application-dependent process's log file. Each entry specifies the stage of the retrofit, the starting and ending times of the stage, and the completion code that indicates the type of error, if any, that occurred. The log file also contains information about processing specific to each stage.

Stages in 5ESS Switch Software Replacement

Replacement of software in the 5ESS switch consists of four major stages: advance preparation, preparation, initialization, and evaluation (Figure 8–32). The advance preparation stage occurs

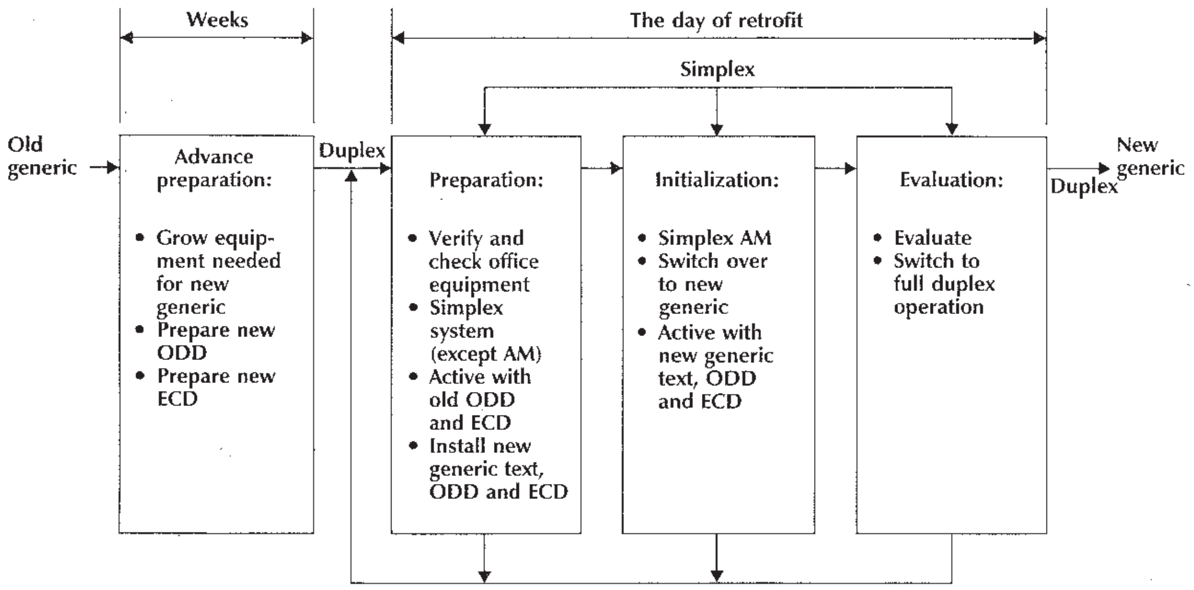

Weeks
The day of retrofit
Simplex
Old generic
Duplex
New generic
Duplex
Advance preparation:
• Grow equipment needed for new generic
• Prepare new ODD
• Prepare new ECD
Preparation:
• Verify and check office equipment
• Simplex system (except AM)
• Active with old ODD and ECD
• Install new generic text, ODD and ECD
Initialization:
• Simplex AM
• Switch over to new generic
• Active with new generic text, ODD and ECD
Evaluation:
• Evaluate
• Switch to full duplex operation

FIGURE 8–32 *5ESS switch retrofit stages* weeks prior to the actual day of retrofit and consists of growing in equipment needed for the new generic and preparing the new databases. The system remains in duplex operation during this time. The preparation, initialization, and evaluation stages occur on the day of retrofit. The system is simplexed during the preparation stage and is duplexed after evaluation of the new generic. After verifying and checking the office equipment for reliability, the new generic text and databases are installed. The system is then initialized and evaluated. During these stages of software replacement, the exchange is able to back out any changes made and revert to the old software if the need should arise. The UNIX RTR system update and 5ESS switch application-dependent software are utilized during the preparation, initialization, and evaluation stages.

Advance Preparation Stage. During the advance preparation stage, the exchange is evaluated to verify that it is properly equipped to perform a retrofit. At this time, any hardware that is required for the new generic to be operational is grown into the system. Additional memory for the administrative module and switching modules is also grown if needed. Also during this stage, a series of tests designed to diagnose and verify the operation of devices and equipment used to read in and store the new generic, equipment units essential to the retrofit, and hardware and software units needed to recover the system from an outage are executed. This testing ensures that there is a low probability of trouble attributable to existing system faults in subsequent stages of the retrofit.

Evolution of the office-dependent data (ODD) [Barclay, Dossey, and Nolan, 1986] and Equipment Configuration Data (ECD) databases takes place during this time interval. Database evolution is the process of converting a database used by one generic to a format required by a different generic. The ODD stores most of the 5ESS switch office data and contains information on subscriber lines, trunks, routing, and features. Information about the 3B20D hardware and 5ESS switch terminal configurations is stored in the ECD. While the ECD resides strictly in the administrative module, the ODD is distributed among the administrative module and switching modules. Evolution is required when structural changes occur in the databases (for example, when relations or attributes are added in the new database). Both the ODD and ECD databases are evolved off site.

Two dumps, the preliminary and final, are taken of the ODD and ECD. The preliminary dump is used to identify major inconsistencies in the databases. Fixes from the preliminary dump are then applied to the databases, and a final dump is taken and used to create the new generic's databases.

Office Dependent Data Evolution. The first step in ODD evolution is to back up the database to ensure that the disk copy is identical to the copy in main memory. Backups of the ODD for evolution cause changes made to the database to be logged so that they can be later evolved and applied to the new ODD. The ODD for the administrative module and all switching modules are copied to tape and shipped to a center responsible for evolving the databases. At the center, the ODD is decompiled from binary form to the original ASCII input. The ASCII input is then evolved to the new generic format and compiled into binary. The evolved ODD is then shipped back to the exchange.

The new generic database is created from a *snapshot* of the ODD. While the old generic database is being evolved to the new generic format, changes are continuously being made to the database in the exchange. Changes to the ODD are made by the exchange (recent changes, RCs) [Fuhrer, Shen, and Yates, 1986] or originated by the customer (customer originated recent changes, CORCs). An example of a CORC is when someone forwards his or her incoming calls to a different number.

At the same time that an RC or CORC is being applied to the database, it is doubly logged for recovery reasons [Locher, Pfau, and Tietz, 1986] and for retrofit purposes. The retrofit log file is in binary format and contains the database relation name and the updated tuple data. Each time the ODD is backed up on disk, the CORCs are automatically flushed from the administrative module and switching modules and are appended to the retrofit log files. Each time a relation gets inserted, updated, or deleted, the relation is logged in the switching module's memory buffers. The CORCs are sorted by timestamp; only the most recent transaction of a particular CORC is retained. The buffers are then shipped to the administrative module and will be written to disk the next time a database backup occurs. At the time of retrofit, these logged changes are evolved to a format compatible with the structure of the new generic's database. These evolved changes are then applied to the new database once the system is initialized on the new generic. Both the old and new generic databases are synchronized (Figure 8–33).

Equipment Configuration Data Evolution. A copy of the exchange's ECD is made to tape and is sent to an off-site processing center for evolution (Figure 8–34). Off-line tools are provided to extract certain exchange options from the ECD automatically. These options are applied to a *base* ECD that contains data items common to all 5ESS systems with the same generic to obtain a customized database. A process in the administrative module dumps the ECD, converts it to ASCII format, and loads it onto tape. A process on an off-site processor extracts the options from the tape and applies them to new base ECD via customization scripts. Since some options may not be customized in the new database, an audit report is sent, along with the new database, to the exchange. The audit report tells which fields have not been customized and allows the technicians to customize the fields after initialization on the new generic.

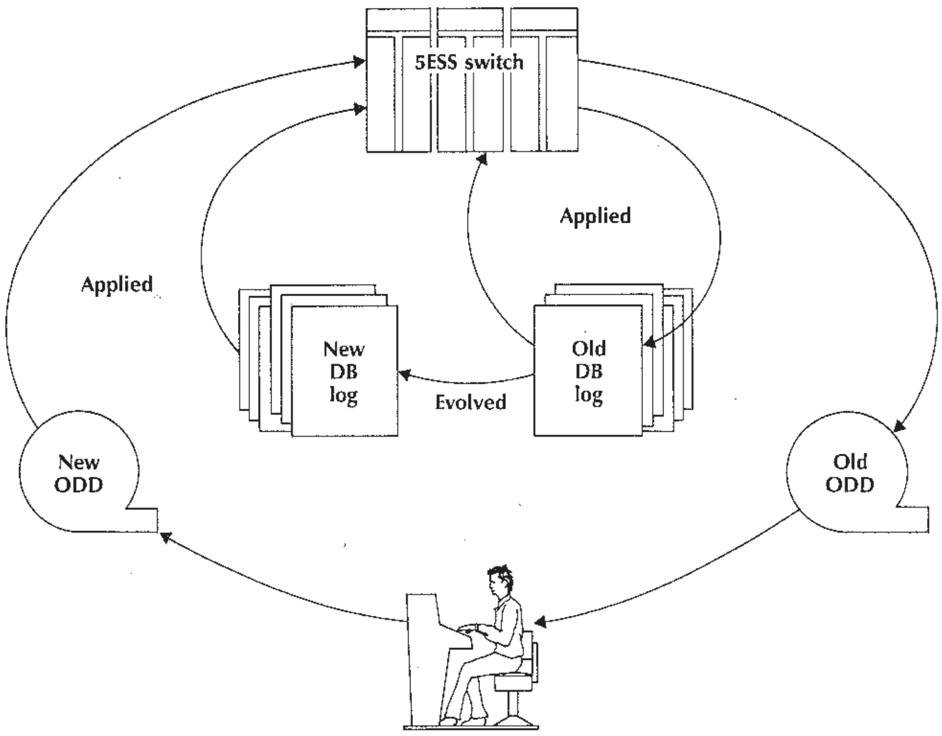
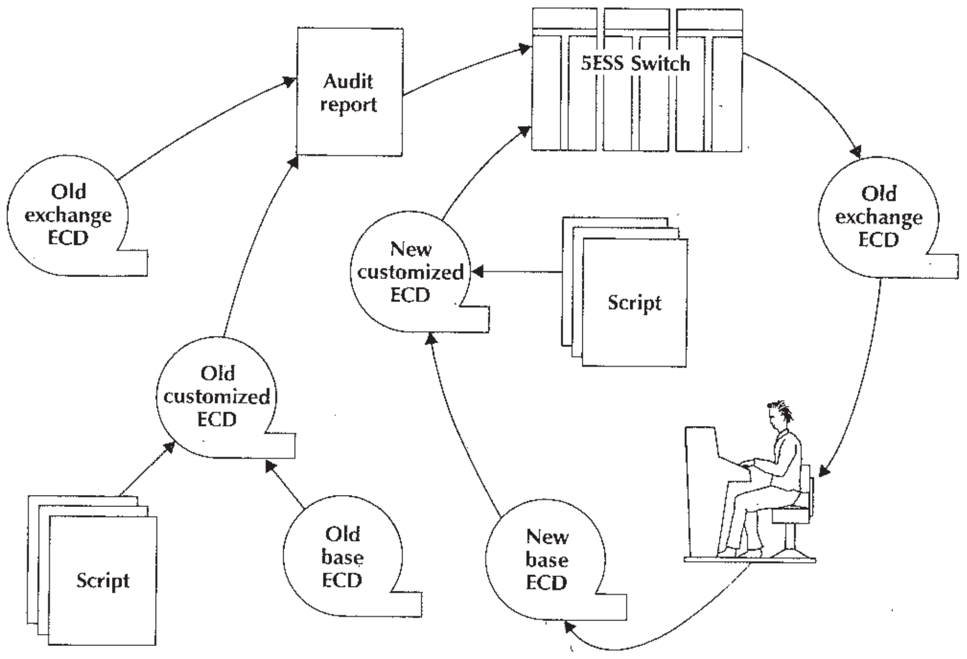


FIGURE 8–33 *Customer-originated recent changes and recent changes reapplications*

FIGURE 8–34
Equipment configuration data evolution



Preparation. This stage prepares the exchange for the actual retrofit and begins during the morning of the planned cutover to the new generic. The exchange verifies that it has installed the latest software and hardware updates and that all temporary overwrites to main memory are installed permanently or removed. The system is then backed up and it is verified that the exchange is running in duplex configuration (Figure 8–35). Diagnostics on equipment that is critical for retrofit are executed during this stage. At this time, billing data are collected and transmitted to a central location over a data link or are dumped to tape.

After the technician enters the command to start the retrofit, the application process sets the generic retrofit indicator on the maintenance control center and creates its log file. During this stage, the new generic and associated databases are read in from tape onto the secondary set of disks (Figure 8–36). The system update process determines the disks for which a set of tapes is targeted. It then removes the disks from service and marks the system off line in the ECD. This prevents restoral of the disks and will protect the new generic data from being overwritten. The remaining active disks continue to serve the live office; changes made to the active disks are not made to the off-line disks. The disks will remain simplex until the system is committed to the new generic.

A special device file provides mapping and allows the system update process to access the off-line disks. Each sequence of tapes contains a volume table of contents that defines which partitions the remaining data belong in. The offset of the data within the disk partition is determined from the physical disk address contained in the tape header and the starting address of the partition obtained from the volume table of contents.

During this stage, the system update process writes several types of log file entries. The beginning entry also includes the path name of the input device, the path name of the destination

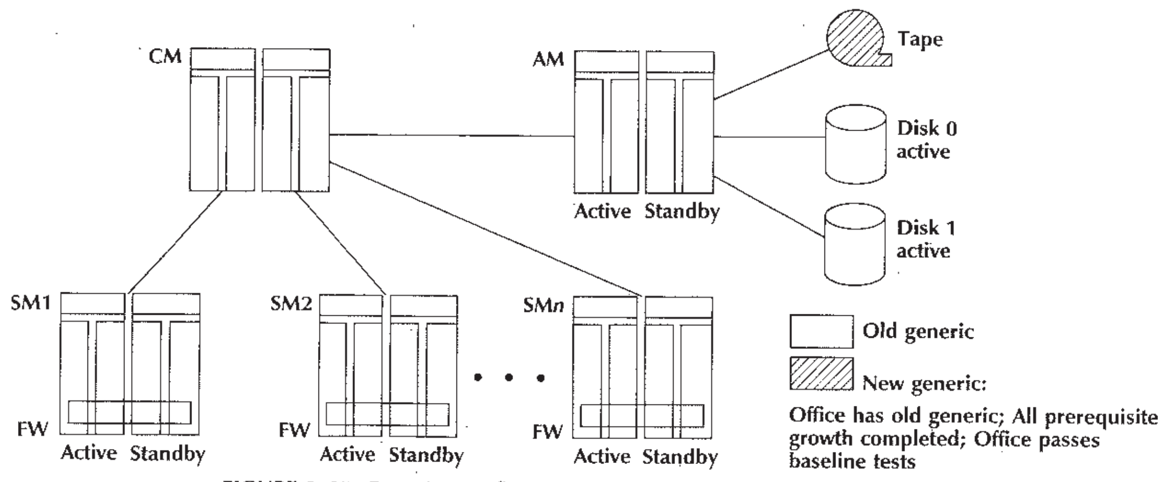


FIGURE 8–35 *Generic retrofit: Starting point*

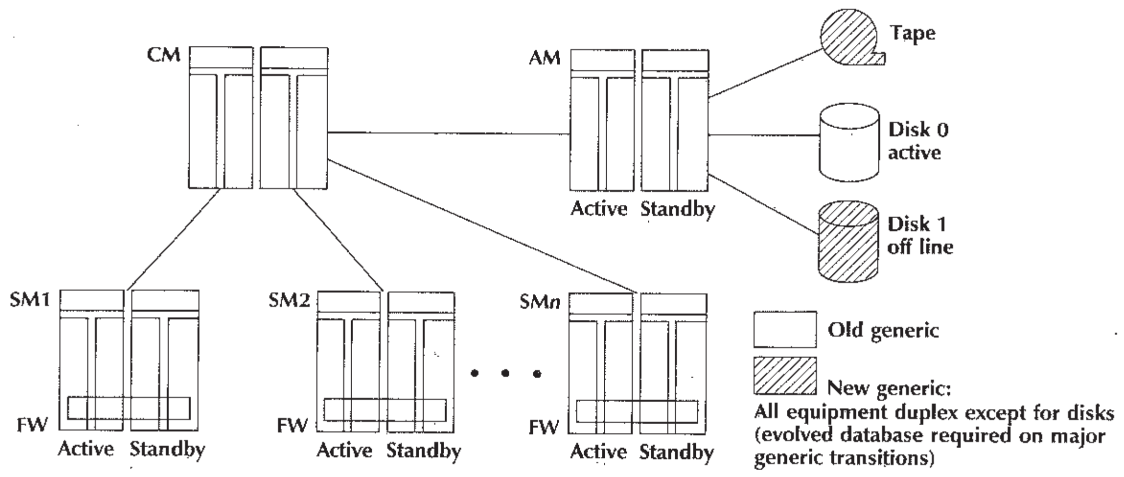


FIGURE 8–36 *Generic retrofit: Step 1, Load new generic from tape to off-line disk*

volume table of contents, and information that will be used to access the new generic's ECD and the old and new generic's system update log files. Log file entries are also made that indicate the disks that are being updated, the partitions on disk that are being updated, and information related to the tapes that are being read. Before the ending entry for this stage is written, an application-dependent process entry with the process identification number is written.

Files used by the old generic and needed by the new generic are copied from the active disk to the off-line disk by the application process. The destination off-line partition is mounted, and then the files are copied over. The application process writes an entry in its log file specifying the names of the files it tried to copy and whether it was successful in doing so. Once the new generic text and database tapes have been read, the duplicated module processors in each switching module are simplexed so that the generic dependent firmware may be replaced (Figure 8–37). One side of the switching module is forced active and continues processing calls. The other side of the switching module is unavailable and is pumped with the new generic, which is stored on the off-line disks (Figure 8–38).

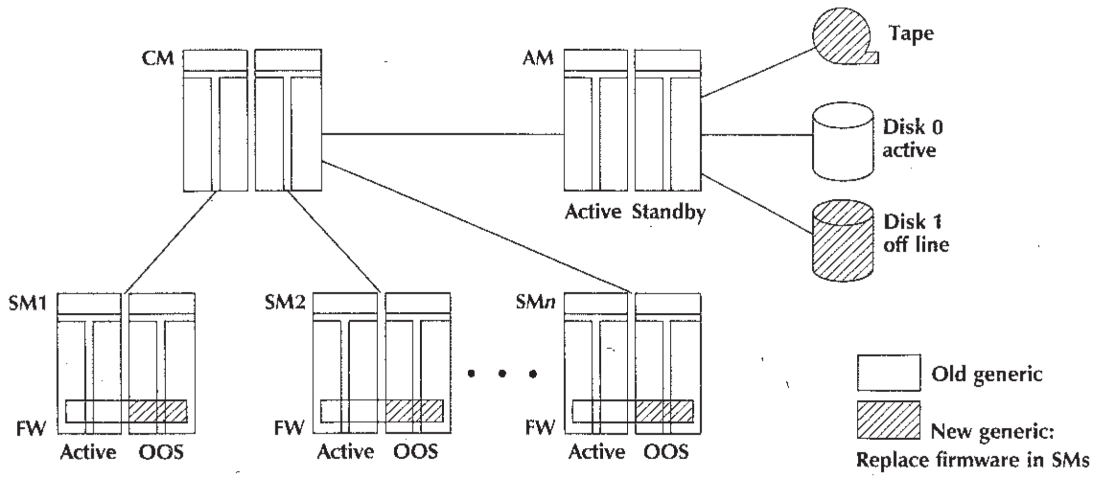


FIGURE 8–37 *Generic retrofit: Step 2, Simplex switching modules*

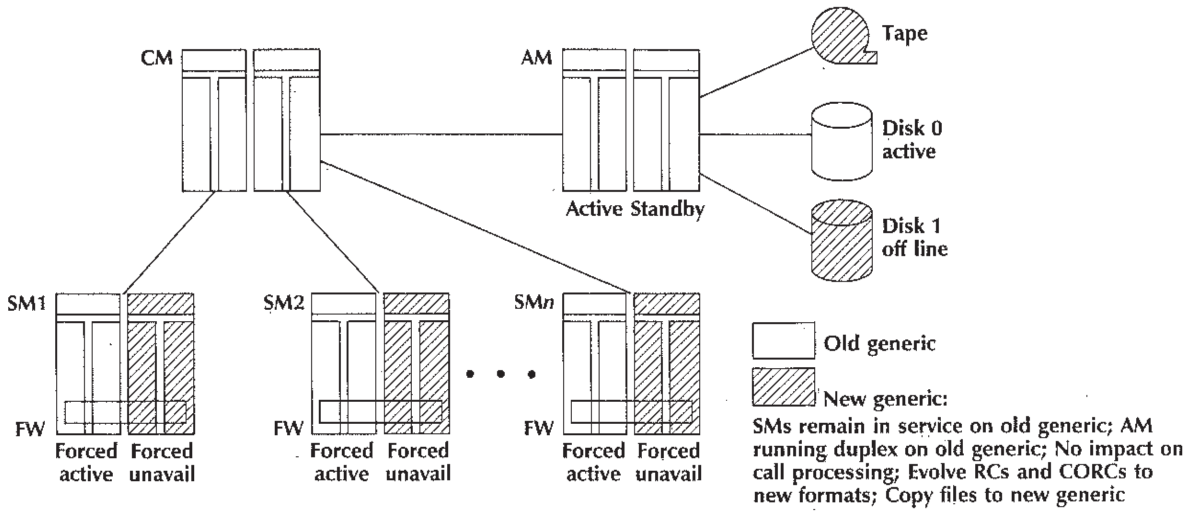

CM
AM
Tape
Disk 0 active
Disk 1 off line
Active Standby
SM1
SM2
SMn
FW
Forced active Forced unavail
Old generic
New generic:
SMs remain in service on old generic; AM running duplex on old generic; No impact on call processing; Evolve RCs and CORCs to new formats; Copy files to new generic

FIGURE 8–38 *Generic retrofit: Step 3, Pump off-line switching modules*

A process in the administrative module coordinates all requests to perform switching module pumps. It queues and prioritizes pump requests and invokes a special process to mount the off-line partitions to make the new generic software accessible for pumping. This special process uses the mknod and mount system calls to create special device files and mount point directories on the active disks. The partitions on the off-line disk are then mounted on the mount point directories. After the off-line partitions have been mounted, a third process in the administrative module controls the transfer of the data from disk to the switching modules via the 3B memory. A process in each switching module sets up and releases the data path for pumping. The process then performs hashsum checks of the memory on the unavailable side. Throughout this stage, the active processor of each switching module is still in service on the old generic, the administrative module is running duplex on the old generic, and there has been no impact on call processing.

At this point in time, the new generic's ODD is out of date with the ODD for the old generic. Changes that have been made to the ODD by the technician or the customer are logged and are evolved to a format compatible with the new generic database. The evolved RCs and CORCs are then copied by the application process from the active disk to the off-line disk.

Initialization Stage. During this stage, the switching modules are switched over from the old generic side to the new generic side and are initialized. The application process is responsible for changing the switching modules from the old generic side to the new one. The application log file entry for this stage contains information about whether the switch was successful for each switching module.

Messages other than ones used to establish the links between the administrative module and the switching modules are throttled in the communication module. The administrative module is the last processor to be initialized, since it needs to retain control over switching the switching modules to the new generic and to monitor their progress. Limited verification of the new ODD is done prior to initializing the switching modules on the new generic.

To protect the disks containing the old generic, the system update process accesses the new ECD on the off-line disks and marks the active disks off line. After the system is initialized, the disks containing the new generic will be active and the disks containing the old generic will be off line (Figure 8–39). The application and system update processes copy over their log files at this time. The system update process then requests the technician to initialize the system using the new generic disks. At any point in the retrofit, the exchange personnel can decide to return to the old generic and databases. The old software is available until the last stage in retrofit; thus, it is a very short process to abort the retrofit.

Evaluation Stage. Operation on the new generic is verified during this stage. Prior to duplexing the disks and switching modules, the technicians want to ensure that the new system is operational. The administrative module is restored to duplex operation. The new ODD is brought up to date with the old database by applying the evolved changes. The length of this stage, during which call processing is unaffected, is determined by local practice (Figure 8–40).

When the technicians are satisfied with the stability of the new system, the exchange is committed to full duplex operation. The new firmware is loaded in the side of the switching modules containing the old firmware, and both the switching modules and disks are restored to duplex operation (Figure 8–41). The system update process marks the off-line disks out of service in the ECD so that UNIX RTR facilities can be used to restore the disks.

The application process is responsible for checking that the switching modules are duplexed and for clearing the generic retrofit indicator on the maintenance control center. It prints out and then destroys its log file. At the completion of this stage, the system update process removes

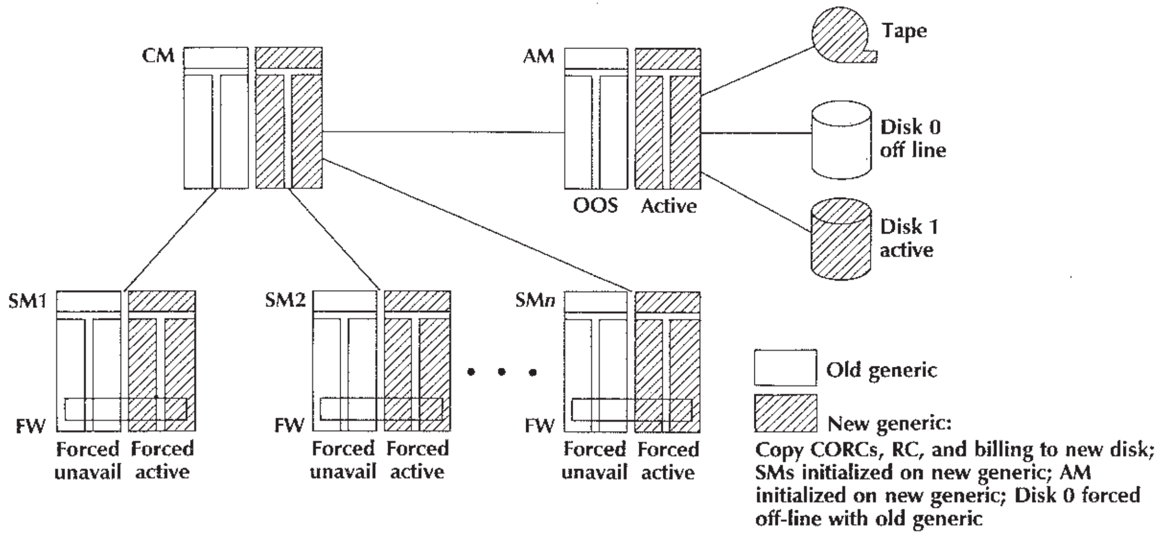


FIGURE 8–39 *Generic retrofit: Step 4, Initialization*

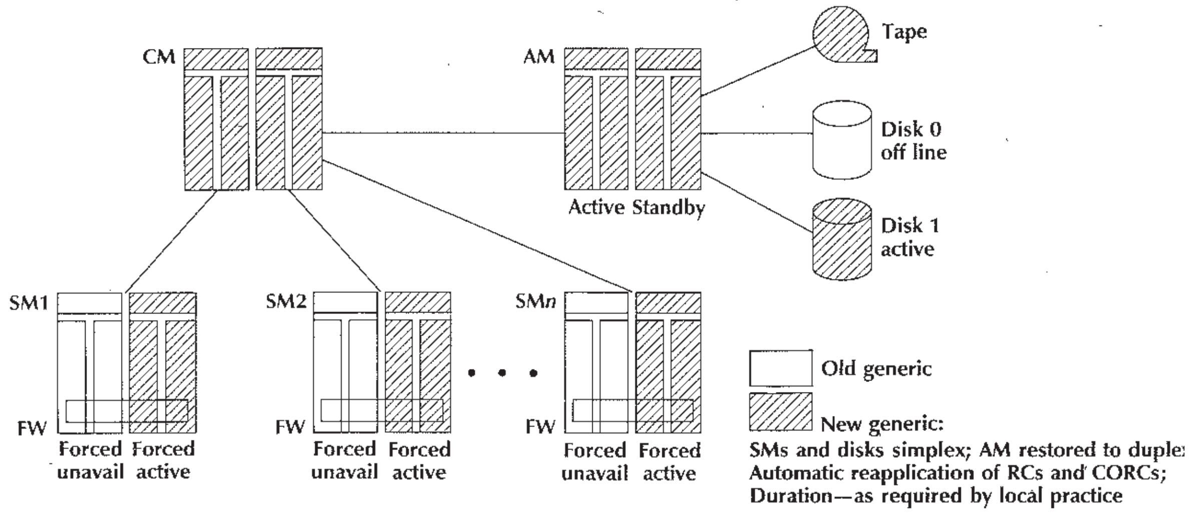


FIGURE 8–40 *Generic retrofit: Step 5, Verify operation on new generic*

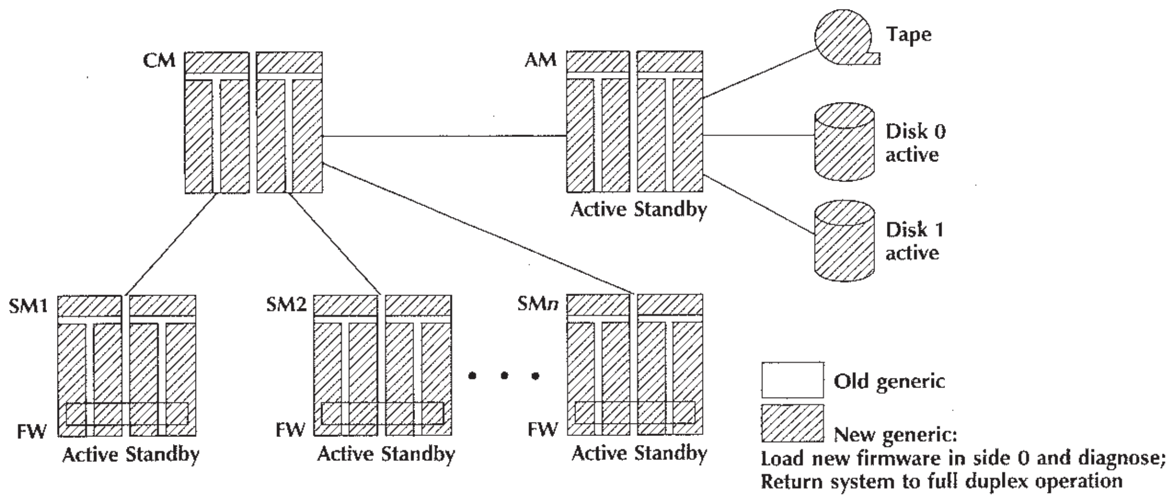


FIGURE 8–41 *Generic retrofit: Step 6, Commit and go full duplex*

its log file, since the update is finished and the system update processes no longer need to communicate.

In rare instances, the exchange personnel will decide to return to the old generic. This is simple to do prior to duplexing the disks, and a backout procedure is provided if necessary. The retrofit process is reversed and the disks that contain the old generic are re-initialized. The disks that contain the new generic are marked inaccessible and are then overwritten with the old generic.

SUMMARY

A much less desirable alternative to the generic retrofit process is to bring the system to a stop, load in new tapes, and then undergo a full initialization of the system. Long system downtimes would result from using this method, and there would be no mechanism for evolving the databases or recent changes. In addition, system integrity would not be guaranteed.

Generic retrofit takes advantage of the duplicated nature of the 5ESS switch in order to minimize impact on call processing and system reliability. It provides a way to read in the new software and databases while the switch is still running on the old generic. Call processing is uninterrupted during most of the initialization of the switching modules with the new generic.

All hardware growth and software changes occur at separate times. Hardware that is essential for the new generic to be operational is grown prior to retrofit and must be compatible with the new generic.

Because structural changes may occur in the databases, they are evolved from their old generic format to one that is compatible with the new generic. Changes made to the ODD are logged and automatically applied to the new generic database. The retrofit process provides databases compatible with the new generic that contain basically the same information as the old.

The architecture of the 5ESS system enables a smooth transition to a new generic that contains new features and hardware without a noticeable service interruption to subscribers. The process has been well tested and has been very successful. As of August, 1987, there were 137 exchanges retrofitted from the 5E1 to the 5E2(1) generic, 215 exchanges retrofitted from the 5E2(1) to the 5E2(2) generic, and 70 exchanges retrofitted from the 5E2(2) to the 5E3(1) generic.

REFERENCES

Allers et al., 1983; Anderson et al., 1987; Barclay, Dossey, and Nolan, 1986; Bauer, Croxall, and Davis, 1985; Byrne and O'Reilly, 1985; Davis et al., 1981; 5ESS, 1986; Fuhrer, Shen, and Yates, 1986; Haglund and Peterson, 1983; Locher, Pfau, and Tietz, 1986; Smith and Andrews, 1981; Toy and Gallaher, 1983; Wallace and Barnes, 1984.

THE TANDEM CASE

Fault Tolerance in Tandem Computer Systems

JOEL BARTLETT, WENDY BARTLETT, RICHARD CARR, DAVE GARCIA, JIM GRAY, ROBERT HORST, ROBERT JARDINE, DOUG JEWETT, DAN LENOSKI, AND DIX McGUIRE

The increasing trend for businesses to go on line stimulated a need for cost-effective computer systems with continuous availability [Katzman, 1977]. The strongest demand for general-purpose, fault-tolerant computing was in on-line database transaction and terminal-oriented applications. In the early 1970s, vendors and customers demanding continuous availability configured multiprocessor systems as hot standbys (see Figure 8–42). This configuration preserved previous development effort and compatibility by introducing devices, such as I/O channel switches and interprocessor communications adapters, to retrofit existing hardware. These architectures, however, still contained many single points of failure. For example, a power supply failure in the I/O bus switch, or an integrated circuit failure in any I/O controller on the I/O bus switch channel, would cause the entire system to fail. Other architectures used a common memory for interprocessor communications, creating another single point of failure. Typically, these architectures did not even approach the problems of on-line maintenance, redundant cooling, or a power distribution system that tolerates brownout conditions. Furthermore, these systems lacked thorough data integrity features, leading to problems in fault containment and possible database corruption.

As late as 1985, conventional, well-managed, transaction-processing systems failed about once every two weeks for about an hour [Mourad and Andrews, 1985; Burman, 1985]. This failure rate translates to 99.6 percent availability, a level that reflects a considerable effort over many years to improve system availability. When the sources of faults were examined in detail, a surprising picture emerged: Faults come from hardware, software, operations, maintenance, and the environment in about equal measure. Hardware could operate for two months without generating problems; software was equally reliable. The result was a one-month mean time between failures (MTBF). But if operator errors, errors during maintenance, and power failures were included, the MTBF fell below two weeks.

In contrast, the goal of Tandem is to build systems with a MTBF measured in years*—more than two orders of magnitude better than conventional designs. The key design principles of the system were, and still are, the following:

- *Modularity*: Both hardware and software are based on modules of fine granularity that are units of service, failure, diagnosis, repair, and growth.

* The original goal was to build a system with 100-year MTBF.

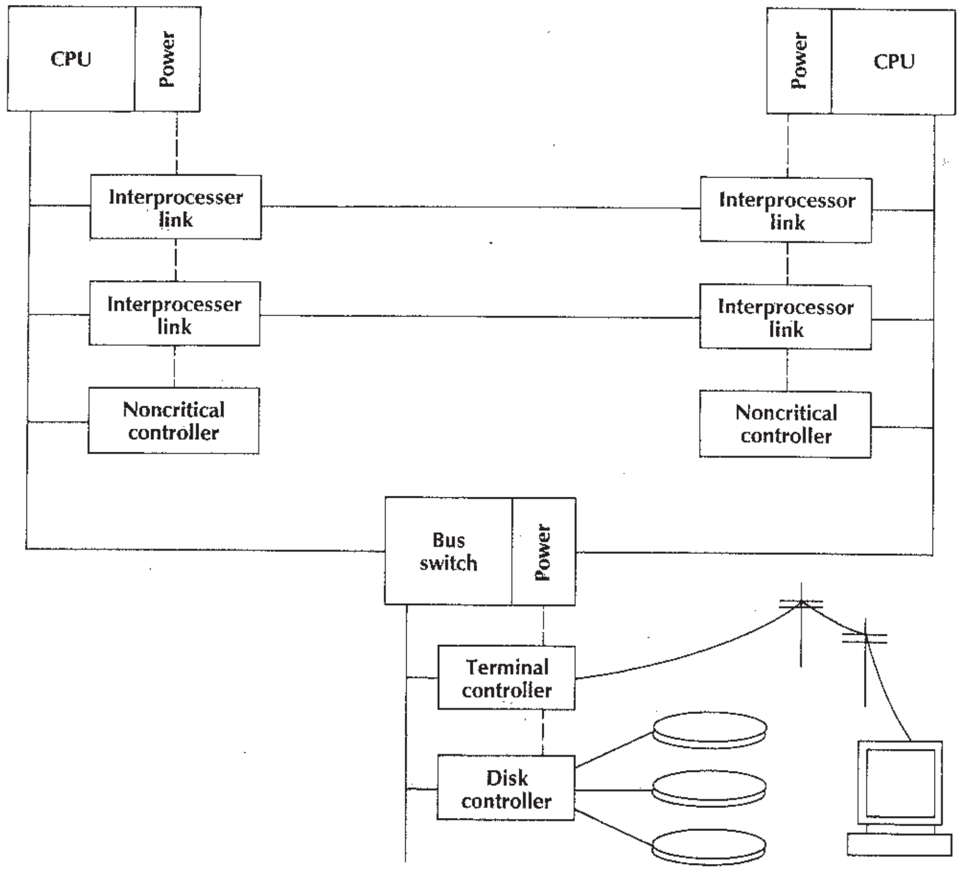


FIGURE 8–42
An example of early fault-tolerant architectures

- *Fail-Fast Operation*: Each module is self-checking; when it detects a fault, the module stops.
- *Single Fault Tolerance*: When a hardware or software module fails, another module immediately takes over the failed module's function, giving a mean time to repair measured in milliseconds. For processors or processes, this takeover means that a second processor or process must exist. For storage modules, it means that the modules and the paths to them are duplexed.
- *On-line Maintenance*: Hardware and software can be diagnosed and repaired while the rest of the system continues to deliver service. When the hardware, programs or data are repaired, they are reintegrated without interrupting service.
- *Simplified User Interfaces*: Complex programming and operations interfaces can be a major source of system failures. Every attempt has been made to simplify or automate interfaces to the system.

This case study presents Tandem NonStop and Integrity systems, viewed from the perspective of these key design features.

HARDWARE

Multiple hardware modules and multiple interconnections among those modules provide a basis for fault-tolerant operation. Two modules of a certain type are generally sufficient for hardware fault tolerance because the probability of a second independent failure during the repair interval of the first is extremely low. For instance, if a processor has an MTBF of 10,000 hours (about a year) and a repair time of 4 hours, the MTBF of a dual-path system increases to about 10 million hours (about 1000 years). If more than two processors were added, the further gains in reliability would be obscured by system failures related to software or system operations.

• *Modularity*: Modularity is important to fault-tolerant systems because individual modules must be replaceable on line. Keeping modules independent also makes it less likely that a failure of one module will affect the operation of another module. Increasing performance by adding modules allows customers to expand the capacity of critical systems without requiring major outages to upgrade equipment.

• *Fail-Fast Logic*: Fail-fast logic is defined as logic that either works properly or stops. Fail-fast logic is required to prevent corruption of data in the event of a failure. Hardware checks (including parity, coding, and self-checking), as well as firmware and software consistency checks, provide fail-fast operation.

• *Serviceability*: As mentioned before, maintenance is a source of outages. Ideally, the hardware should have no maintenance. When maintenance is required, it should require no special skills or tools.

• *Price and Price/Performance*: Commercial pressures do not permit customers to pay a high premium for fault tolerance; if necessary, they will use ad-hoc methods for coping with unreliable, but cost-effective, computers. Vendors of fault-tolerant systems have no special exemption from the requirement to use state-of-the-art components and architectures, frequently compounding the complexity already required by fault tolerance.

Hardware Architecture

The Tandem NonStop computer system was introduced in 1976 as the first commercial fault-tolerant computer system. Its basic architecture is shown in Figure 8–43. The system includes from 2 to 16 processors, connected by dual buses collectively known as the Dynabus interprocessor bus. Each processor has its own memory, containing its own copy of the operating system. The processors communicate with one another through messages passed through the Dynabus mechanism. The system can continue operation despite the loss of any single component. Each processor has its own input/output bus. Dual-ported controllers connect to I/O buses from two different processors. An ownership bit in each controller selects which of its ports is currently the primary path. When a processor or I/O bus failure occurs, all controllers that are designated as primary on that I/O bus switch to their backup paths. The controller configuration can be arranged so that in a multiprocessor system, the failure of a processor causes that processor's I/O workload to be spread out over the remaining processors. All subsequent systems have been upward-compatible with this basic design.

Processor Modules

The primary components of a system are its processor modules, each of which includes an instruction processing unit (IPU), memory, I/O channel, and Dynabus interprocessor bus interface. The design of the system's processor module is not much different from that of any traditional processor, with the addition of extensive error checking to provide fail-fast operation.

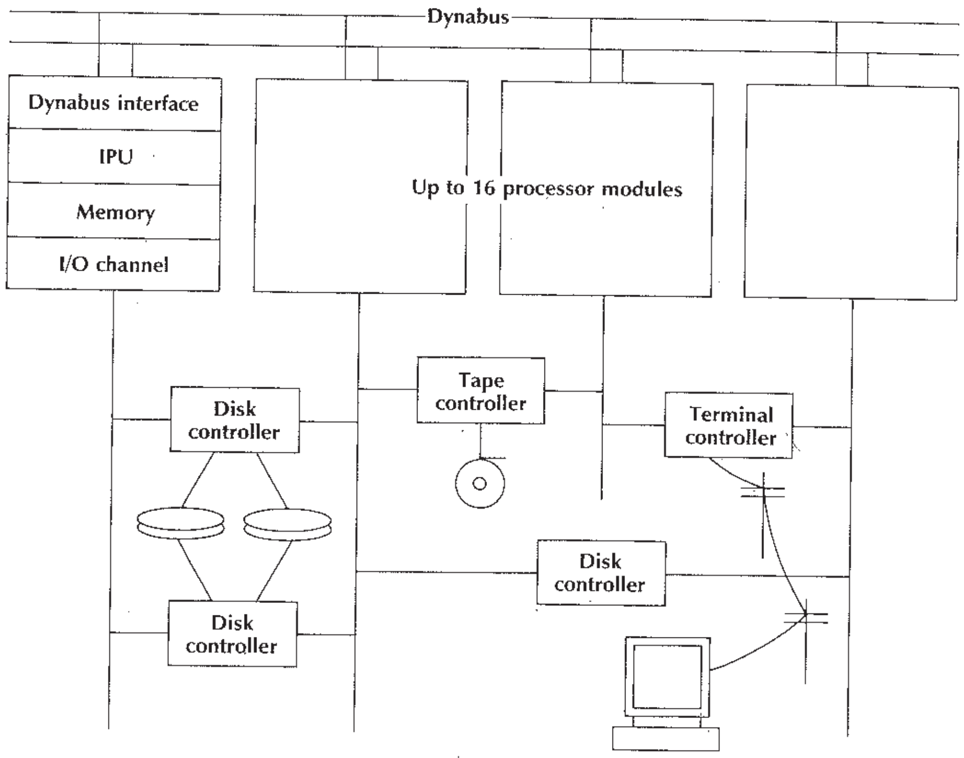


FIGURE 8–43 *Original Tandem system architecture, 1976*

Each processor operates independently and asynchronously from the rest of the processors. Another novel engineering requirement is that the Dynabus interfaces must prevent a single-processor failure from disabling both buses. This requirement focuses on the proper selection of a single component type: the buffer that drives the bus. This buffer must be well behaved when power is removed from the processor module to prevent errors from being induced on both buses.

The power, packaging, and cabling must also be carefully considered. Parts of the system are redundantly powered through diode ORing of two different power supplies. In this way, I/O controllers and Dynabus controllers tolerate a power supply failure. To allow on-line maintenance and modular growth, all boards are designed for *hot insertion;* that is, they can be inserted while the slot is powered. Battery backup power is standard in all systems. It preserves the system state for several hours in case of power failure.

The evolution of these processors is summarized in Table 8–5. Features common to all processors are described in the following sections. More details about the individual processors appear later in this case. Each processor provides a basic set of instructions that includes operations on bits, integers, decimal numbers, floating-point numbers, and character strings; procedure calls and exits; I/O operations; and interprocessor SENDS to streamline the performance of the message-based operating system. All instructions are 16 bits long. The Tandem NonStop I was designed as a stack-oriented, 16-bit processor with virtual memory. This instruction set has evolved to an upward-compatible, 32-bit addressing machine. Program binaries from the

TABLE 8–5 *Summary of Tandem NonStop processor evolution*

Processor / Feature	NonStop I (1976)	NonStop II (1981)	TXP (1983)	VLX (1986)	CLX 600 (1987)	CLX 700 (1989)	Cyclone (1989)	CLX 800 (1991)
Processor								
MIPS/IPU	0.7	0.8	2.0	3.0	1.0	1.5	10.0	2.2
Instructions	173	285	285	285	306	306	306	306
Technology	MSI	MSI STTL	MSI, Fast PAL	ECL, Gate array	Custom 2μ CMOS	Custom 1.5μ CMOS	ECL, Gate array	Custom 1μ CMOS
Cycle time	100 ns	100 ns	83 ns	83 ns	133 ns	91 ns	45 ns	61 ns
Microstore	—	8K × 32b	Two level: 8K × 40b, 4K × 84b	10K × 120b, dual	14K × 56b	14K × 56b	8K × 160b std, 8K × 160b pairs	14K × 56b
Cache (data and instructions)	—	—	64 KB, direct map	64 KB, direct map	64 KB, direct map	128 KB, direct map	2 × 64 KB, direct map	192 KB, direct map
Gates (approx)	20K	30K	58K	86K	81K	81K	275K	81K
Processor boards	2	3	4	2	1	1	3	1
Processors/ system	2–16	2–16	2–16	2–16	1–6	2–8	2–16	2–16
Memory								
Virtual	512 KB	1 GB	1 GB	1 GB	1 GB	1 GB	2 GB	1 GB
Physical	2 MB	16 MB	16 MB	256 MB	32 MB	32 MB	2 GB	32 MB
Per board	64 KB 384 KB	512 KB 2 MB 4 MB	2 MB 8 MB	8 MB 16 MB 48 MB	4 MB (on processor board) 8 MB	8 MB (on processor board) 8 MB	32 MB 64 MB	32 MB (on processor board)
Max. boards	2	2	4	2	1	1	2	0
Cycle time	500 nsec/2 B	400 nsec/2 B	666 nsec/8 B	416 nsec/8 B	933 nsec/8 B	637 nsec/8 B	495 nsec/16 B	414 nsec/8 B
Input Output								
Interprocessor bus speed	2 × 13 MB/ sec	2 × 13 MB/ sec	2 × 13 MB/ sec	2 × 20 MB/ sec	2 × 20 MB/ sec	2 × 20 MB/ sec	2 × 20 MB/ sec	2 × 20 MB/ sec
Channel speed	4 MB/sec	5 MB/sec	5 MB/sec	5 MB/sec	3 MB/sec	4.4 MB/sec	2 × 5 MB/sec	4.4 MB/sec

† Additional 225 nsec if CAM miss.

NonStop I will run on a Cyclone. The processor implementations have been fairly conventional, using a mix of special-purpose hardware for basic arithmetic and I/O operations, along with microcode to implement higher-level functions. Two novel features are the special hardware and microinstructions to accelerate the sending and receipt of messages on the Dynabus. The performance of these instructions has been a key component of the success of the message-based operating system.

Memory, as originally implemented, was designed to support a 16-bit minicomputer. In 1981, designers added a 32-bit addressing scheme to provide access to 4-MB code space (for users), multiple 127.5-MB data spaces (for users), 4-MB code space (for the operating system), and 1-GB (2 GB for Cyclone) of virtual data space (for the operating system). The code and data spaces in both the user and the system areas are logically separate from one another.

In order to make processors fail-fast, extensive error checking is incorporated in the design. Error detection in data paths typically is done by parity checking and parity prediction, while checking of control paths is done with parity, illegal state detection, and self-checking. Loosely coupling the processors relaxes the constraints on the error-detection latency. A processor is required to stop itself only in time to avoid sending incorrect data over the I/O bus or Dynabus. In some cases, to avoid lengthening the processor cycle time, error detection is pipelined and does not stop the processor until several clocks after the error occurs. Several clocks of error-detection latency are permitted in the architecture, but cannot be tolerated in systems in which several processors share a common memory. In addition, the true fail-fast character of all processors eliminates the need for instruction retry in the event of errors.

Dynabus Interprocessor Bus

The Dynabus interprocessor bus is a set of two independent interprocessor buses. All components that attach to either of the buses are kept physically distinct so that no single component failure can contaminate both buses simultaneously. Bus access is determined by two independent interprocessor bus controllers. Each of these controllers is dual-powered in the same manner as an I/O controller. The Dynabus controllers are not associated with, nor physically part of, any processor. Each bus has a two-byte data path and several control lines associated with it. No failed processor can independently dominate bus utilization upon failure because, to electrically transmit onto the bus, the bus controller must agree that a given processor has the right to transmit.

The original Dynabus, connecting from 2 to 16 processors, was designed with excess capacity to allow for future improvements in processor performance without redesign of the bus. The same bus is used on the NonStop II, introduced in 1980, and the NonStop TXP, introduced in 1983. The NonStop II and NonStop TXP processors can even plug into the same backplane to operate in a single system with mixed processors. A full 16-processor TXP system does not drive the bus near saturation. A new Dynabus was introduced with the VLX system. It provides peak throughput of 40 megabytes per second, relaxes the length constraints of the bus, and has a reduced manufacturing cost due to improvements in its clock distribution. It was, again, overdesigned to accommodate the higher processing rates predicted for future processors. The CLX and Cyclone systems also use this bus.

For any given interprocessor data transfer, one processor is the sender and the other is the receiver. To transfer data over the Dynabus interprocessor bus, the sending processor executes a send instruction. This instruction specifies the bus to be used, the intended receiver, and the number of bytes to be sent. Up to 64 kilobytes can be sent in a single send instruction. The sending processor continues to execute the send instruction until the data transfer is completed, during which time the Dynabus interface control logic in the receiving processor is storing the data in memory. In the receiving processor, this activity occurs concurrently with program execution. Error recovery action is taken in case the transfer is not completed within a specified timeout interval. In the Dynabus design, the more esoteric decisions are left to the software (for example, alternate path routing and error recovery procedures): Hardware, then, implements fault detection and reporting [Bartlett, 1978].

Fiber-Optic Extension Links

In 1983, a fiber-optic bus extension (FOX) was introduced to link systems together in a high-speed local network. FOX allows up to 14 systems of up to 16 processors each to be linked in a ring structure, for a total of 224 processors. The maximum distance between adjacent nodes is 1 kilometer on the original FOX and 4 kilometers with FOX II, which was introduced on the VLX processor. A single FOX ring can mix NonStop II, TXP, VLX, and Cyclone processors. The interconnection of systems by FOX links is illustrated in Figure 8–44. Each node in the group can accept or send data at rates of up to 4 MB/sec.

The FOX connection is based on a store-and-forward ring structure. Four fibers are connected between a system and each of its two neighbors. Each interprocessor bus is extended by a pair of fibers, allowing messages to be sent in either direction around the ring. The four paths provided between any pair of systems ensure that communication is not lost if a system is disabled (perhaps because of a power failure) or if an entire four-fiber bundle is severed.

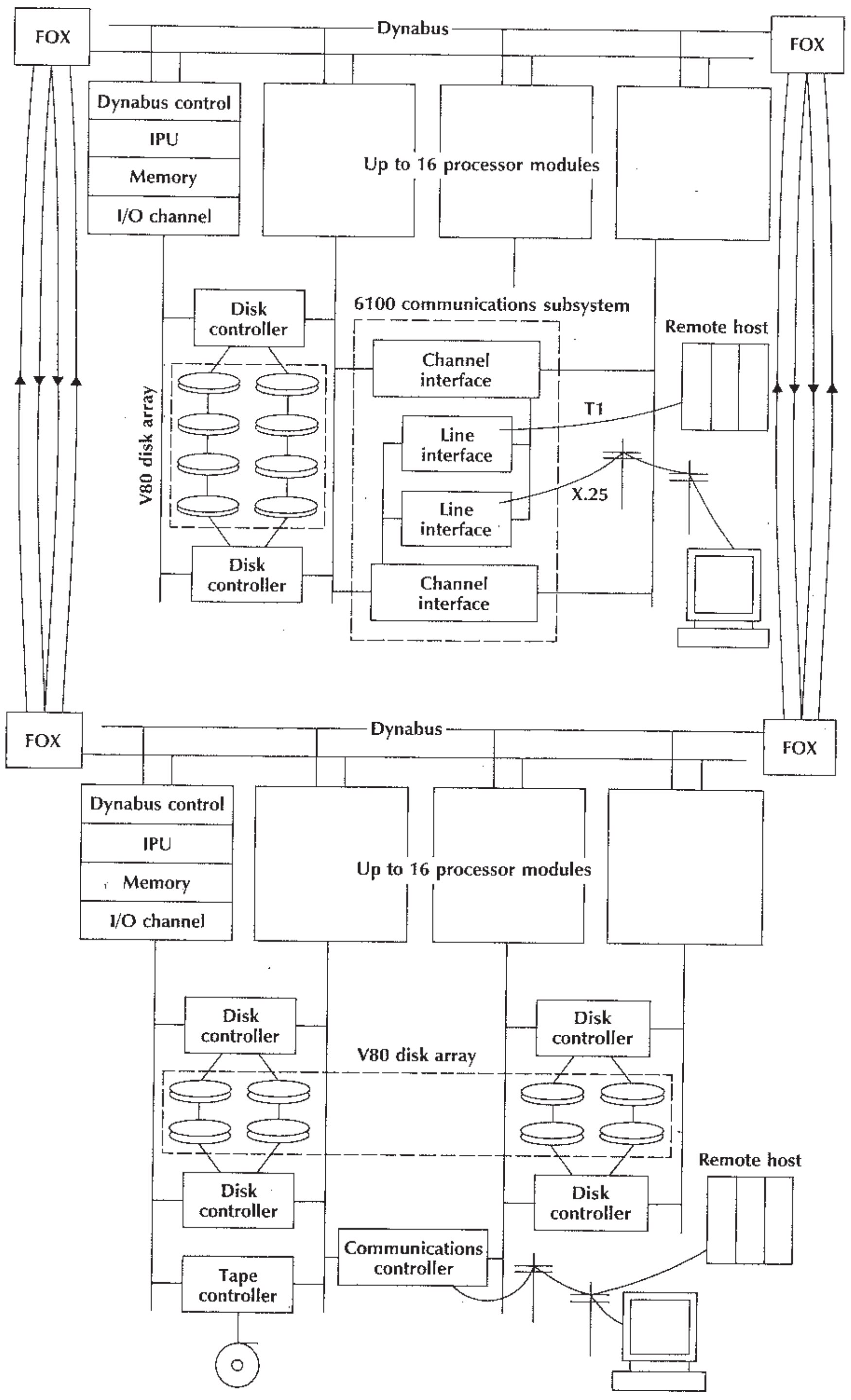
The ring topology also has advantages over a star configuration because a ring has no central switch that could constitute a single point of failure and because cable routing is easier with a ring than with a star. In a ring structure, bandwidth increases as additional nodes are added. The total bandwidth available in a FOX network depends on the amount of pass-through traffic. In a 14-node FOX ring, if each node sends to other nodes with equal probability, the network has a usable bandwidth of 10 MB/sec. When there is no pass-through traffic, the bandwidth increases to 24 MB/sec. Theoretically, an application generating 3 kB of traffic per transaction, at 1000 transactions per second, would require a FOX ring bandwidth of only 3 MB/sec. In this situation, the FOX network would use less than 30 percent of the total available bandwidth. Transaction processing benchmarks have shown that the bandwidth of FOX is sufficient to allow linear performance growth in large multinode systems [Horst and Chou, 1985; Englert, 1989].

Fiber-optic links were chosen to solve both technical and practical problems in configuring large clusters. Fiber optics are not susceptible to electromagnetic interference, so they provide a reliable connection even in noisy environments. They also provide high-bandwidth communication over fairly large distances (4 km/hop). This capability lessens the congestion in the computer room and allows many computers in the same or nearby buildings to be linked. FOX links allow computer sites to physically isolate nodes by housing them in different buildings, thereby providing a degree of fault isolation and protection against disaster. For example, a fire in the computer room in one building would not affect nodes in other buildings.

Dynabus+ Fiber-Optic Dynabus Extension

With the introduction of the Cyclone system in 1989, Tandem made two additional uses of fiber-optic links. Their use between peripheral controllers and I/O devices is described later in this case. Their use as an interprocessor link within a system (as compared with FOX, which is an intersystem link) is described here.

FIGURE 8–44
Tandem system architecture, 1990



Cyclone processors are grouped into sections, each containing up to four processor modules. The sections may be geographically distributed up to 50 meters. Within a section, the normal (backplane) Dynabus interface is used. Sections within a system are connected in a ring arrangement, similar to the FOX arrangement.

Individual Dynabus+ fiber-optic links are capable of 12.5-MB/sec bandwidth, a good match for the 20-MB/sec bandwidth of the Dynabus. While increasing performance was not the major design goal of Dynabus+, the design has resulted in additional aggregate interprocessor bus bandwidth, up to 160 MB/sec in a system. Each section contains its own Dynabus controllers, so message traffic local to the section can proceed concurrently with local traffic in other sections. In addition, intersection traffic can proceed concurrently on the multiple fiber-optic links.

The Dynabus+ system is transparent to all levels of software except the maintenance subsystem (described later in this case). The system is self-configuring when power is first applied: Maintenance processors within each cabinet determine the configuration and routing rules. In the event of a failure of any of the fiber-optic links or interface logic boards, the system reconfigures itself, establishes new routing paths, and notifies the maintenance subsystem. In addition, a VLX interface to Dynabus+ allows intermixing VLX and Cyclone processor modules within a system. This feature allows an existing VLX customer to add Cyclone processors to a system, providing a smooth upgrade path.

Evolutionary Changes

Processor architecture has evolved to keep pace with technology. These improvements include (1) expansion of virtual memory to 1 GB (NonStop II system) and to 2 GB (Cyclone system), (2) incorporation of cache memory (TXP system), (3) expansion of physical memory addressability to 256 MB (VLX system) and to 2 GB (Cyclone system), (4) incorporation of separate instruction and data caches (Cyclone system), (5) incorporation of superscalar architecture (Cyclone system) [Horst, Harris, and Jardine, 1990], and (6) incorporation of an independent instruction fetch unit with dynamic branch prediction (Cyclone system).

Technological improvements include evolution from core memory to 1-Mb dynamic RAM and evolution from Schottky TTL (NonStop I and II systems) to programmable array logic (TXP system) [Horst and Metz, 1984] to bipolar gate arrays (VLX and Cyclone system) to silicon-compiled custom CMOS (CLX system) [Lenoski, 1988].

The Tandem multiprocessor architecture allows a single processor design to cover a wide range of processing power. Having processors of varying power adds another dimension to this flexibility. For instance, for approximately the same processing power, a customer can choose a two-processor VLX system, a three-processor TXP system, or a four-processor CLX-700 system. Having a range of processors extends the range of applications from those sensitive to low-entry price to those with extremely high-volume processing needs. In a different performance range, the customer may choose a four-processor Cyclone system or a 16-processor VLX system.

Peripherals

In building a fault-tolerant system, the entire system, not just the processor, must have the basic fault-tolerant properties of dual paths, modularity, and fail-fast design, as well as good price/performance. Many improvements in all of these areas have been made in peripherals and in the maintenance subsystem. The basic architecture provides the ability to configure the I/O system to allow multiple paths to each I/O device. With dual-ported controllers and dual-ported peripherals, there are actually four paths to each device. When disks are mirrored, there are eight paths that can be used to read or write data.

In the configurations illustrated in Figure 8–44, there are many paths to any given disk. Typically, two controllers access each disk, and each controller is attached to two processor channels. Software is used to mirror disks; that is, data is stored on two disks so that if one fails, the data is still available on the other disk. Consequently, the data can be retrieved regardless of any single failure of a disk drive, disk controller, power supply, processor, or I/O channel.

The original architecture did not provide as rich an interconnection scheme for communications and terminals. The first asynchronous terminal controller was dual-ported and connected to 32 terminals. The terminals themselves were not dual-ported, so it was not possible to configure the system so that it would withstand a terminal controller failure without losing a large number of terminals. The solution for critical applications was to have two terminals nearby that were connected to different terminal controllers.

The 6100 Communications Subsystem. The 6100 communications subsystem, introduced in 1983, helped reduce the impact of a failure in the communications system. The 6100 consists of two dual-ported communications interface units (CIUs) that communicate with I/O buses from two different processors (see Figure 8–44). Individual line interface units (LIUs) connect to both CIUs and to the communications line or terminal line. With this arrangement, CIU failures are completely transparent, and LIU failures result in the loss of only the attached line or lines. An added advantage is that each LIU can be downloaded with a different protocol in order to support different communications environments and to offload protocol interpretation from the main processors.

The 6100 communications subsystem is configured to have up to 45 LIUs. Each LIU can support up to 19.2 Kb/sec of asynchronous communication or 64 Kb/sec of synchronous communication. Redundant power supplies and cooling fans provide an extra margin of fault tolerance and permit on-line replacement of components.

Disk Subsystem. Modularity is standard in peripherals. It is common to mix different types of peripherals to match the intended application. In on-line transaction processing (OLTP), it is desirable to select increments of disk capacity and of disk performance independently. OLTP applications often require more disk arms per megabyte than are provided by traditional large (14") disks. This requirement may result in customers' buying more megabytes of disk than they need in order to avoid queuing at the disk arm.

In 1984, Tandem departed from traditional disk architecture by introducing the V8 disk drive. The V8 was a single cabinet that contained up to eight 168-MB, 8" Winchester disk drives in six square feet of floor space. Using multiple 8" drives instead of a single 14" drive provided more access paths and less wasted capacity. The modular design was more serviceable because individual drives could be removed and replaced on line. In a mirrored configuration, system software automatically brought the replaced disk up to date while new transactions were underway.

Once a system can tolerate single faults, the second-order effects begin to become important in system failure rates. One category of compound faults is the combination of a hardware failure and a human error during the subsequent human activity of diagnosis and repair. The V8 reduced the likelihood of such compound hardware-human failures by simplifying servicing and eliminating preventative maintenance.

In fault-tolerant systems design, keeping down the price of peripherals is even more important than in traditional systems. Some parts of the peripheral subsystem must be duplicated, yet they provide little or no added performance. For disk mirroring, two disk arms give better read performance than two single disks because the seeks are shorter and because the read work is spread evenly over the two servers [Bitton and Gray, 1988; Bitton, 1989]. Write operations, on the other hand, do demand twice as much channel and controller time. Also, mirroring does double the cost per megabyte stored. To reduce the price per megabyte of storage, the XL8 disk drive was introduced in 1986. The XL8 had eight 9" Winchester disks in a single cabinet and had a total capacity of 4.2 GB. As in the V8 drive, disks within the same cabinet could be mirrored, saving the costs of added cabinetry and floor space. Also, like the V8, the reliable sealed media and modular replacement kept maintenance costs low.

The V80 disk storage facility replaced the V8 in 1988. Each of the V80's eight 8" disk drives has a formatted capacity of 265 MB. Thus, each cabinet can hold 2.7 GB of unformatted storage, or 2.1 GB of formatted storage. Externally, the V80 resembles the V8, housed in a single cabinet that occupies six square feet of floor space. The internal design of the V80, however, extends the capacity and reliability of the V8 with a fully checked interface to the drives. Furthermore, the design reduces by a factor of five the number of external cables and connectors between the storage facility and the control unit.

The disk drive interface is based on the emerging industry-standard IPI-2 interface design, which has parity checking on data and addressing to ensure the integrity of data and commands. (The previous SMD-based design provided only data parity.) IPI's parallel and batched data and command interface between the disks and their controller allow higher data transfer rates (2.4 MB/sec) and reduced interrupts. A radial connection between the controller and the drives eliminates possible drive interaction that could occur with conventional bus structures. The fivefold reduction in the number of external cables and connections is achieved by placing the control logic in the disk cabinet. Within the cabinet, a new interconnect design has reduced by a factor of five the number of internal cables and connections.

In 1989, the XL80 replaced the XL8 in similar fashion, doubling the storage capacity per drive and also moving to an IPI-2 storage interface. In addition, the XL80 cabinet contains sensors for inlet air temperature, power supply and board voltages, and fan operation; this information is polled periodically by the cabinet's maintenance subsystem and reported to the peripheral controller when an exception condition exists. A fully configured XL80 disk subsystem, including storage modules, power supplies, and cooling fans, appears in Figure 8–45.

Peripheral Controllers. Peripheral controllers have fail-fast requirements similar to processors. They must not corrupt data on either of their I/O buses when they fail. If possible, they must return error information to the processor when they fail. In terms of peripheral fail-fast design, the Tandem contribution has been to put added emphasis on error detection within the peripheral controllers. An example is a VLSI tape controller that uses dual, lock-stepped Motorola 68000 processors with compare circuits to detect errors. It also contains totally self-checked logic and self-checking checkers to detect errors in the ad-hoc logic portion of the controller. Beyond this contribution, the system software uses end-to-end checksums generated by the high-level software. These checksums are stored with the data and are recomputed and rechecked when the data is reread.

The single-board controller supporting the V80 and the XL80 disks uses CMOS VLSI technology. The controller is managed by dual, lockstepped Motorola 68010 microprocessors that provide sophisticated error-reporting and fault-isolation tools. The controller is contained on a single board. Thus, it requires only half the I/O slots of previous controllers.

Other efforts to reduce peripheral prices include the use of VLSI gate arrays in controllers to reduce part counts and improve reliability and the use of VLSI to integrate the stand-alone 6100 communications subsystems into a series of single-board controllers. The Tandem evolution of fault tolerance in peripherals is summarized in Table 8–6.

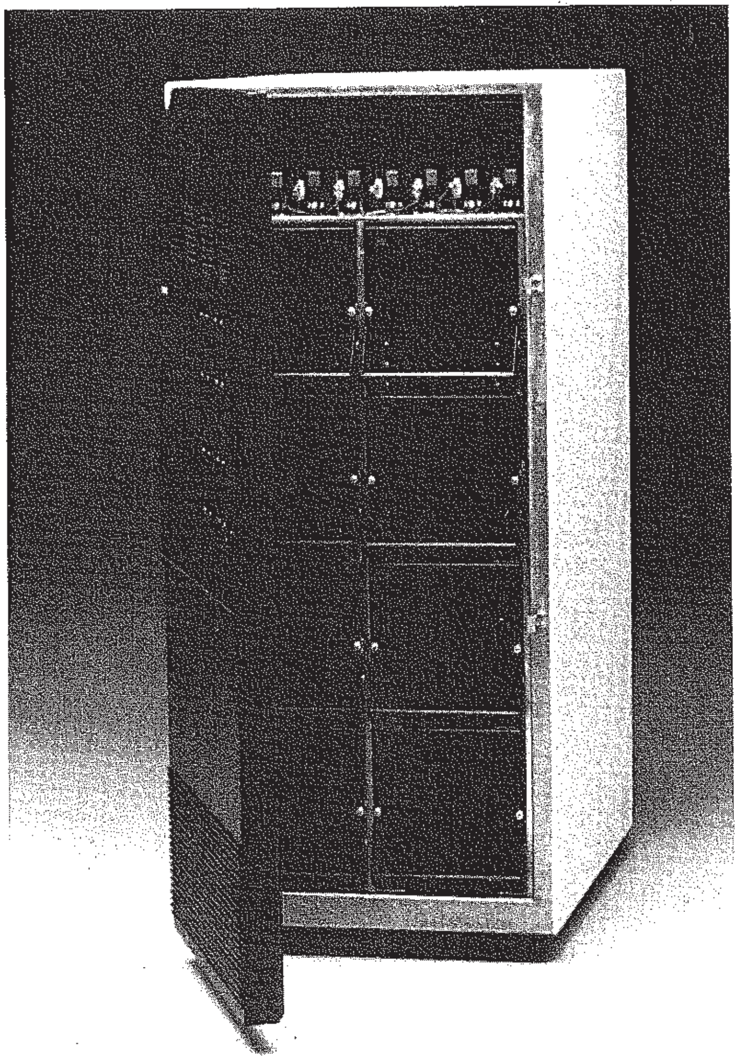


FIGURE 8–45
XL80 disk subsystem (front view)

PROCESSOR MODULE IMPLEMENTATION DETAILS

The following sections outline the implementation details of each of the Tandem processors summarized in Table 8–5.

NonStop I

The NonStop I processor module, introduced in 1976, included a 16-bit IPU, main memory, Dynabus interface, and an I/O channel. Physically, the IPU, I/O channel, and Dynabus control consisted of two PC boards that measured 16″ × 18″, each containing approximately 300 integrated circuit packages. These boards employed Schottky TTL circuitry. The processor module was

TABLE 8–6 *Tandem evolution of peripheral fault tolerance*

Year	Product	Contribution
1976	NonStop I system	Dual-ported controllers, single-fault tolerant I/O system
1977	NonStop I system	Mirrored and dual-ported disks
1982	INFOSAT	Fault-tolerant satellite communications
1983	6100 communications subsystem	Fault-tolerant communications subsystem
1983	FOX	Fault-tolerant, high-speed, fiber-optic LAN
1984	V8 disk drive	Eight-drive, fault-tolerant disk array
1985	3207 tape controller	Totally self-checked VLSI tape controller
1985	XL8 disk drive	Eight-drive, high-capacity/low-cost, fault-tolerant disk array
1986	TMDS	Fault-tolerant maintenance system
1987	CLX	Fault-tolerant system that is 98 percent user-serviceable
1988	V80 storage facility	Reduced disk cabling and fully-checked disk interfaces
1988	3120 disk controller	Totally self-checked VLSI disk controller
1989	XL80 storage facility	Reduced disk cabling, fully-checked disk interfaces, environmental monitoring within disk cabinet
1989	Fiber-optic interconnect for V80 and XL80	Reduced cabling to a minimum, reduced transmission errors

viewed by the user as a stack-oriented, 16-bit processor, with a demand paging, virtual memory system capable of supporting multiprogramming.

The IPU was a microprogrammed processor consisting of (1) an execution unit with ALU, shifter, register stack, and program counter, (2) a microprogram sequencer with 1024 32-bit words stored in ROM, (3) address translation maps supporting system code and data, and current user code and data segments, (4) main memory of up to 512 KB, (5) 96 KB memory boards with single-error correction and double-error detection, and (6) battery backup for short-term main memory ride-through of power outages of up to 4 hours.

The heart of the I/O system is the I/O channel. In the NonStop I, all I/O was done on a direct memory access basis. The channel was a microprogrammed, block-multiplexed channel; individual controllers determine the block size. The channel did not execute channel programs, as on many systems, but did transfer data in parallel with program execution. The memory system priority always permitted I/O accesses to be handled before IPU or Dynabus accesses. The maximum I/O transfer was 4 KB.

Dual-Port Controllers. The dual-ported I/O device controllers provided the interface between the NonStop I I/O channel and a variety of peripheral devices using distinct interfaces. While these I/O controllers were vastly different, depending on the devices to which they interfaced, there was a commonality among them that fitted them into the NonStop I architecture. Each controller contained two independent I/O channel ports implemented by IC packages that were physically separate from each other so that no interface chip could simultaneously cause failure of both ports. Logically, only one of the two ports was active. The other port was utilized only in the event of a path failure to the primary port. An ownership bit, as illustrated in Figure 8–46, indicated to each port if it was the primary port or the alternate.

Ownership changed only when the operating system issued a Take-Ownership I/O command. Executing this special command caused the I/O controller to swap its primary and alternate port designation and to do a controller reset. Any attempt to use a controller that was not owned by a given processor resulted in an ownership violation. If a processor determined that a given controller was malfunctioning on its I/O channel, it could issue a Disable-Port command that logically disconnected the port from that I/O controller. This disconnection did not affect the ownership status. Thus, if the problem was within the port, the alternate path could be used, but if the problem was in the common portion of the controller, ownership was not forced on the other processor.

Fault-Tolerant I/O Considerations. The I/O channel interface consisted of a 2-byte data bus and control signals. All data transferred over the bus was parity checked in both directions, and errors were reported through the interrupt system. A watch-dog timer in the I/O channel detected if a nonexistent I/O controller was addressed or if a controller stopped responding during an I/O sequence. The data transfer byte count word in the channel command entry contained four status bits, including a protect bit. When this bit was set on, only output transfers were permitted to this device.

Because I/O controllers were connected between two independent I/O channels, it was very important that word count, buffer address, and direction of transfer be controlled by the processor instead of within the controller. If that information were kept in the controller, a single failure could fail both processors attached to it. Consider what would happen if a byte count register were located in the controller and the count did not decrement on an input transfer. It

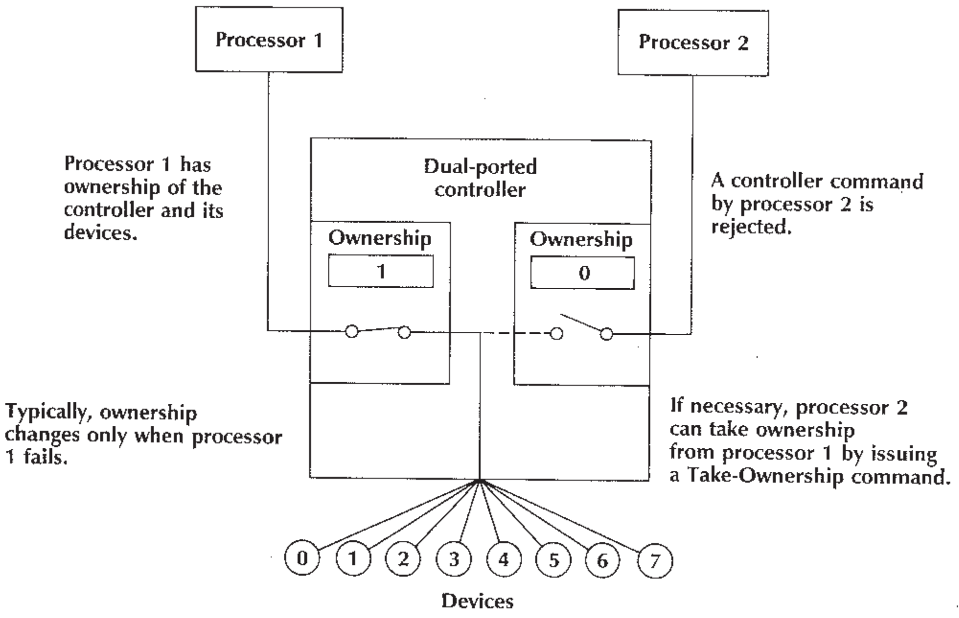

Processor 1
Processor 2
Processor 1 has ownership of the controller and its devices.
Dual-ported controller
A controller command by processor 2 is rejected.
Ownership
1
Ownership
0
Typically, ownership changes only when processor 1 fails.
If necessary, processor 2 can take ownership from processor 1 by issuing a Take-Ownership command.
0
1
2
3
4
5
6
7
Devices

FIGURE 8–46 *Ownership circuitry and logic*

would be possible to overwrite the buffer and render system tables meaningless. The error would propagate to the other processor upon discovery that the first processor was no longer operating.

Other error conditions that the channel checked for were violations of I/O protocol, attempts to transfer to absent pages (it is the operating system's responsibility to lock down the virtual pages used for I/O buffering), uncorrectable memory errors, and map parity errors.

NonStop II

The NonStop II was a compatible extension of the NonStop I. The major changes from the NonStop I processor module were the introduction of a 32-bit addressing scheme and a diagnostic data transceiver processor. The software for the NonStop II system was upward-compatible with the NonStop I system. Thus, application programs written for the NonStop I system could be run on the NonStop II system.

The IPU was implemented using Schottky TTL logic using a microinstruction cycle time of 100 nsec. Instructions were added to support 32-bit extended addressing. An optional floating point instruction set was also added for high-speed scientific calculations; eventually, these instructions became a standard part of the instruction set. The instruction sets were implemented on microcode in a high-speed control store, which had 8K 32-bit words of loadable storage and 1K words of read-only storage. The loadable part of the control store was initialized when the operating system was loaded. Before loading the control store, the system performed a set of diagnostic routines to verify that the processor was operating correctly. The processor's internal data paths and registers were parity-checked to ensure data integrity. The IPU featured a two-stage pipeline that allowed it to fetch the next instruction while executing the current instruction.

Memory boards for the NonStop II system contained 512 KB, 2 MB, or 4 MB of storage. Up to four of these boards, in any combination, could reside in one processor for a maximum of 16 MB. A fully configured 16-processor system allowed up to 64 boards with a total of 256 MB of memory. The memory access time was 400 nsec. Each memory word was 22 bits long. Six bits of the word provided an error-correction code that enabled the system to correct any single-bit error and detect any double-bit error. The error-correction code also checked the address sent from the IPU to ensure that the memory access was valid.

I/O Channel. Each processor module contained a separate processor dedicated to I/O operations. Because the I/O processor operated independently from the IPU, I/O transfers were extremely efficient and required only a minimum of IPU intervention. The channel was a burst multiplexor. Every I/O device controller was buffered, allowing data transfers between main memory and the controller buffer to occur at full memory speed. I/O transfers had a maximum length of 64 KB. The high-speed I/O channels used burst-multiplexed direct memory access to provide transfer rates of up to 5 MB/sec. Thus, the aggregate burst I/O rate of a fully configured 16-processor system was 80 MB/sec.

The I/O processor supported up to 32 device controllers. Depending on the type, device controllers could support up to eight peripheral units. Therefore, as many as 256 devices could be connected to a single processor. Multipoint communication lines were treated as a single device, so each processor could support very large terminal configurations. I/O device controllers were intelligent devices. This intelligence allowed them to relieve the central processing unit of many routine functions such as polling synchronous terminals, checking for data transmission errors, and so forth.

Diagnostic Data Transceiver. The diagnostic data transceiver (DDT) was a separate microprocessor included as part of each processor module. The DDT provided two distinct functions:

1. The DDT allowed communication between a processor module and the operations and service processor (OSP), which supports both operational and maintenance functions, such as running diagnostics. (More about the OSP appears in the section on maintenance facilities and practices later in this case.)
2. The DDT monitored the status of the central processing unit, Dynabus interface, memory, and the I/O processor, and reported any errors to the OSP.

Virtual Memory. The virtual memory addressing scheme, introduced by the NonStop II processor, is used by all subsequent processors. It converted the system from 16-bit addressing to 32-bit addressing. This addressing is supported by the instruction set and is based on segments that contain from 1 to 64 pages each. A page contains 2048 bytes. Each processor can address up to 8192 segments, providing a billion bytes (1 GB) of virtual memory address space (later extended to 2 GB on the Cyclone processor, introduced in 1989).

The instruction set supports standard and extended addressing modes. The standard 16-bit addressing mode provides high-speed access within the environment of an executing program. The extended 32-bit addressing mode allows access to the entire virtual memory space by privileged processes. Programs written in Pascal, C, COBOL85, and the Transaction Application Language (TAL) can use extended addressing for access to large data structures.

The instruction set supports two types of extended addressing: absolute and relocatable. Absolute extended addressing is available only to privileged users such as the operating system itself. Absolute addresses can address any byte within the virtual memory. Relocatable extended addresses are available to all users. This form of addressing can reference any byte of the current process's data space, as well as one or more private relocatable extended data segments. Each extended data segment can contain up to 127.5 MB.

To provide efficient virtual-to-physical address translations, each NonStop II processor included 1024 high-speed map registers. The memory maps contained absent, dirty, and referenced bits to help the software manage virtual memory.

Maintenance. A major feature of the NonStop II system was the OSP located in a console supplied with the system. In addition to serving as an operations interface for communication with the system, the OSP was a powerful diagnostic and maintenance tool. The OSP is described later in this section.

NonStop TXP Processor

While the NonStop II system extended the instruction set of the NonStop I system to handle 32-bit addressing, it did not efficiently support that addressing mode. The existing 5 MB/s I/O channel and 26 MB/s Dynabus interprocessor bus offered more than enough bandwidth to handle a processor with two to three times the performance. The existing packaging had an extra processor card slot for future enhancements, and the existing power supplies could be reconfigured to handle a higher powered processor. The NonStop TXP processor module, introduced in 1983, was designed in this environment.

The main problems concerned designing a new micro-architecture that would efficiently support the 32-bit instructions at much higher speeds, with only 33 percent more printed circuit board area and the existing backplane. This design involved eliminating some features that were not critical to performance and finding creative ways to save area on the PC board, including strategic uses of programmable array logic and an unusual multilevel control-store scheme.

The performance improvements in the NonStop TXP system were attained through a combination of advances in architecture and technology. The NonStop TXP architecture used dual

16-bit data paths, three levels of macro-instruction pipelining, 64-bit parallel access from memory, and a large cache (64 KB/processor). Additional performance gains were obtained by increasing the hardware support for 32-bit memory addressing. The machine's technology includes 25-nsec programmable array logic, 45-nsec 16K static RAM chips, and Fairchild Advanced Schottky Technology (FAST) logic. With these high-speed components and a reduction in the number of logic levels in each path, a 12-MHz (83.3 nsec per microinstruction) clock rate could be used.

The TXP's dual data-path arrangement increased performance through added parallelism, as shown in Figure 8–47. A main ALU operation could be performed in parallel with another operation done by one of several special modules. Among these modules were a second ALU to perform both multiplications and divisions, a barrel shifter, an array of 4096 scratch-pad registers, an interval timer, and an interrupt controller. Other modules provided interfaces among the IPU and the interprocessor bus system, I/O channel, main memory, and a diagnostic processor.

The selection of operands for the main ALU and the special modules was done in two stages. In the first stage, data was accessed from the dual-ported register file or external registers and placed into two of the six registers. During the same cycle, the other four pipeline registers were loaded with cache data, a literal constant, the result of the previous ALU operation, and the result of the previous special-module operation. In the second stage, one of the six pipeline registers was selected for each of the main ALU inputs, and another one of these registers was selected for each special-module operand. Executing the register selection in two stages, so that the register file could be two-ported rather than four-ported, greatly reduced the cost of multiplexers and control storage; the flexibility in choosing the required operands was unimpeded.

The dual 16-bit data paths tended to require fewer cycles than a single 32-bit path when manipulating byte and 16-bit quantities. However, the paths did require slightly more cycles when manipulating 32-bit quantities. A 32-bit add took two cycles rather than one, but the other data

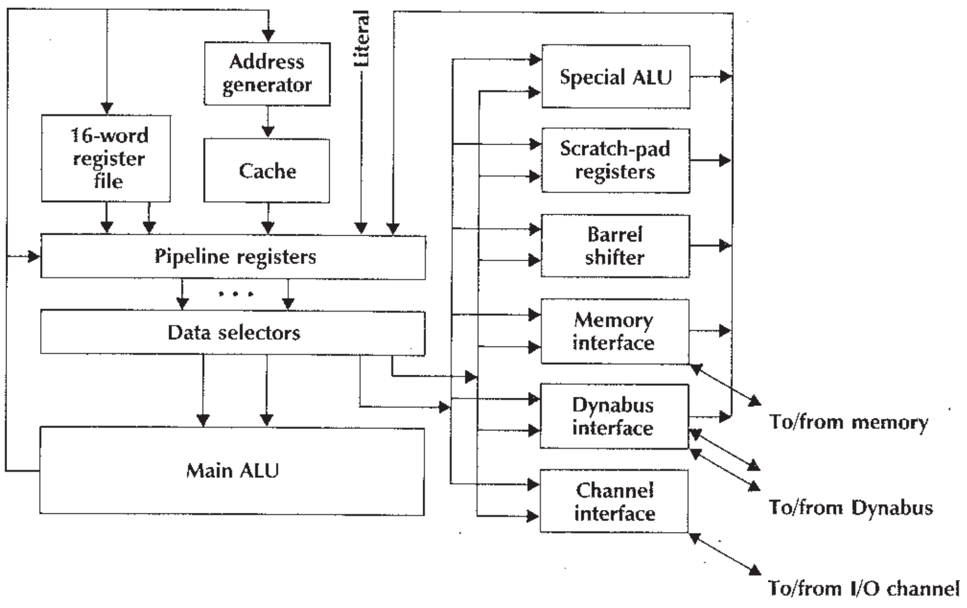


FIGURE 8–47 *Parallel data paths of the TXP processor [Horst and Metz; © 1984 by McGraw-Hill]*

path was free to use the two cycles to perform either another 32-bit operation or two 16-bit operations. Measurements of transaction-processing applications showed that the frequencies of 32-bit arithmetic were insignificant relative to data-movement and byte-manipulation instructions, which were handled more efficiently by the dual data paths than by a single 32-bit data path. Most instructions include enough parallelism to let the microcode make effective use of both data paths.

To control the large amount of parallelism in the NonStop TXP processor, a wide control-store word was required. The effective width of the control store was over 100 bits. To reduce the number of RAMs required, the control store was divided between a vertical control store of 8K 40-bit words and a horizontal control store of 4K 84-bit words. The vertical control store controlled the first stage of the microinstruction pipeline and included a field that addressed the horizontal control store, whose fields controlled the pipeline's second stage. Lines of microcode that required the same or similar horizontal controls could share horizontal control-store entries.

Unlike microprocessor-based systems that have microcode fixed in read-only memory, the NonStop TXP system microcode was implemented in RAM so that it could be changed along with normal software updates and so that new performance-enhancing instructions could be added. Because instructions were pipelined, the TXP processor could execute its fastest instructions in just two clock cycles (167 nsec). The processor could also execute frequently used load and branch instructions in only three clock cycles (250 nsec).

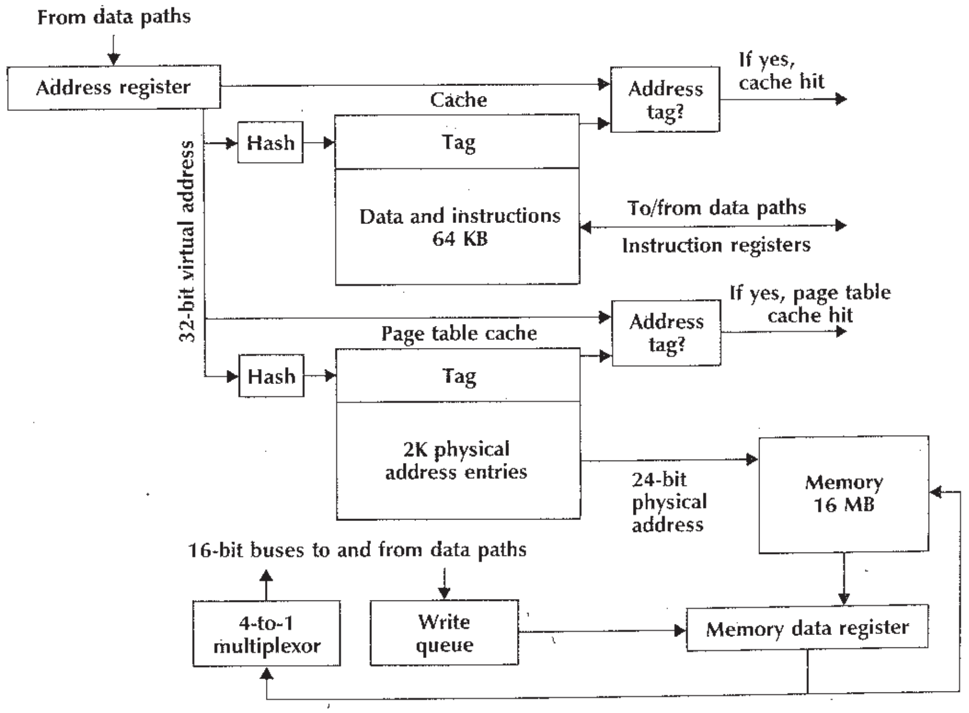
Each NonStop TXP processor had a 64-KB cache holding both data and code. A 16-processor NonStop TXP system had a full megabyte of cache memory. To determine the organization of the cache, a number of measurements were performed on a NonStop II system using a specially designed hardware monitor. The measurements showed that higher cache hit ratios resulted with a large, simple cache (directly mapped) than with a smaller, more complex cache (organized as two-way or four-way associative). Typical hit ratios for transaction processing on the NonStop TXP system fell in the range of 96 percent to 99 percent. Cache misses were handled in a firmware subroutine, rather than by the usual method of adding a special state machine and dedicated data paths for handling a miss. Because of the large savings in the cache hardware, the cache could reside on the same board as the primary data paths. Keeping these functions proximal reduced wiring delays, contributing to the fast 83.3-nsec cycle time.

The cache was addressed by the 32-bit virtual address rather than by the physical address, thus eliminating the extra virtual-to-physical translation step that would otherwise be required for every memory reference. The virtual-to-physical translation, needed to refill the cache on misses and to store through to memory, was handled by a separate page table cache that held mapping information for as many as 2048 pages of 2 KB each (see Figure 8–48).

Manufacturing and Testing. The NonStop TXP processor was implemented on four large PC boards using high-speed FAST logic, PALs, and high-speed static RAMs. Each processor module had from one to four memory boards. Each memory board contained up to 8 MB of error-correcting memory. A 16-processor NonStop TXP system could therefore contain up to 256 MB of physical memory.

The NonStop TXP system was designed to be easy to manufacture and efficient to test. Data and control registers were implemented with shift registers configured into several serial-scan strings. The scan strings were valuable in isolating failures in field-replaceable units. This serial access to registers also made board testing much faster and more efficient because the tester could directly observe and control many control points. A single custom tester was designed for all four IPU boards and for the memory-array board.

FIGURE 8–48
TXP memory access



VLX Processor Module

The VLX processor module combines advanced VLSI technology with the fault-tolerant features of its predecessors. This processor module uses emitter-coupled logic (ECL) gate array technology to implement its dual path structure and other extensions to the TXP system. These features include dual interleaved control store; 83.3-nsec cycle time; 64-KB direct-mapped, store-through cache with 16-byte block size; hardware cache fill; 256-MB physical memory addressing; up to 96 MB of physical memory with 48-MB memory boards; on-line power and temperature monitoring; four-stage instruction pipeline, supporting single clock instruction execution; and dual 20-MB/sec Dynabus interprocessor bus.

One of the VLX processor module's printed circuit boards appears in Figure 8–49. The VLX IPU uses 32-bit native addressing and 64-bit main memory transfers to improve upon the transaction throughput of its predecessors, move large amounts of data, and lower the cost per transaction. Failed component sparing in cache memory allows a single malfunctioning component to be replaced by means of a logical switch to a spare; thus, a single point of failure does not require a service call.

Chips in the VLX processors contain up to 20,000 circuits, producing modules with over three times the density of the TXP processor. This increased density adds functions that enhance error checking and fault correction, as well as performance. By making it possible to reduce the number of components and interconnections, the increased density improves both performance and reliability. VLX processor gate arrays use ECL for enhanced internal performance and TTL for

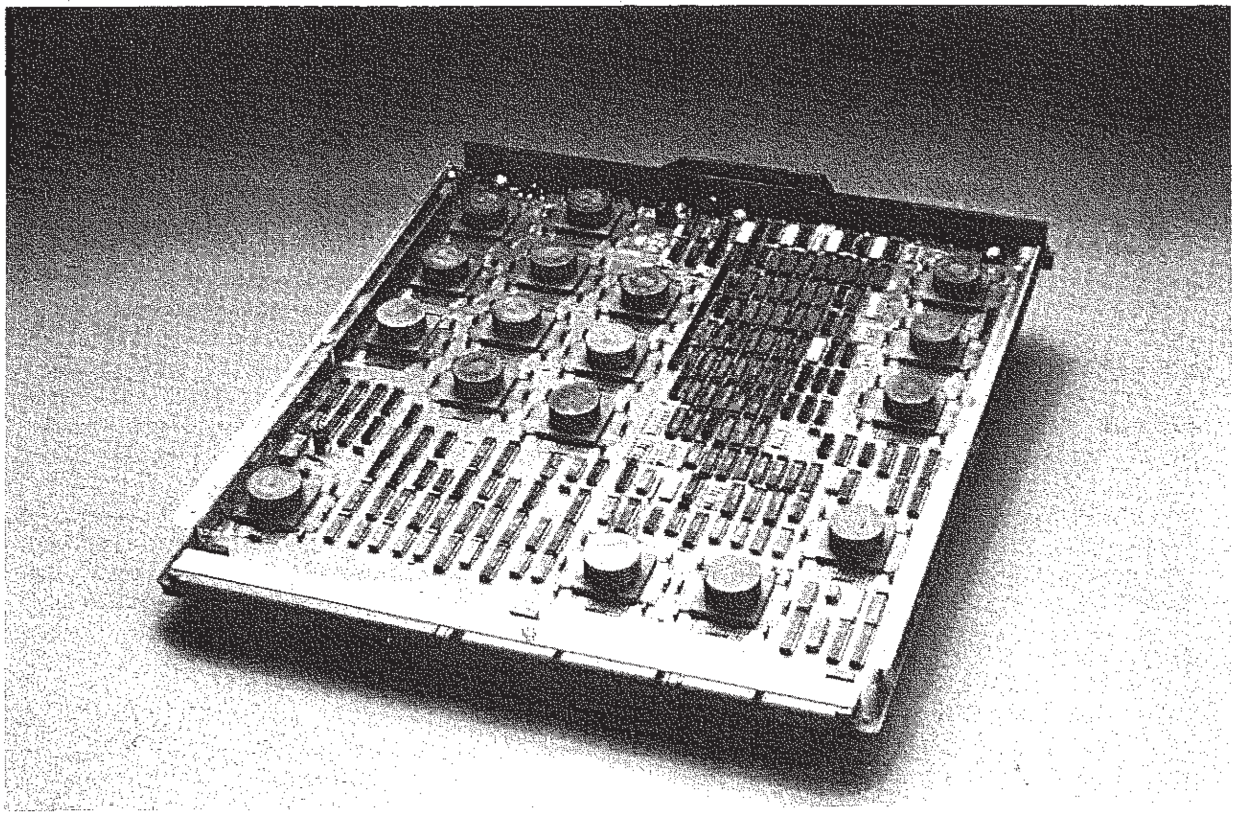


FIGURE 8–49 *Printed circuit board from VLX processor module*

input/output functions. Each VLX processor includes 31 ECL/TTL gate arrays spread over only two modules.

Maintenance. A major goal of the VLX processor was to reduce the cost of servicing the system. This goal was accomplished in several ways.

Traditional mainframe computers have error-detection hardware as well as hardware that allows instructions to be retried after a failure. This hardware is used both to improve availability and to reduce service costs. The Tandem architecture does not require instruction retry for availability; processors can be fail-fast. The VLX processor is the first Tandem processor to incorporate a kind of retry hardware, primarily to reduce service costs.

In the VLX processor, most of the data-path and control circuitry is in high-density gate arrays, which are extremely reliable. This design leaves the high-speed static RAMs in the cache and the control store as the major contributors to processor unreliability. Both the cache and the control store are designed to retry intermittent errors, and both have spare RAMs that can be switched in to continue operating despite a hard RAM failure [Horst, 1989].

The cache provides store-through operation, so there is always a valid copy of cache data in main memory. A cache parity error just forces a cache miss, and the correct data is refetched from memory. The microcode keeps track of the parity error rate; when this rate exceeds a threshold, the microcode switches in the spare RAM. The VLX control store has two identical copies to allow a two-cycle access of each control store starting on alternate cycles. The second copy of control store is also used to retry an access in case of an intermittent failure in the first copy. Again, the microcode switches a spare RAM on line once the error threshold is reached. Traditional instruction retry was not included due to its high cost and complexity relative to the small improvement in system MTBF it would yield.

There is also parity checking on all data paths, single-bit error correction and double-bit error detection on data in memory, as well as single-bit error detection on addresses. Bus control lines are checked for line errors, and hardware consistency checks are used throughout the system.

Each processor contains a microprocessor-based diagnostic interface, which ensures that the processor is functioning properly before the operating system receives control. Pseudo-random scan diagnosis is conducted to provide a high level of coverage and a short execution time. Correct operation of the processor is verified before processing begins.

The VLX system cabinet, shown in Figure 8–50, is divided into four sections: the upper card cage, the lower card cage, the cooling section, and the power supply section. The upper card cage contains up to four processors, each with its own I/O channel and private memory. The lower card cage contains up to 24 I/O controller printed circuit (PC) cards, where each controller consists of one to three PC cards. The cooling section consists of four fans and a plenum chamber

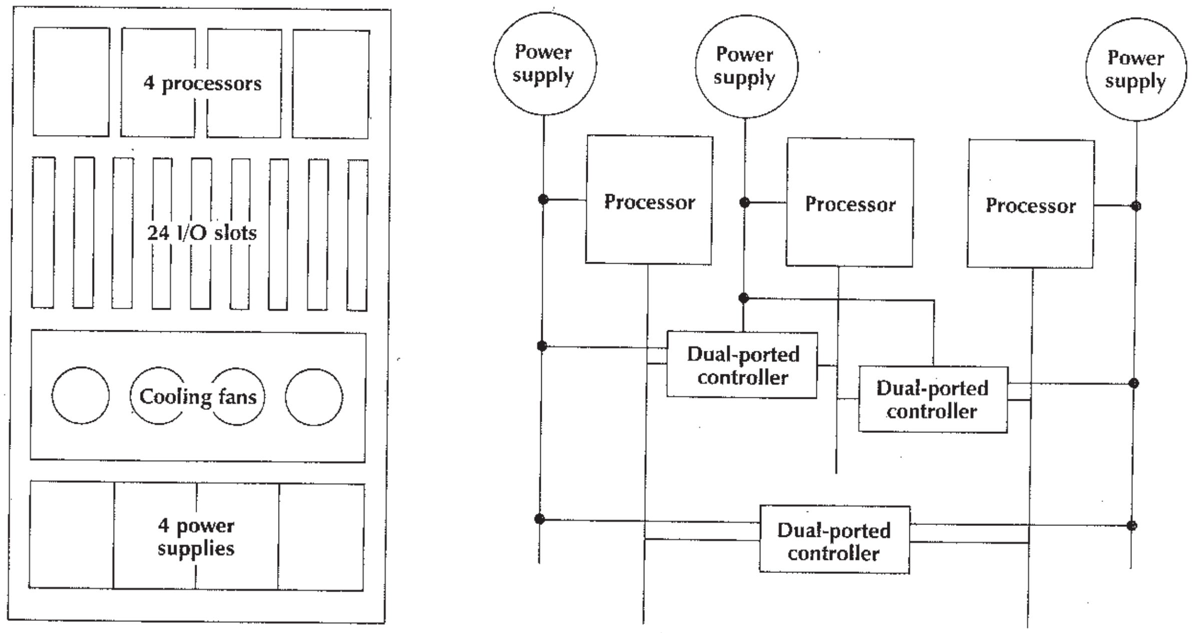


FIGURE 8–50 *NonStop VLX system cabinet (left) and power distribution (right)*

that forces laminar air flow through the card cages. The power supply section contains up to four power supply modules. Multiple cabinets can be bolted together.

For the VLX system, the Tandem Maintenance and Diagnostic System (TMDS) replaced the operations and service processor used on the NonStop II and TXP processors. Information about TMDS appears later in this case, in the section on maintenance facilities and practices.

CLX Processor Module

The CLX system was designed to fill the need for a low-cost distributed system. The design goal was to provide user serviceability, modular design, and fault tolerance with lower service and maintenance costs. The CLX is based on a custom CMOS chip set developed using silicon compilation techniques [Lenoski, 1988]. The original CLX-600 processor was introduced in 1987 and is based on 2.0-μ CMOS. The silicon compiler allowed the processor chip to be retargeted into 1.5-μ CMOS for the CLX-700 and into 1.0-μ CMOS for the CLX-800. The CLX-700, introduced less than 18 months after the 600, raised performance by 50 percent, and the CLX-800 raised performance again by nearly 50 percent. The description in this section is based on the latest CLX-800.

All CLX processors have a similar micro-architecture that integrates the features of both traditional board-level minicomputers and high-performance VLSI microprocessors. This hybrid design incorporates several novel structures, including a single static RAM array that serves three functions: writable control store, data cache, and page table cache. The processor also features intensive fault checking to promote data integrity and fault-tolerant operation.

A fully-equipped, single-cabinet CLX system contains two processor boards with optional expansion memory, six I/O controllers, five 145-MB to 1-GB disk drives, and one cartridge tape drive. Dual power supplies and cooling fans are also included in the cabinet. The entire system operates within the power, noise, and size requirements of a typical office environment. To expand the system, the customer simply adds more I/O or processor cabinets to the basic configuration. The CLX system architecture appears in Figure 8–51. A view of the actual cabinet appears in Figure 8–52.

As with the other Tandem processors, each CLX processor communicates with other processors over two interprocessor buses (IPBs). Each bus operates synchronously on 16-bit wide data, and each provides a peak bandwidth of 20 MB/sec. The two buses transfer data independently of one another, providing a total bandwidth of 40 MB/sec for a maximum of eight processors. Processors communicate with I/O devices either through a local I/O bus or through the interprocessor bus (IPB) to another processor and its I/O bus. Each processor contains a single asynchronous, burst-multiplexed I/O bus that transfers data at a maximum rate of 4.4 MB/sec to a maximum of 16 controllers. As with the other processors, these controllers are dual-ported and can be driven by either of the processors to which they are attached.

The CLX uses a multifunction controller (MFC) based on a Motorola 68010 microprocessor to control dual small computer system interfaces (SCSI) that support up to five disk drives and one tape drive. The MFC runs its own real-time operating system kernel that coordinates independent disk control, tape control, synchronous and asynchronous communication, and remote maintenance tasks. The CLX processor module's printed circuit board appears in Figure 8–53. Through the maintenance buses, maintenance and diagnostic information can flow among the system control panel, processors, multifunction controllers, and environmental monitors. Enhanced diagnostic software and careful design of all replaceable units allows customers to service 98 percent of all component failures.

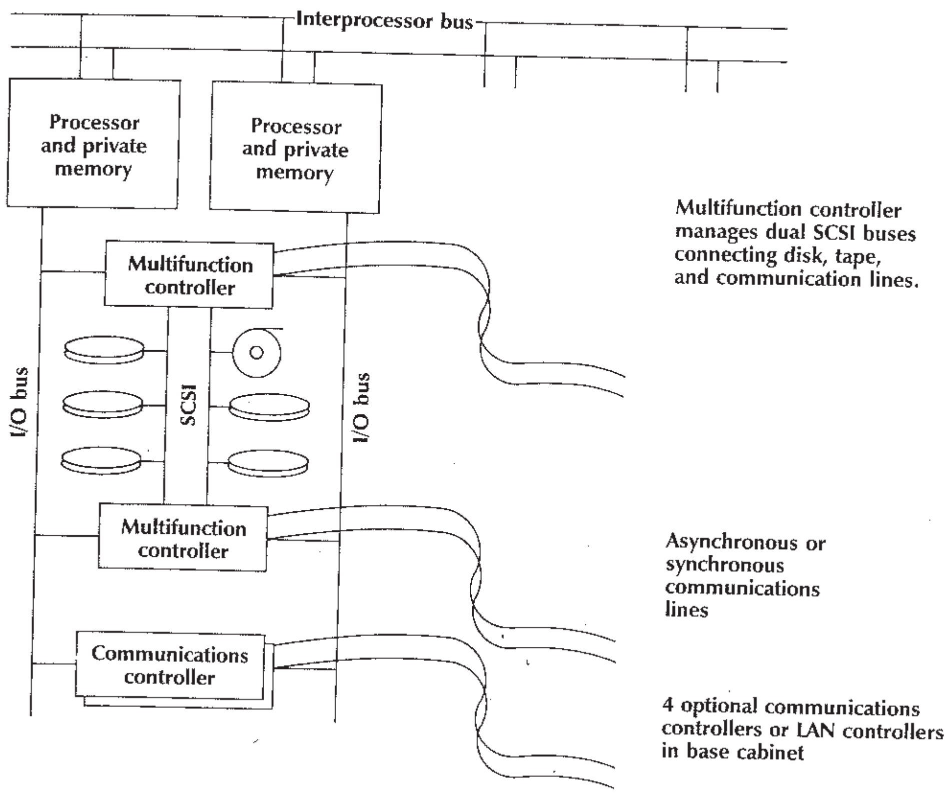

Interprocessor bus
Processor and private memory
Processor and private memory
Multifunction controller
I/O bus
SCSI
I/O bus
Multifunction controller
Communications controller
Multifunction controller manages dual SCSI buses connecting disk, tape, and communication lines.
Asynchronous or synchronous communications lines
4 optional communications controllers or LAN controllers in base cabinet

FIGURE 8–51 *CLX system architecture*

The processor logic resides within six custom CMOS chips, allowing the processor and main memory to be implemented in a single board. A block diagram of the processor appears in Figure 8–54. The chip set was designed using a silicon compiler supplied by Silicon Compiler Systems Corporation. The two IPU chips are identical, running in lock-step to form a fully self-checking module. These chips provide the complete IPU function. They work together with a single bank of static RAM that serves as the microcode control store, page table cache, and data/instruction cache. The RAM provides for 14K × 7B of microcode and scratch pad memory, 4K entries of page table cache, and 192 KB of instruction/data cache.

The MC chip includes the control and ECC logic (SEC/DED) to interface with the up to 32 MB of on-board dynamic memory. This chip contains FIFO buffers to hold data in transit to and from the main-memory dynamic RAMs, using nibble mode access. The chip also features a wraparound mode to support high-speed memory-to-memory block transfers.

Each processor has one IPB chip per interprocessor bus. Each chip contains a 16-word in-queue and a 16-word out-queue. These queues work with on-chip state machines for sending and receiving interprocessor message packets asynchronous to processor execution.

The IOC chip contains the data latches and logic to control a burst-multiplexed, asynchronous I/O bus. The I/O bus is primarily controlled by the IPU, but it can also handle DMA transfer polling and selection without microcode intervention. The bus also includes priority-encoding logic to support the fair servicing of I/O interrupts.

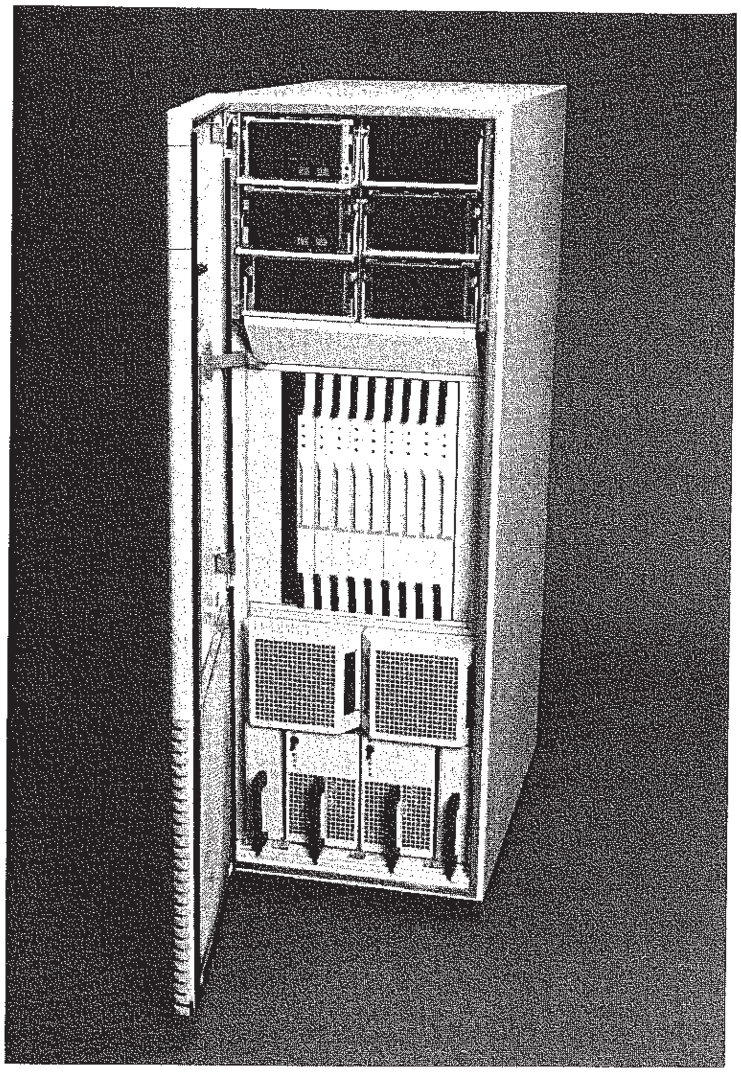


FIGURE 8–52
CLX system cabinet (front view showing from top to bottom: cartridge tape drive and SCSI disk drives, two processor modules with memory and I/O boards, dual fans, power supplies, and batteries for memory power)

The final component of the processor is a Motorola 6803-based maintenance and diagnostic processor. This processor furnishes overall control of the main processor, as well as a diagnostic and error-reporting path for the main processor through the maintenance buses.

The IPU architecture for the CLX, as mentioned earlier, is a blend of features found in both minicomputer and microcomputer architectures. The IPU chip's external interface is similar to that of a VLSI microprocessor. For example, the interface features one address bus, one data bus, and one status bus, along with miscellaneous signals, such as an interrupt request, memory wait

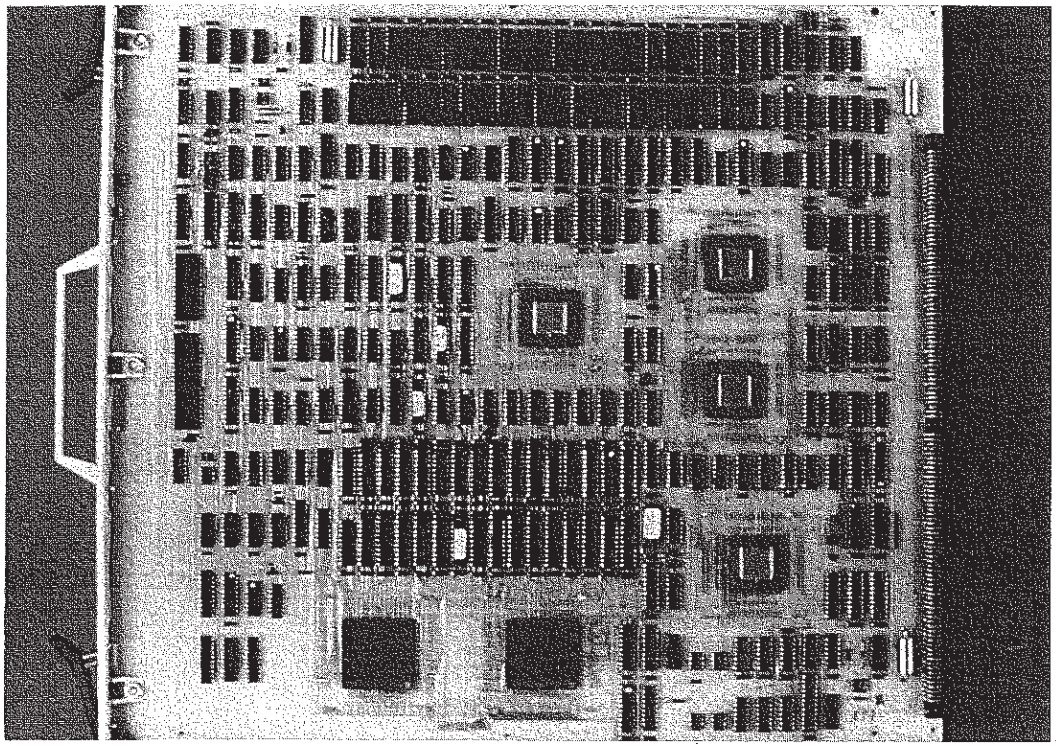


FIGURE 8–53 *Printed circuit board for CLX-600 processor module, showing dual, lock-stepped IPU chips (bottom center)*

controls, and three-state bus controls. Minicomputer features, however, appear in the size of the address bus, which is 18 bits wide, and the data bus, which is 60 bits wide.

The IPU chip interface merges many buses that would normally be separate in a minicomputer architecture. In particular, a bus cycle on the CLX can execute any of the following functions: microcode control store access, instruction or data cache access, page table cache access, main memory access, microcode scratchpad memory access, and special module (IPB, IOC, MDP) access. Merging these buses reduced the cost of the processor by decreasing the number of static RAM parts and their associated support logic and by reducing the number of pins needed on the IPU chips. If this merging were not implemented carefully, however, performance would have degraded significantly. To reduce the bandwidth required on the buses and to minimize the impact on performance, the designers employed a variety of techniques, including the use of a small on-chip microcode ROM, a virtually-addressed cache, nibble-mode DRAM with block operations to the main memory controller, and high-level control operations for special modules.

The on-chip micro-ROM is most important in reducing the impact from the merged bus structure. The micro-ROM contains 160 words of microcode, with an identical format to the off-chip microcode. This ROM is addressed either by the microcode PC or through an explicit index

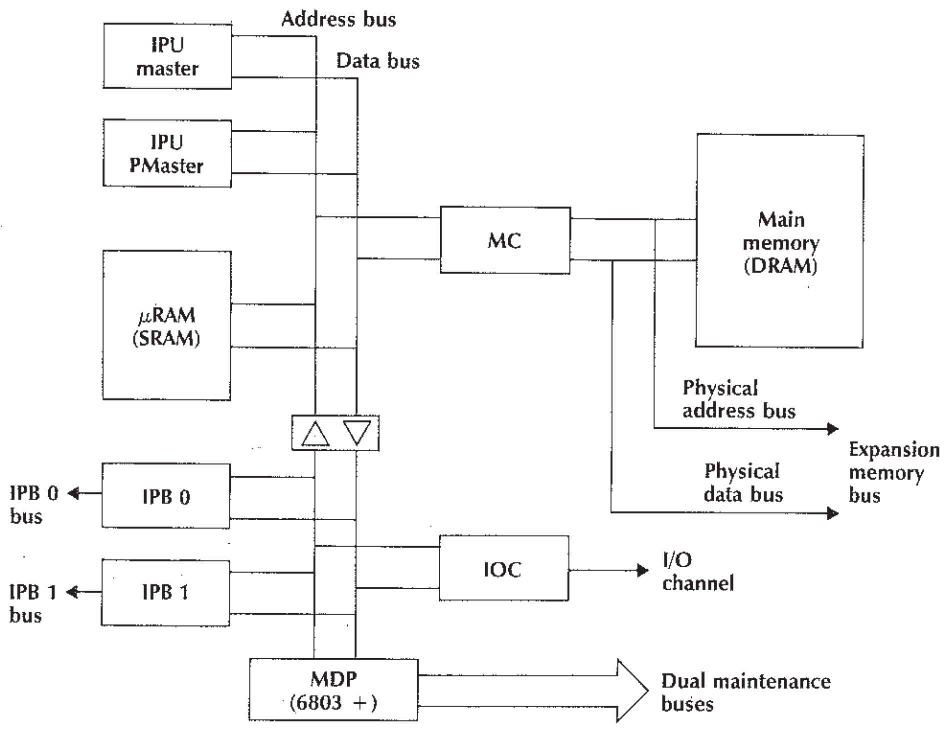


FIGURE 8–54
CLX processor block diagram

specified in the previous line of microcode. The microcode PC addressing is used to implement the inner loops of IPB and IOC transfers, cache filling routines, and block memory moves. The explicit index is used for short sequences of common microcode. These lines overlay otherwise sequential lines of external microcode. Use of these ROM lines does not conflict with other micro-operations.

The virtually-addressed cache reduces the number of page-table accesses; thus, it decreases the required bandwidth to the shared micro-RAM. Likewise, the use of block-mode commands to the memory controller reduces the number of memory commands needed during cache filling and block moves. Finally, the use of higher-level commands to the IPB and IOC reduces the control transfers needed to receive and transmit data to these devices. The on-chip micro-ROM, together with these other features, reduces the penalty of using a single bus approach from over 50 percent to less than 12 percent.

The main alternative to the micro-ROM used on the CLX would be an emulation scheme in which a subset of instructions is implemented entirely by internal ROM, and the remaining instructions are emulated by a sequence of the simpler instructions. The micro-ROM scheme has two chief benefits when compared with emulation techniques. First, it provides much higher performance when the amount of ROM space is limited relative to the number of instructions that must be implemented. Second, the dispatch of each instruction is to external writable control store, enabling any ROM microcode errors to be corrected externally (although with some performance penalty).

Data Integrity Through Fail-Fast Operation. In a NonStop system, fail-fast hardware operation is essential to providing fault tolerance at the system level. Fail-fast operation requires that faults do not escape detection and that the processor is halted before a fault is propagated. The CLX's processor module uses a variety of error-checking strategies to provide extensive fault coverage.

The IPU chip itself is covered by a duplicate-and-compare scheme. This scheme minimizes the amount of internal logic required for a high degree of coverage, and it maximizes the utilization of existing library elements in the silicon compiler CAD system. The implementation of the IPU's duplicate-and-compare logic appears in Figure 8–55. The CLX's scheme improves the fault coverage of other duplicate-and-compare schemes by providing for a cross-coupling of data and parity outputs. One chip, designated the data master, drives all data outputs, while the other chip, designated the parity master, drives all parity outputs. This action ensures that both chips' outputs and checking logic are active and that latent errors in the checking logic cannot lead to an undetected double failure. The parity outputs of the IPU also cover the address and data lines connecting the IPU to other parts of the processor and the micro-RAM.

Within the memory system, ECC with encoded address parity provides checking of all memory system data paths. In addition, redundant state machines are contained in the MC chip

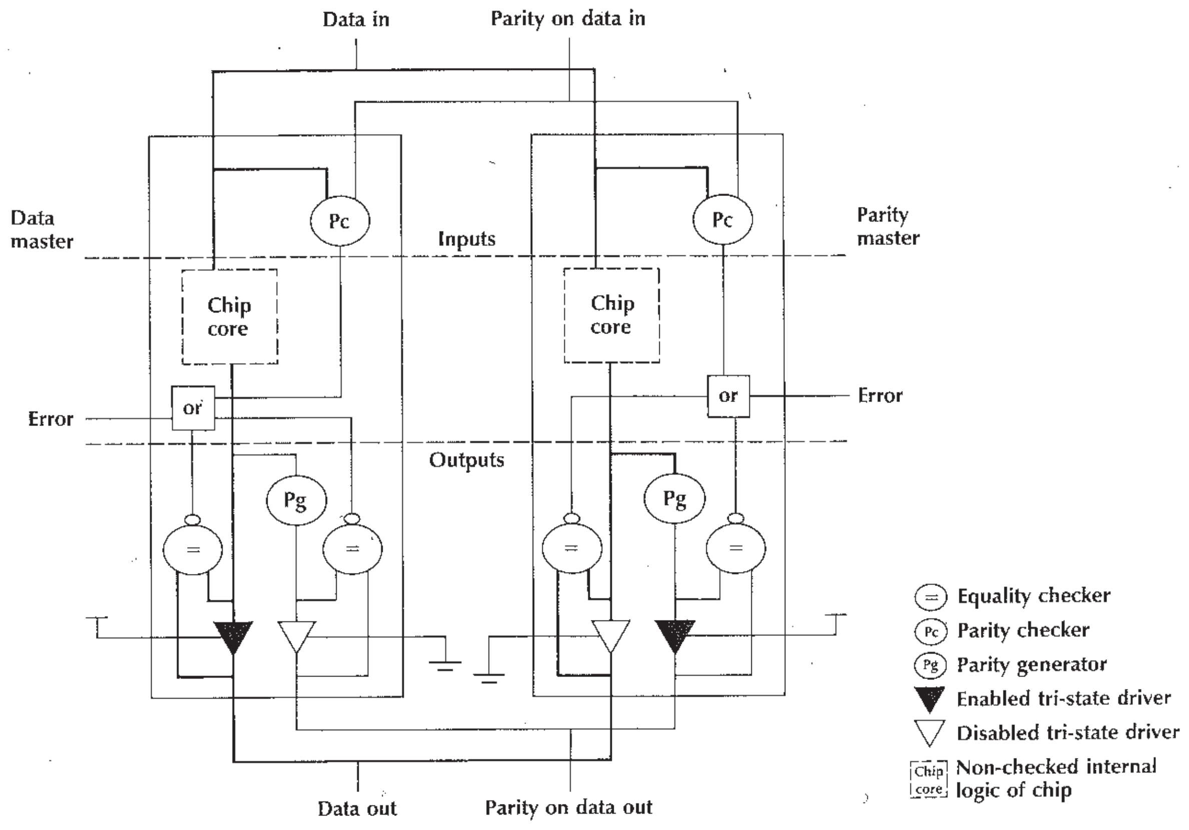


FIGURE 8–55 *CLX processor's cross-coupled checking*

and in the external RAS/CAS generation logic. The state transitions of these machines are encoded into CRC registers whose outputs are compared. The resulting structure produces a high fault coverage for both the data and control sections of main memory.

The IOC and IPB provide for parity protection of the data and control lines to which they are interfaced. In addition, they are protected by end-to-end checksums supported in software; these checksums guarantee the integrity of their respective buses.

Cyclone Processor Module

The design goals of the Cyclone system were to significantly increase performance, while providing improvements in serviceability, manufacturability, installability, and overall cost of ownership. The Cyclone processor, introduced in 1989 [Horst, Harris, and Jardine, 1990], provides more than three times the performance of the VLX, yet it retains full object-code compatibility. About half of the performance improvement is due to higher clock rates, and the other half is due to the new micro-architecture. Much of the architectural improvement is due to the ability to execute up to two instructions per clock cycle, a technique that has been called *superscalar* [Jouppi and Wall, 1989]. Other improvements are due to parallel data paths and new designs for the caches and main memory.

Cyclone Technology. The technology for Cyclone is a combination of ECL for speed, CMOS for high density, and TTL for standard interfaces and buses. The ECL gate arrays, jointly developed by Tandem and Advanced Micro Devices, contain approximately 5000 gate equivalents. These gate arrays are implemented in 155-pin, grid-array packages. The pins can be individually programmed for TTL or ECL interfaces.

The processor is implemented on three 18" × 18" circuit boards, with a fourth board holding either 32 MB or 64 MB of main memory. A second, optional, memory board allows expansion up to a total of 128 MB of main memory per processor (with 1 Mb DRAMs). The circuit boards have eight signal layers, four of which have controlled impedance for routing ECL signals.

Like the VLX, Cyclone uses an interleaved control store, allowing two clock cycles for access. The control store is implemented in 16K × 4 CMOS SRAMs, surface-mounted on a double-sided ceramic substrate, which is then vertically mounted on the main boards.

Cyclone Processor Architecture. The superscalar design of the Cyclone processor was necessitated by the fact that the VLX processor executes many frequent instructions in a single clock cycle, and the goal of three times VLX performance could not realistically be achieved based on cycle time only. Such a fast cycle time would involve higher risk, lower reliability, and higher product cost. Thus, Cyclone needed to break the one-cycle-per-instruction barrier. At peak rates, the Cyclone processor executes two instructions per clock cycle. To do this, it incorporates an independent, asynchronous instruction fetch unit (IFU), separate large caches for instructions and data, a deep pipeline, and a dynamic branch prediction mechanism. A block diagram of the Cyclone processor is shown in Figure 8–56.

The IFU operates independently of the rest of the processor. It fetches up to two instructions per clock cycle from the instruction cache, decodes the instructions to determine whether they are candidates for paired execution, and presents a microcode entry address for either a single instruction or a pair of instructions to the control unit and data unit for execution. It also assists in the execution of branching instructions and of exception handling. Up to 16 different instructions can be in some stage of execution at any point in time. The IFU is shown in more detail in Figure 8–57.

The Cyclone processor uses a dynamic branch prediction mechanism for conditional

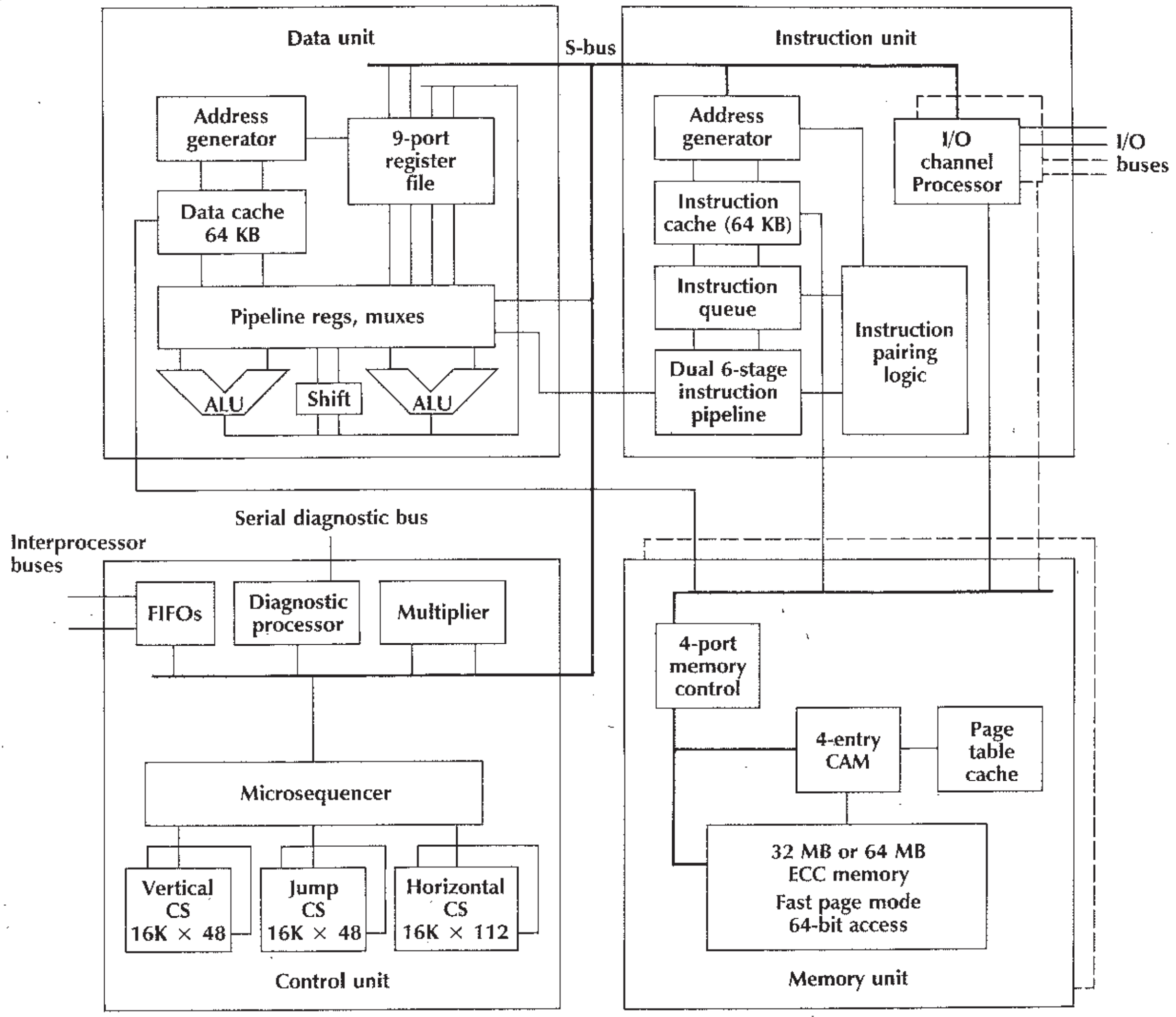
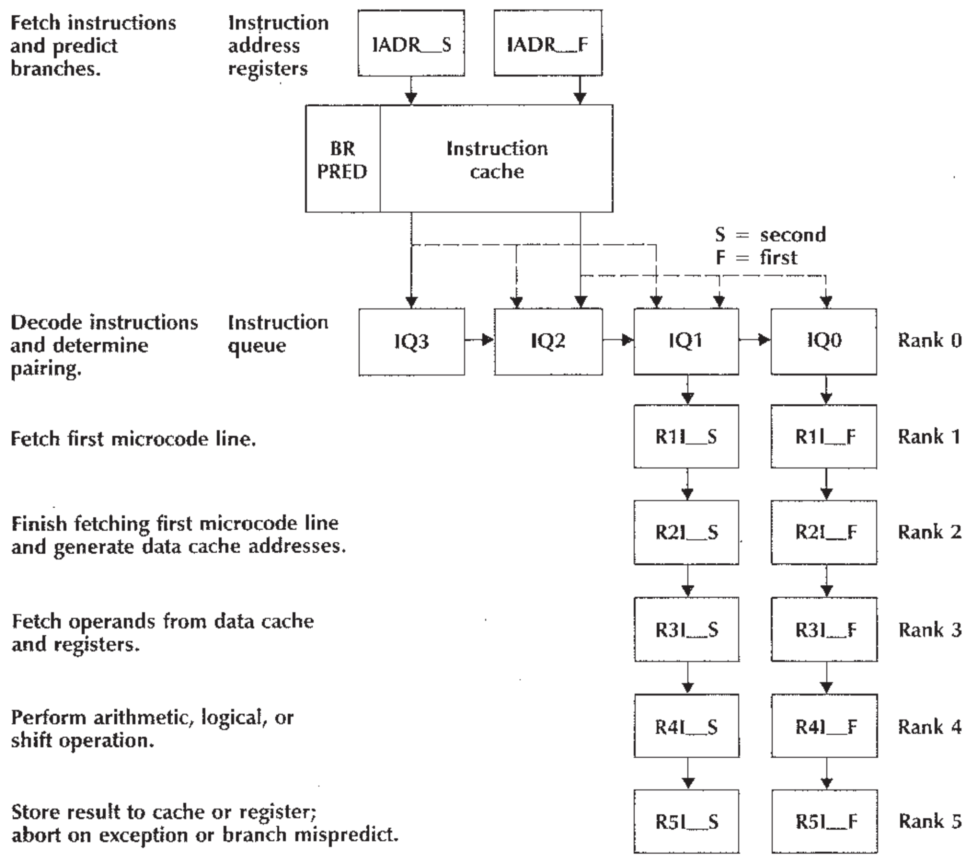


FIGURE 8–56 *Cyclone processor block diagram [Chan and Horst, 1989; reprinted by permission from CMP Publications]*

branches. This mechanism relies on the premise that when a particular branch instruction is repeatedly encountered, it will tend to be taken (condition met) or not taken (condition not met) the same direction each time. An extra bit for each instruction is included in the instruction cache. This bit records the branch direction actually taken by the last execution of each branch instruction in the cache. When a branch is fetched from the instruction cache, the IFU predicts that the branch will choose the same path as the previous time, so it continues prefetching along the predicted path. When the branch instruction later enters the execution pipeline, the microcode determines whether the prediction was correct. If so, it does nothing. If not, the microcode directs the IFU to back up and resume instruction fetching along the other path. Modeling has

FIGURE 8–57
Cyclone instruction fetch unit


Fetch instructions and predict branches.
Instruction address registers
IADR_S
IADR_F
BR PRED
Instruction cache
S = second
F = first
Decode instructions and determine pairing.
Instruction queue
IQ3
IQ2
IQ1
IQ0
Rank 0
Fetch first microcode line.
R1I_S
R1I_F
Rank 1
Finish fetching first microcode line and generate data cache addresses.
R2I_S
R2I_F
Rank 2
Fetch operands from data cache and registers.
R3I_S
R3I_F
Rank 3
Perform arithmetic, logical, or shift operation.
R4I_S
R4I_F
Rank 4
Store result to cache or register; abort on exception or branch mispredict.
R5I_S
R5I_F
Rank 5

shown that this mechanism would be correct between 85 percent and 95 percent of the time. The result is an average cost of 1.3 to 1.9 cycles per branch instruction. In addition, because the branch prediction occurs early in the prefetch queue, branches may be executed in a pair with the previous instruction, the sequentially following instruction, or the target instruction.

The Cyclone data path uses two 16-bit ALUs, similar to the TXP and VLX, but with two major differences. First, the two ALUs are connected with the register file in a very general way. This interconnection is necessary for the execution of many of the instruction pairs, but it is also quite useful in improving the performance of many complex, multicycle instructions as well. In addition, the two ALUs can be linked together so that 32-bit arithmetic can be accomplished in a single clock cycle.

Both the instruction cache and the data cache are capable of fetching two adjacent 16-bit words in a single cycle, regardless of alignment. This feature, along with the instruction pairing, the nine-port register file, the 32-bit ALU capability, and the deep pipeline, allows the execution of a double (32-bit) load instruction and a 32-bit arithmetic instruction, as a pair, in a single clock cycle.

The Cyclone sequencer is similar to the VLX sequencer in that two copies of the control store are used to allow two-cycle access time. In addition to allowing the use of slower, denser CMOS RAM parts, the two copies provide backup for each other. In the event of an error in fetching a word from control store, the alternate bank is automatically accessed. If the error is a soft error, one of the banks can be refreshed from the other bank. In the event of a hard failure, a spare RAM can be switched in. Part of the control store is duplicated yet again (four total copies). This duplication allows both potential paths of a microcode branch to be fetched simultaneously, thus minimizing the penalty for microcode choices.

In the Cyclone processor, virtual addresses are sent directly to the main memory. A four-entry, content-addressable memory (CAM) compares each access to the row address that previously addressed a bank of DRAM. When the addresses match, the DRAM column address is generated by a few bits from the CAM plus the address offset. Translation from virtual to physical address is performed only on a CAM miss. Since the translation is performed infrequently, it was possible to implement the page table cache (translation look-aside buffer) in relatively slow, but dense, CMOS static RAMS.

For both increased bandwidth and increased connectivity, Cyclone allows the connection of up to four I/O channels per processor, whereas previous Tandem processors allowed only one channel. Two channels are supplied on the instruction unit board, while an additional two channels are available on an optional board. The maximum Cyclone processor thus contains six boards (three processor, two memory, one optional I/O).

Cyclone Fault Tolerance, Data Integrity, and Reliability Features. Parity checking is used extensively to detect single-bit errors. Parity is propagated through devices that do not alter data, such as memories, control signals, buses, and registers. Parity prediction is used on devices that alter data, such as arithmetic units and counters. Predicted parity is based strictly on a device's data and parity inputs; it does not rely on the device's outputs, which may be faulty. Thus, an adder might generate an erroneous sum, but the parity that accompanies the sum will correspond to the correct result. Parity checkers downstream will then detect the error. Invalid-state checking or duplication-and-comparison are used in sequential state machines.

The hardware multiplier is protected by a novel technique similar to recomputation with shifted operands (RESO) [Sohi, Franklin, and Saluja, 1989]. After each multiplication, a second multiplication is initiated with the operands exchanged and one operand shifted. Microcode compares the two results whenever the multiplier is needed again or before any data leaves the processor. Unlike other implementations of RESO, these checking cycles incur almost no performance penalty because they occur concurrently with unrelated execution steps.

If the processor hardware detects a fault from which it cannot recover, it shuts itself down within two clock cycles, before it can transmit any corrupt data along the interprocessor bus or I/O channel. The error is flagged in one or more of the approximately 300 error-identification registers, allowing quick fault isolation to any of the 500 hardware error detectors in each processor. Like the VLX, Cyclone processors include spare RAM devices in all of the large RAM arrays, such as caches and control stores. These devices are automatically switched in to replace hard-failed RAMs.

Cyclone systems make extensive use of fiber-optic interconnections, which, among other advantages, increase reliability. The Dynabus+ fiber links between sections were described earlier in this case. In addition, Cyclone systems use fiber optic links between the disk controllers and the disk units themselves and between the communications controllers and outboard communications concentrators.

The Cyclone approach to diagnostics is similar to the approach taken on VLX, but it goes beyond in many respects. Test coverage of microprogrammed diagnostic routines has been dramatically increased, and more support has been added for pseudo-random scan test. Together, these changes improve the ability to automatically diagnose faults on line and quickly pinpoint the field-replaceable unit responsible for the fault. In addition, a guided-probe facility, which leads factory personnel through the diagnostic process, enhances the product's manufacturability.

Like the VLX processor, the Cyclone processor is implemented primarily in ECL gate arrays, although Cyclone's arrays are considerably more dense. Because of this added density and the increased clock speed, Cyclone's gate arrays dissipate up to 11 watts. In order to cool these devices without resorting to liquid cooling, Cyclone uses an *impingement* air-cooling technique. Instead of blowing chilled air across the circuit board, Cyclone's boards include an orifice plate, which serves to focus the incoming air onto the hottest components. This design is shown in Figure 8–58. The result is that Cyclone's devices, in spite of dissipating much more power, operate at a junction temperature 10°C cooler than those in the VLX, significantly increasing reliability.

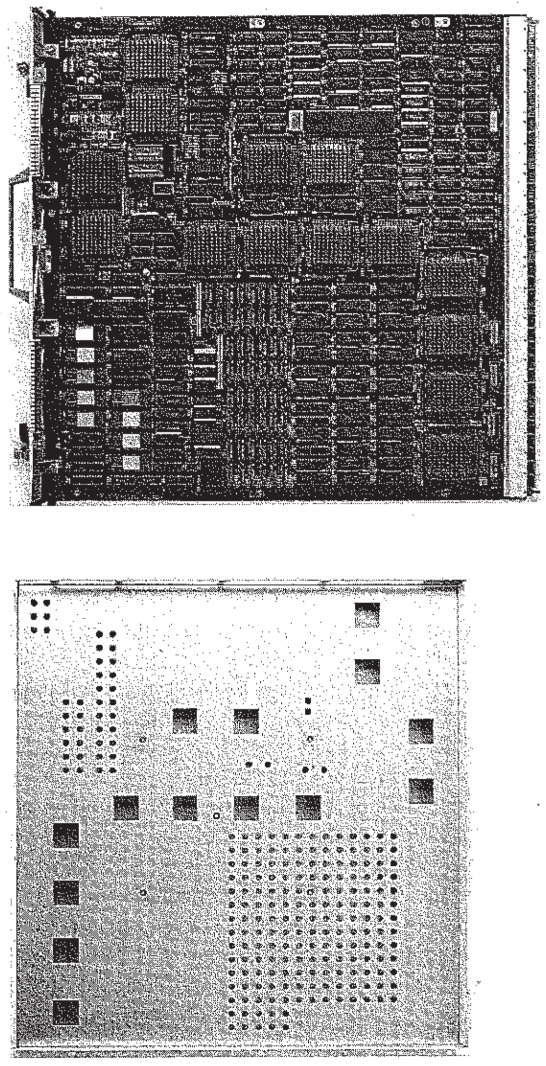


FIGURE 8–58 *Cyclone printed circuit board showing impingement cooling*

INTEGRITY S2

While Tandem's traditional NonStop architecture provides efficient fault tolerance through its fail-fast hardware and proprietary Guardian operating system, some computing environments require an open standards-based operating system and fault tolerance based strictly in hardware (e.g., the telecom industry). Tandem's Integrity S2 was designed to meet the needs of these markets.

The primary design objective for Integrity S2 was to provide a fault-tolerant on-line user-serviceable UNIX-based system [Jewett, 1991]. Application portability at the source level was a requirement as well as support for an industry standard peripheral bus. Furthermore, no single hardware failure should corrupt the data stored or manipulated by the system. Last, but certainly not least, among the major design objectives was the recognition that the operating system would represent a single point of failure.

System Architecture

A depiction of the machine architecture is provided in Figure 8–59. The system is divided into a number of customer-replaceable units (CRUs). Every CRU in the system is designed to be hot-pluggable. This permits on-line removal of fault CRUs and on-line insertion of replacement CRUs.

The system consists of a triplicated processor-local memory system contained on three central processor CRUs (CPCs). Duplexed triple modular redundant controllers (TMRCs) provide a large secondary main memory (global memory) and serve as the nexus for the I/O operations of the machine. The CPCs connect to the TMRCs over the reliable system bus (RSB). Duplexed input/output packetizers (IOPs) provide the interface for a superset of an industry standard I/O bus (VME) on one side and an interface to the TMRC on the other. The interconnection between the IOPs and the TMRCs is called the reliable I/O bus (RIOB). The IOPs are the conduit through which all I/O in the machine flows. Each IOP produces a bus that is called the NonStop-V+, which is a high data integrity variant of the popular VME bus. Ordinary VME controllers are connected to the system via a bus interface module (BIM). The BIM provides a dual-ported path from a peripheral controller to each IOP.

Each CPC consists of a 33.33-MHz oscillator driving an R2000 processor, R2010 floating point coprocessor, 128 KB of split instruction and data caches, local memory and RSB interface. In addition, the CPC contains a DMA machine used to transfer blocks of data between local memory and the secondary memory store, a minimum of 8 MB of on-board DRAM, augmented with hardware write-protection logic and an RSB interface. The DMA engine transfers packets of memory between local and global memory and accumulates a checksum of the data during the transfer. Upper layers of the system software use this checksum to provide end-to-end checking for disk transfers.

The clocks of the three processing modules have no fixed-phase relationship that is maintained by the system. The processors operate independently, but are kept in logical phase via proprietary synchronization logic. Two different time domains are relevant to synchronization: virtual and physical. Virtual time is measured by the passage of instructions on a given CPC. The independent processing modules are designed to execute the same instruction stream in virtual time. These instruction streams proceed until such time as the processing complex needs to access a resource beyond the CPC boundary. All such non-CPC resource requests generate RSB transactions that are voted at the TMRC. Voted read operations inherently bring the processors into closer alignment in physical time because there is a single logical copy of the data.

Since the machine can operate within the bounds of the cache and local memory subsystem for long periods of time, another synchronization mechanism is required. The progress of the processors is monitored on each CPC by a set of counters which are incremented as the machine

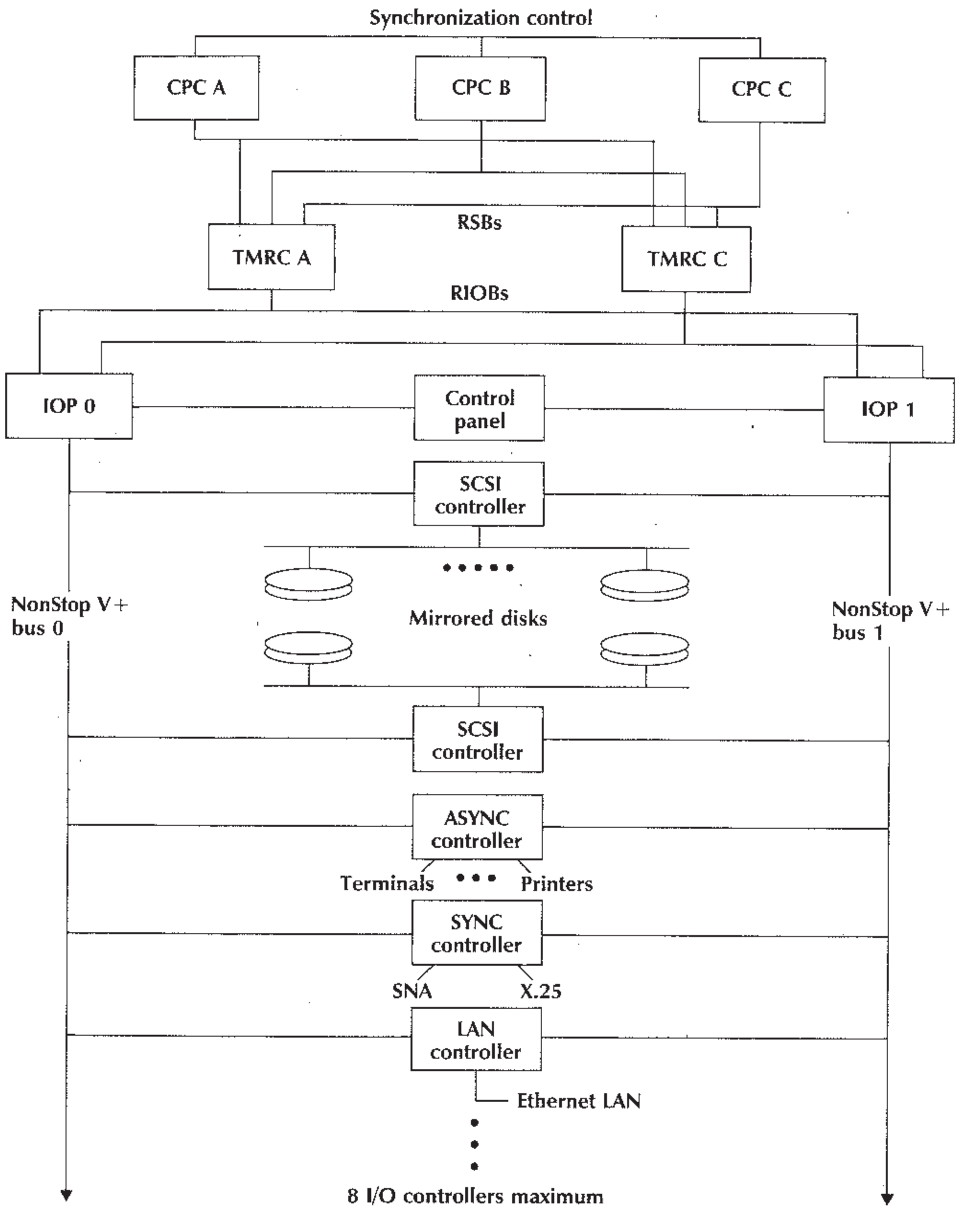


FIGURE 8–59 *Integrity S2 architecture*

pipeline advances. Periodically (every 2047 instructions), each of the processors is stalled until all of the processors arrive at the synchronization point or a timeout expires. In addition, the arbiter for the local bus on the CPC ensures that all machines execute reads and writes in the same temporal order.

The technique devised to provide precise presentation of exceptions to each of the processors involves instruction counting. As the pipeline of the processor advances, a number of counters are incremented. In the current system, interrupts can be presented every 64 instructions. The process of collecting, distributing, voting and presenting exceptions on specific modulos of an instruction counter guarantees that all processors will field the exception at the same virtual time.

The TMRC contains up to 128 MB of global memory and interfaces to the CPC via the RSB and to the IOPs via the RIOB. The TMRC also contains the cause, mask, and clear registers for the interrupt mechanism. Having the TMRC serve as the repository for causes of exceptions presents a uniform view of interrupts to the processors.

A central role that the TMRC has in system operations is voting the RSB transactions. All processor transactions that are external to the CPC are voted on a bit-by-bit basis and the vertical OR of these results implicate the errant CPU module. The voting circuit is duplicated and compared and any self-check error halts the board. During system operation one of the TMRCs is designated as primary and the other as backup. The primary TMRC provides the data in the case of a read operation and both TMRCs perform all write operations. All TMRC registers and static RAMs are protected by even byte parity. The memory is word organized, and the even parity of the word address of the datum is hashed into every byte of data parity in order to detect addressing failures in the memory controller.

The nonvolatile memory (EPROM) on the TMRC is not parity protected, but is checksummed by the software. All of the data paths on each TMRC are protected by even parity hashed among the four data bytes. State machines are protected using either parity or duplication. Scrubbers, implemented in the operating system, are used to detect, and correct if possible, latent errors in both local and global memory.

Like the TMRC, the IOP is designed to be a self-checking fail-fast CRU. The system has two IOPs, each of which can support the full complement of peripheral controllers. If an IOP fails, software arranges for the peripheral controllers to become bound to the other IOP and system operation continues. The data path on the IOP is checked using techniques analogous to those described for the TMRC.

All peripheral controller-initiated bus transactions result in NonStop-V+ transactions that are translated into RIOB bus transactions by the IOP. In order to prevent errant peripheral controllers from writing or reading inappropriate global memory cells, the IOP contains an access and validation RAM (AVRAM). This AVRAM is a direct-mapped cache that translates virtual VME addresses into physical RIOB addresses. During the translation process, the IOP checks the permission bits in the AVRAM entry to see if the specific peripheral controller is permitted to read or write the appropriate physical RIOB address.

The power subsystem was also designed to be fault-tolerant. Batteries are provided to support continued system operation during power failures. Redundant bulk supplies drive independent 36 V_{DC} rails to protect against bulk failures. Redundant batteries drive dual 24 V_{DC} rails to protect against battery failures. The DC-DC converters use these four independent DC rails to produce the requisite DC voltage required by a specific CRU.

Software Architecture

Since one of the basic design goals of the system was to use an existing UNIX kernel as the basis for the operating system, the system software architecture was based on an industry standard implementation of UNIX. A major addition that was made in the kernel was a two-tiered memory management system to support the local/global bifurcation of the main memory system. This virtual memory system treats local memory as the main memory pool with global memory serving as a backing paging pool. Finally, pages in global will be swapped to disk if global memory is in short supply. The text, most of the data and per process stacks for the kernel reside in local memory. User processes text, data and stacks may also reside in local memory.

I/O operations are launched on behalf of user processes by the operating system. The operating system arranges for the data to be moved to the appropriate destination for the user process. If the source of the data for a write operation is in local memory then a buffer is allocated in global memory and the data is moved from local to global by the special purpose DMA hardware implemented on the CPC and TMRC. Similarly, if the destination of a read operation is in local memory, the data transfer performed by the operating system uses this DMA engine after the data is placed in global memory by the peripheral controller. The collection of services that assist in providing the illusion of fault-free operation to the system is called subscription based services. An entity in the operating system that required notification of an event calls a function to subscribe to the occurrence of that event. If the event occurs, a function specified by the caller is invoked with parameters that depend on the particular event type.

Faults that occur within the processor memory complex (CPC and TMRC) are known as core faults. The hardware guarantees that the operating system can continue to execute instructions to effect repair of the system, but the operating system must identify the faulting component and take the appropriate recovery action.

A core fault is indicated to the processors via a high priority interrupt. The processors then use the private-write mechanism to distribute a global view of the faulting condition. The private-write mechanism allows the processors to store private, possibly asymmetric, non-voted data in the global memory without causing the voting mechanism to register a fault.

Once a common view of the cause is obtained by each of the three processors, low-level exception processing code follows a deterministic parsing mechanism of the hardware status. This parse results in an implication of the offending CRU, which precipitates an action based on the type and severity of fault and the current configuration graph of the system. Typical actions are, in order of likelihood, incrementing a threshold counter, logging an event to the event reporting mechanism, or finally removing a CPC or TMRC.

Faults beyond the core, namely in the IOP or peripheral controllers, are handled by the I/O fault handling layer. IOP failures are typically recoverable. At any given moment a collection of controllers is bound to either IOP via the BIM, and the system can still access the controller via the alternate path.

A key to high availability in replicated systems is the ability to repair the system on-line. The operating system support for such repair activity is called reintegration. Consider CPC integration. A newly inserted CPC runs a power-on self-test (POST) from its local EPROM to verity the health of the board. The remaining processors agree to reintegrate and copy a small amount of local state to global memory. The processors then issue a soft reset (an operation supported by the RSB and TMRC, and hence voted) that results in all three CPCs entering reset. The CPCs notice that this is a soft reset operation and, after initializing some local state, find the communication block deposited by the operating system in global memory and load the program counter and stack pointer. The operating system now has control again and it proceeds to move a copy routine into each processor's local memory. Then, the processors use the local-global DMA engine to move pages from local to global and back, using the voting mechanism to bring the contents of all three local memory arrays into agreement. After this copying process is completed, normal execution resumes. Note that processing is suspended during the copy period of CPC reintegration. This is slightly more than one second on an 8-MB local memory system.

Unlike the processor, TMRC reintegration takes place during normal machine operation and only borrows cycles from the machine in small chunks that are controlled by the system administrator. The first phase of TMRC reintegration just copies global memory to a global memory buffer and back to global memory using the local-global DMA engine. During this phase, the IOPs believe that the replacement TMRC is off-line. The replacement TMRC returns "good" status for all RSB TMRC reads and writes.

In the second phase, the replacement TMRC remains a write-only memory. The processors lock the RIOB, copy a packet from global to local and back to global using the DMA engine. The RIOB is unlocked, and the process is repeated until the entire memory array has been restored. During the entire IOP revive process, both TMRCs accept writes from the CPCs and IOPs. This guarantees that the memories are consistent after the copying process is complete.

The standard UNIX operating system assumes perfect hardware and software. A failure in either the hardware or the kernel will result in an unconditional loss of all services (a "panic"). Given this collection of self-imposed constraints, a fault recovery model based on forward recovery was adopted. The kernel uses consistency checks as a fault-detection mechanism through approximately 1000 ASSERTs. An ASSERT is simply a macro that ensures that an expression is true. In the standard UNIX kernel, the failure of an ASSERT results in a system panic. Recovery from an assertion failure is provided using an assertion-specific forward recovery routine. These recovery routines are guided by data structure audit routines. Data structure audit routines determine the validity and consistency of various data structures.

A provably correct implementation of any version of UNIX has yet to be produced. A reliable "panic" mechanism was implemented to greatly increase the probability that various disk resident data structures are consistent upon reboot from an unrecoverable operating system fault. To accomplish this, the hardware write-protection feature is used to lock a number of critical kernel data structures during the panic procedure. Then, a number of kernel data structure consistency checks are performed and only those data structures that pass the various validity checks are subsequently utilized. The dirty blocks in the buffer pool are written to disk using a polled version of the disk driver. This ensures that a minimal amount of the system structure is used to accomplish the write operation. Finally, an image of the kernel is written to the disk using the PROMs. Experience shows that this procedure greatly increases the probability of the file systems being in a consistent state upon reboot.

In sum, the Integrity S2 incorporates numerous hardware and software techniques appropriate to a commercial, standards-dominated marketplace which demands fault tolerance. Fault tolerance has been accomplished without compromising the programmatic interface, operating system or system performance.

MAINTENANCE FACILITIES AND PRACTICES

Tandem's tools, facilities, and practices for hardware maintenance have evolved considerably in the last ten years. Over time, the trend has been increasingly to share maintenance responsibility with its customers, making it easier for customers to resolve hardware problems quickly on their own.

Early Methods

Early maintenance systems were based mainly on the use of on-line diagnostic tests to isolate the causes of readily apparent failures. Subsequent systems, however, moved toward the ability to detect failures automatically, analyze them, report upon them, and track their repair.

The first diagnostic and maintenance tools were very primitive. For example, to support the NonStop I systems, only a set of lights and switches was available on each processor for communicating error information and for resetting and loading the processor. In 1979, the Diaglink diagnostic interface was introduced to permit access to the system from remote maintenance sites. Diaglink featured an asynchronous modem. With Diaglink, customer engineers could remotely examine customers' systems, obtain system status by running operating system utilities, and execute diagnostics with customer assistance for remote, low-level debugging.

Operations and Service Processor (OSP)

The NonStop II system replaced Diaglink with an operations and service processor (OSP). The OSP was a microcomputer system that communicated with all processors and a maintenance console. The OSP offered all of the capabilities of Diaglink, as well as additional features to diagnose failures and to operate a system remotely. The OSP enabled operations and support personnel to obtain an internal view of the status of each processor.

The OSP communicates with the diagnostic data transceiver (DDT) included as a part of each processor module in the system. This communication allows the operator to diagnose software and hardware problems through the operator's console. The DDT monitors the status of the Dynabus interface, I/O channel processor, memory, and IPU, including the internal data paths. For example, the DDT enables the operator to put the processor in single-step mode and monitor the contents of its registers before and after execution of a specific instruction.

The OSP includes a built-in modem that can connect it to a remote terminal or to another OSP. This connection allows an operator or customer engineer to diagnose and possibly even correct problems from the remote site. A remote customer engineer can, for example, run microdiagnostics residing on a local OSP. Alternatively, the customer engineer can download diagnostics from the remote OSP to the local OSP, remotely reset and load processors, and display error information. For the TXP system, the OSP was enhanced to include an asynchronous modem, improved microdiagnostics, more remote operations capability, and additional remote support capabilities.

Tandem Maintenance and Diagnostic System. For the VLX system, the Tandem Maintenance and Diagnostic System (TMDS) replaced the OSP. TMDS provides a framework for problem detection with the VLX system, which was intended to reduce the cost of ownership in various ways [White, 1987]. A major aspect of this attack on costs was improved diagnostic and maintenance facilities. TMDS permitted the elimination of front panel lights and switches from the system design, dramatically streamlining maintenance activities. Unlike its predecessors, TMDS operated on line without requiring significant system resources. It provided a uniform interface to many diagnostic subsystems [Troisi, 1985].

By the time of the VLX system, the maintenance strategy had evolved beyond real-time monitoring of system components to include automatic on-line fault analysis and automatic dial-out for on-line support by remote service centers [Eddy, 1987]. TMDS was based on that strategy. Today, although it is known primarily for its use on the VLX, CLX, and Cyclone systems, TMDS is compatible with all NonStop systems. It runs under the Guardian operating system and is distributed automatically to all customers.

Through pervasive instrumentation, an internal fault-tolerant maintenance and diagnostic subsystem continuously monitors the system's processors, power supplies, and cabinet environments. When the Guardian operating system or an I/O process detects a change of state or an irregular event, it writes an event signature to an event log on disk. Then, TMDS examines each event signature. If further study seems advisable, TMDS starts a module known as an automatic fault analyzer. Thus, TMDS supports both active testing of components and symptom-based fault analysis.

TMDS fault analyzers relieve the customer of the need for an intimate knowledge of hard- ware, status codes, or specific error values. TMDS uses if-then-rule–based algorithms to evaluate events against a knowledge base, automating many of the detection, interpretation, and reporting tasks previously required of a console operator. The knowledge base contains a range of accept- able component states and environmental factors. If the fault analyzer finds that an event falls within the acceptable range, TMDS saves the fault analysis in a local log (a catalog of system events and patterns that can aid future troubleshooting).

However, if a fault analyzer detects an event that suggests an active or potential problem, TMDS transmits a signal to a fault-tolerant service processor called the remote maintenance interface (RMI). The RMI consists of dual Motorola 68000–based processors that communicate with each other and with other subsystems over dual bit-serial maintenance buses. The proces- sors, FOX controllers, and power supply monitors all connect to the maintenance buses. The RMI supports all the functions of the old OSP, but does so as a much more compact unit—two circuit boards residing in one of the cabinets (VLX and Cyclone) or a part of the MFC (CLX). Through a synchronous protocol, a special communication process, and a password requirement, the RMI also greatly reduces the risk of unauthorized users gaining access to the system through the diagnostic facility.

When it receives a problem signal, the RMI alerts the on-site operator and, on the CLX, VLX, or Cyclone system, optionally dials out to a Tandem National Support Center (TNSC). Tandem staffs two such centers: one to service sites in North America and one for sites in Europe. Other TNSCs are planned as business needs for them develop.

Through the RMI, either the on-site operator or the remote analysts and engineers at the TNSC can review the event log, run diagnostics, test components, and isolate and diagnose the problem. On newer equipment, these actions include detecting out-of-spec intake and exhaust cabinet temperatures, malfunctioning disks or tape controllers, and faulty fans. TMDS also uses processor diagnostics to test power supplies, clocks, and batteries. If necessary, the TNSC can dispatch a field service engineer for on-site troubleshooting or part replacement. The TNSC staff has identified and diagnosed the problem, so the service engineer is very likely to arrive with the correct replacement part in hand.

TMDS also allows analysts and engineers to run on-line diagnostics to identify problems that fault analyzers don't cover. In fact, many diagnostics can be run while the device that is being studied is on line. In the worst cases, only the problem device needs to be shut down; under previous diagnostic systems, both the device and its controlling processor needed to be shut down. In any case, testing with TMDS only minimally affects the system's performance. Processing continues unhindered by the diagnostic tests, unless a processor itself is being evaluated.

The following steps illustrate how TMDS operates if a tape I/O process detects an error event involving a tape unit:

1. The tape I/O process immediately creates an error event and sends it to the TMDS event log.
2. TMDS signals the fault analyzer.
3. The fault analyzer localizes the error to a particular controller board.
4. The fault analyzer writes additional error information to the event log, specifying the prob- able FRU, the controller address, and the terminal error code. All of these actions take place within seconds.
5. After completing the analysis, TMDS dials out to the TNSC.
6. The TNSC dials back in to verify the analysis.

7. The TNSC dispatches a customer engineer to replace the controller board.
8. TMDS records the replacement in the event log.

TMDS event logs and the reports generated by the remote intervention are archived in a centralized support database. This database contains a history of service requests, diagnoses, and support actions for hardware and software. Experts periodically scrutinize the database, seeking out diagnostic patterns and irregularities that they can use to improve system maintenance.

TMDS continues to evolve, incorporating many new features. One of these features is a built-in self-test (BIST) method that uses pseudo-random test vectors and scan-path design [Garcia, 1988]. On the CLX, the pseudo-random test covers several custom ICs, commercial MSI logic, a static RAM array, and their interconnects. The BIST also does a functional test of the dynamic RAM main memory and its control logic. The test is controlled by maintenance processor software, simplifying the processor board hardware dedicated to the BIST. With the BIST, hardware problems on the CLX processor can be detected and reported to TMDS without requiring downloaded, handwritten diagnostics.

In the Cyclone system, the power and environmental monitoring facilities have been significantly enhanced. In addition to sensing more components (voltages both on the circuit boards and at the power supply outputs, intake and outlet air temperatures, battery condition, and fan rotation), sensors are polled much more frequently, and most sensors are replicated to allow differentiation between a failing component and a failing sensor. In addition, Cyclone's maintenance subsystem can detect the physical presence or absence of many components, such as cables, power supplies, fans, and bus terminators. Both the logical address and physical location (which cabinet and which slot within the cabinet) are automatically available to the maintenance subsystem so that failed components can be easily identified and reliably replaced.

SOFTWARE

Overview

In the preceding discussion of the evolution of the Tandem system, the careful reader will find no mention of fault-tolerant hardware modules. In fact, one of the primary design criteria for Tandem hardware is to make it *fault intolerant.* Any incorrectly functioning hardware module should detect the problem and, as quickly as possible, shut itself down. There are a few exceptions to this rule, such as error-correcting main memory, but the fundamental design of the Tandem system is to connect *fail-fast* hardware with fault-tolerant software.

Fault tolerance normally implies hardware redundancy. While this is true for a Tandem system, the net additional cost to the customer has been kept surprisingly low due to the many innovative features of the Tandem system. In most cases, the redundant modules each perform useful work in their own right and, except when they have failed, contribute to the capacity of the system. Failing modules can be replaced while the system is running. The net effect of a hardware failure is, at worst, a short period of slightly degraded performance. A system with a normal amount of excess capacity will survive a failure without any noticeable effect.

The key to fault tolerance without wasted redundancy is the Guardian operating system. The following sections describe the many components of Guardian, from the kernel (process and memory management and message system) to the transaction manager, networking support, and NonStop Structured Query Language (SQL). Each makes an important contribution, not only to the functionality of the Tandem system, but also to the support of fault tolerance.

Fault tolerance would be of little value without data integrity; business demands accurate record keeping, and an inconsistent database is often worse than no database at all. Furthermore, a business that depends on its computer system must be able to grow that system at least as fast as the business. The following sections also describe how the system has been engineered to prevent data corruption and to provide expandability.

Guardian: An Integrated OLTP Operating System

A basic difference between Guardian and other systems is the very high level of software integration. Although there is the usual layering of software function, these layers are relatively closely tied together and interdependent. This approach has its costs, but the resulting efficiency, coupled with a high level of functionality, is unique in the computer industry. There are many software components that contribute to the fault tolerance, data integrity, expandability, and basic functionality of the Tandem system. In this section, we will give a general overview of the Guardian elements that differentiate Tandem from other systems.

• The *kernel* includes the usual support for the management of processes, memory, interprocess communication, names, time, peripheral I/O, process synchronization, and debugging. In addition, the kernel detects failures of any processor, interprocessor bus, or I/O channel and performs recovery. Several innovative techniques are used to synchronize the independent processors and to provide a consistent view of time and system state, despite failures and communication delays. Finally, the kernel supports the management of process pairs, which are the keystone of both hardware and software fault tolerance.

• The *file system* hides the distributed nature of the system from application processes and presents a conventional view of peripheral I/O devices. Communication with I/O devices and other processes is accomplished without regard to the location of the resource, be it in the same processor, another processor in the same system, or a processor in a remote system. The file system also provides checkpoint and retry mechanisms for hiding the effects of hardware and software failures from the application process.

• *I/O processes* manage peripheral devices and react to component failures by routing access over working paths and devices and then notifying operators and customer engineers about the need to repair or replace the failing component. I/O processes receive messages from application processes (via the file system) and perform the requested operations on physical devices. In the view of the application programmer, the I/O process and the device it manages are indistinguishable. There are dozens of different I/O processes, each designed to manage a particular class of device.

• The *disk process* is probably the single most important component of the system. It provides data integrity and reliable access to data despite processor, channel, controller, or media failure. It supports mirrored disks efficiently, making good use of the dual access paths to data. It supports four types of file structures, as well as SQL tables; it supports file partitioning, alternate indices, data security, and main memory cacheing for both reading and writing of data. It supports full transaction management, including two-phase locking and two-phase commit protocols. Last, but far from least, it can execute those parts of SQL queries that require data local to one disk, greatly reducing message traffic for many applications.

• The *transaction management facility* (TMF) coordinates the processing of disk accesses and updates, ensuring the requirements for atomicity, consistency, isolation, and durability. An application can, in a very simple manner, request multiple database updates in many physical locations, possibly thousands of miles apart, and be assured that either all or none of them will be performed; during the transaction, other applications will always have a consistent view of the database, never being able to see only some of the updates and not others. TMF protects the database from total media failure (including the loss of a mirrored disk pair) through the technique of on-line dumping and roll-forward of the transaction log. TMF also supports the remote duplicate database facility, which can quickly recover from the loss of an entire computing facility.

• The *transaction processing monitor* (Pathway) provides a flexible method for managing customer applications in a distributed system. Pathway automatically distributes application processes (called servers) to the available processors and, in the event of a processor failure, redistributes the applications to the remaining processors. Any work that was lost or compromised by the failure is automatically restarted after being rolled back to its initial state. Customer programming is straightforward and is not required to perform any special operations to achieve full fault tolerance.

• The *network control process and line handler processes* (Expand) manage communications between a system and a network of other Tandem systems. To a user on one node of a network, the rest of the network appears as a seamless extension of the local system. The requirement for local autonomy may impose access barriers, and communication delays may impose performance penalties; otherwise, it is as easy to manage distributed applications and databases as it is to manage local ones.

Fundamental System Structure

A Tandem system is composed of from 2 to 16 independent processors connected by a dual, high-speed interprocessor bus (IPB). Guardian, Tandem's proprietary operating system, has two primary responsibilities: (1) to maintain a consistent view of the system while allowing each processor to exercise independent autonomy, and (2) to provide general services for its clients and the processes and particularly to provide an efficient and reliable means of communication between them.

The first responsibility of the operating system requires that each processor establish and maintain communication with the other processors of the system. Continuous availability of the IPB is a fundamental assumption, since the processors must coordinate many operations and notify each other of changes in system state. If any two processors are not able to communicate with each other for any period, it is likely that they will have inconsistent views of the system state; one or the other must be restarted. Thus, a dual (fault-tolerant) IPB is an important requirement.

Except for the lowest-level functions of process dispatching, message transmission, and interrupt processing, all work in the system is managed by one or more processes. System processes manage process creation and termination, virtual memory, security, performance measurement, event management, diagnostics, and debugging. I/O processes manage access to peripheral devices and are responsible for dealing with failing components. Application and utility processes direct the operation of the system towards some useful purpose.

Messages are the primary method for process-to-process interaction, eliminating the need for applications to deal with the multiple-computer organization of the system. To applications, the system has the appearance of a conventional uniprocessor programmed in conventional programming languages such as COBOL85, Pascal, C, and FORTRAN. Processes interact with one another using a client-server protocol typical of the remote procedure call protocols that are common in workstation-server LANS today. This client-server design is well accepted today, but fifteen years ago it was considered novel.

Fault tolerance is provided by duplication of components in both the hardware and the software. Access to I/O devices is provided by process pairs consisting of a primary process and a backup process. The primary process must checkpoint state information to the backup process so that the backup can take over if a failure occurs. Requests to these devices are routed using the logical process name so that the request is always routed to the current primary process. The result is a set of primitives and protocols that allow recovery and continued processing in spite of bus, processor, I/O controller, or I/O device failures. Furthermore, these primitives provide access to all system resources from every process in the system.

Initialization and Processor Loading. System initialization starts with one processor being *cold loaded* from a disk attached to that processor; any processor can be used for this operation, as long as it is connected to a disk with a copy of the operating system image file. The image file contains the code of the kernel and the system processes that are automatically started when each processor is loaded. Once any processor is loaded, it is then used to load the other processors via the IPB. All processors other than the first can be loaded in parallel. Should a processor fail or be removed for maintenance, it can be reloaded by any other processor. There is no essential difference between an initial load of a processor and a later reload of a processor after it has been repaired. A processor reload operation does not interfere with the operation of application processes.

Each processor receives an identical copy of the kernel and other system-level software, but a different processor configuration, depending upon the peripheral devices attached to the processor. Each processor will start the appropriate I/O processes to manage the attached devices. Once a processor's software and configuration is loaded, it passes through a phase in which it is synchronized with the other processors. Basically, this synchronization involves transmitting a consistent copy of the system state to the processor. Once this is accomplished, the processor becomes a fully independent entity. There is no master processor.

Processor Synchronization. Although Tandem computer systems are designed to tolerate any single fault, either hardware or software, multiple failures do sometimes occur, either through multiple hardware errors (unlikely), software errors (likely), operation errors (more likely), or during maintenance periods. Even when multiple faults occur, only a portion of the system (some disk volumes and/or processors) becomes unusable until it can be repaired and reintegrated into the system.

Two different mechanisms are provided for dealing with multiple faults: processor synchronization and transactions. The following sections describe the issues of processor synchronization. In addition to a simple algorithm to detect processor failure, Guardian also has three more complex algorithms to ensure that all processors of a system have closely agreeing views of processor configuration, replicated data, and time. Each of these algorithms requires a minimum of communication and is sufficiently robust without resorting to the cost and complexity of solving the Byzantine Generals problem.

I'm-Alive Protocol. Once two or more processors are loaded, they must continually check on each other's health. Because the processors are designed to be fail-fast, they respond to any internal error by shutting down completely and refusing to communicate with the outside world until receiving an explicit signal that they are ready to be reloaded. Thus, it is up to the remaining processors to detect a failure through the absence of any response from the failed processor.

Using the I'm-Alive protocol, each processor transmits a short message to each processor, including itself, at least once every second, over each of the two IPBs (the message to itself verifies that the sending and receiving bus circuitry is functioning). Every two seconds, each processor checks to see that it has received at least one message from each processor in the interval. Any missing messages imply that a processor is not healthy and has probably failed. Normally, any processor would immediately declare such a processor down and proceed to cancel all outstanding communication with that processor.

Regroup Protocol. Experience showed, however, that there are rare instances in which one processor might merely be a little late in sending out its I'm-Alive messages. This situation usually occurs when recovery from power failure or other high-priority error recovery momentarily usurped a processor. Because the I'm-Alive intervals are not synchronized between processors, a late I'm-Alive might result in a processor's being declared down by some processors and not by others. Such a case, termed a *split-brain* situation, could lead to a lack of database integrity. Thus, a Regroup algorithm was implemented to handle these cases with as little disruption as possible. In essence, the slow processor is given a second chance. Whenever any processor detects a missing I'm-alive message, all processors (including the suspect processor, if able) exchange messages to decide which processors are healthy. After two broadcast rounds, the decision is made and all processors act on it.

Global Update Protocol. Certain information, notably the destination control table (described later), is replicated in each processor and must be identical in each processor. Updates to replicated information, however, originate in multiple processes and multiple processors. Consistency of an update depends upon *atomicity* [Gray, 1979], which demands that (1) any update is completed within a maximum time, (2) either all replicated copies are updated or no copy is updated, and (3) all updates occur serially.

In Tandem systems, atomic update is guaranteed by the Global Update protocol [Carr, 1985]. All such updates are performed by the Guardian kernel so that high-priority processes cannot delay the completion of an update within a maximum time. All updates must first be sent to a locker processor, which ensures that updates occur serially and also ensures that an update is propagated to all processors even if the originating processor fails, including simultaneous failure of the updating and locker processors.

Time Synchronization. An OLTP system is designed to record and control real-world events at the time they actually occur. An important part of the information processed is the current time. Furthermore, it is important that the sequence in which events occur can be reconstructed from timestamps associated with each event.

Although it is clearly difficult to have coordinated clocks in a widely distributed system, initial attempts to synchronize time on a local Tandem system showed that this is not a simple problem either. Although it is no great problem to keep clocks within seconds of each other, synchronization of a multiprocessor system requires that no message that can be sent from processor A, with A's clock time T, should arrive at processor B before B's clock time has reached T. Otherwise, time would appear to move backwards.

A novel algorithm [Nellen, 1985, 1986] passes time-adjustment packets from processor to processor; each processor not only adjusts its own clock to the average time of all processors, but it also calculates its clock error relative to the average and makes adjustments on a continual basis. This algorithm ensures that the average time does not fluctuate wildly when a processor fails and is replaced by a processor with a different speed clock.

The provision for an external clock can extend the local synchronization algorithm to geographically distributed Tandem systems. Electronic clocks that monitor a broadcast time standard can keep Tandem systems synchronized to less than the time it takes them to communicate with each other.

Guardian Kernel Services

Now that the basic problem of maintaining consistency between the distributed processors has been addressed, we can turn to the performance of useful work. As in all modern systems, Guardian supports the concurrent execution of multiple independent processes. What distinguishes Guardian is its heavy dependence on messages to coordinate the operations of processes. Messages are essential for the operation of both system-level software and customer applications.

The design of the system was strongly influenced by Dijkstra's "THE" system [Dijkstra, 1968] and Brinch Hansen's implementation of a message-based operating system nucleus [Brinch Hansen, 1970]. Both hardware and software have been optimized to facilitate the sending of messages between processors and between processes. The heavy dependence upon messages, in preference to other communication and synchronization mechanisms, has been very important in the design of a system that is both distributed and smoothly expandable. Customer applications can be easily grown by simply adding more processors, disks, and other peripherals, without changing software or even reloading the system.

Because of the message-based structure, the applications are unaware of the physical configuration of the system. An application accesses a directly connected peripheral and a remotely connected peripheral in exactly the same way: Messages are exchanged with the peripheral's manager (an I/O process); the I/O process is also unaware if the requestor is in the same or a different processor.

Efficient messages have also been a key element in implementing fault tolerance. System-level software uses messages to *checkpoint* information critical to data integrity and continued operation in the event of a failure. Applications are generally unaware that a failure has occurred because messages can be automatically rerouted from a failing component to a functioning one.

Processes

A process is an independently executable entity that consists primarily of shareable program code, private memory, and the process state vector. The process state vector includes the program counter, registers, privileges, priority, and a microsecond timer that is incremented only when the process is executing.

Once a process begins execution in some processor, it remains in that processor until it is terminated. Each process in the system has a unique identifier or name by which other processes may communicate with it on a network-wide basis. Messages to a process are addressed to its identifier, and so the identifier provides a network-wide and fault-tolerant method for process addressing.

Guardian maintains a destination control table that is identical in all processors of a system. This table relates the name of a process with its location (that is, its processor and process number) so that messages to a process can be routed in an efficient manner.

Processes scheduling uses a pure priority mechanism with preemption. Scheduling is round robin within a priority class. Considerable care is taken to avoid the priority inversion problem in which a low-priority client makes a request of a high-priority server, thereby promoting its work to high priority. This problem is unique to message-based systems and must be solved in order to provide a global priority scheduling mechanism.

Memory Management

Each process has a virtual address space containing its program and a process data stack. A program consists of *user code* and an optional *user library*. The user library is an object code file that can be shared by processes that execute different user code program files. All processes within a processor executing the same program share memory for object code. The process data stack is private to the process and cannot be shared.

Processes may allocate additional *extended memory segments,* which have a maximum size of 127.5 MB each. A process may access its data stack and one extended segment concurrently. If multiple extended segments are allocated, the process must explicitly request that a particular segment be placed in use when it is required.

Data in extended segments may be shared with other processes in two different ways:

1. A read-only segment is a method of accessing the contents of a file as if it were main memory, using virtual memory to load the information on demand. Such segments can be shared among all processes in all processors, although multiple copies of the data may exist in different processors.
2. A read-write segment can be shared only by processes in a single processor, since it would be impractical to update multiple copies in different processors.

The sharing of read-write data among customer application processes is discouraged so that fault containment is maintained and so that the system load can be distributed by the placing processes in idle processors. Guardian does not provide interprocess concurrency control (other than via messages) necessary for coordination of updates to shared memory.

The Guardian memory manager supports virtual memory using demand paging and a clock replacement strategy [Carr, 1982]. It intentionally does not support load (thrashing) control, as the performance requirements of on-line transaction processing do not permit the paging or swapping of processes commonly found in interactive time-sharing systems.

Interprocess Messages

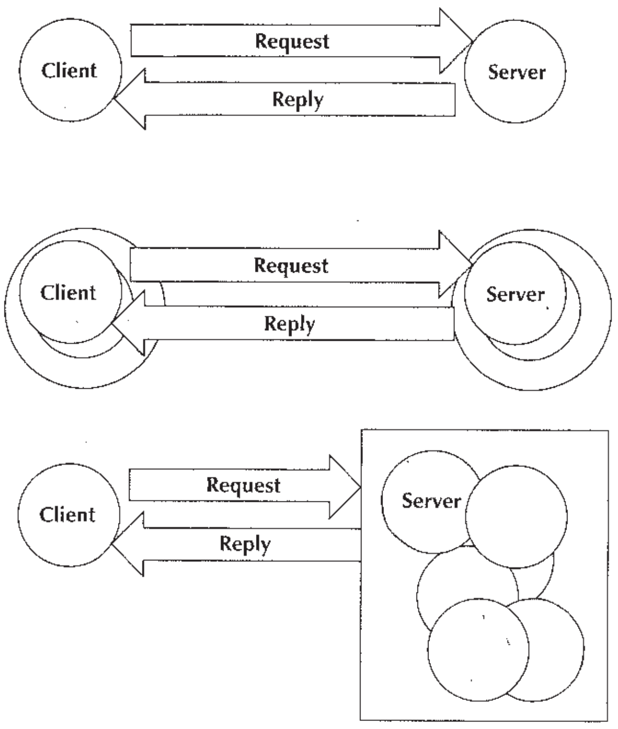
The basic message system protocol follows the requestor-server model. One process sends a message with a request to another process and waits for a response. The second process services the request and replies with a result as depicted in Figure 8–60. This is considered to be a single message and can also be viewed as a remote procedure call. Naturally, this basic mechanism also suffices for simple datagrams or for transferring bulk data (up to 60 KB per message) in either direction.

Multi-threaded requestor processes may have many messages outstanding to collections of server processes. Multi-threaded server processes may accept multiple requests and reply to each in any desired order. Any process may operate concurrently both as a client and as a server. For application processes, access to the message system is through the file system, which provides some protection from abuse of privileged mechanisms and simplifies the interface for sending messages. System processes, on the other hand, generally use the message-system primitives directly, as these provide better performance.

The simplest use of messages is for application processes to access peripheral devices; the programmer is generally unaware that messages are involved, as the program makes a simple read or write of data on a file. If, for example, the file is a disk file, then the read/write request will send a message to the manager of the disk, known as the *disk process.*

It is simple for a process to masquerade as a device manager. An example of this is the printing spooler process: Applications send print output messages to a spooler (which pretends to be a physical printer process) and stores them for printing later; the spooler then sends the output, again via messages, to a true physical printer process. The application programmer need not be concerned with whether the message system routes the messages to the spooler process

FIGURE 8–60
Process concepts in the Guardian operating system



Each process has a program and private storage; they communicate via messages.

A process pair is one logical process; the backup process is idle until the primary fails.

A server class is a group of processes spread over many processors for load balancing.

or directly to an I/O process controlling a physical printer; only the process name must be changed to choose the destination process. A more interesting use of messages is in the structuring of applications as a network of communicating processes, which is described in the section on Pathway.

In the Tandem system, messages are a fundamental mechanism for providing fault tolerance:

• The *communication protocol* for the interprocessor buses tolerates any single bus error during the execution of any message-system primitive. A communication failure will occur only if the sender or receiver processes or either of their processors fails. Any bus errors that occur during a message-system operation are automatically corrected without involving the communicating processes.

• The *process-pair mechanism,* as described in the next section, provides fault-tolerant access to peripheral devices despite processor, channel, or even most software failures. A request message that is being processed by the failing component is automatically resent to the backup component; the application is not even aware that this has happened and need not make any provision for it.

Memory moves rather than the interprocessor buses are used for communication between processes in the same processor, but there is no apparent difference to the communicating processes. In addition to messages between processes, Guardian also implements simpler *control* messages, which are for communication between processor kernels. Control messages are used as a basis of the full message-system protocol, as well as an inexpensive mechanism to maintain synchronization between processors.

Guardian and the IPB are highly optimized for the processing of interprocess messages, especially for short messages of 2 KB or less. A message between processes in different processors is only marginally more expensive than an intraprocessor message. In fact, an interprocessor message usually has a shorter *elapsed* time than an intraprocessor message, since both sender and receiver processes can execute in parallel.

A final advantage of the message system is its transparent support of both short- and long-haul networks. Except for the inevitable communication delays, the client and server can detect no apparent difference between accessing a local disk file (or process or printer) and a remote one [Uren, 1986]. The message-system and file-system protocols are precisely the same in both the local and remote cases.

Tolerating Software Faults

Systems whose fault tolerance is based solely on hardware mechanisms can be designed to provide high reliability and to continue to function in the presence of hardware component failures. Unfortunately, a high percentage of computer system failures are due to software.

Unlike the situation with hardware components, it is possible to develop perfect, defect-free, failure-proof software. It is only a matter of cost to the manufacturer and inconvenience to the customer, who must wait much longer for some needed software to be delivered. In the commercial world, customers demand a continuous flow of new software and improvements to old software. They demand that this be done quickly (more quickly than the competition) and at a reasonable price. New software systems are inevitably more functional and more complex than the systems they replace.

The use of structured programming and higher-level languages has not eliminated software errors because they have enabled the building of larger and more complicated programs. Methods to improve software quality, such as code inspections and structured testing techniques, are effective, but they only reduce the number of errors; they do not eliminate them. Therefore, in practice, even with significant care taken in software development processes, software faults are inevitable. In fact, as previously stated, software failures are typically more common than hardware failures.

Software fault tolerance leads, indirectly, to better software quality and data integrity. At Tandem, system programmers are encouraged to make numerous consistency checks and, if a problem is detected, to *halt* the processor. (Tandem's system software probably has one of the highest densities of processor-halt instructions in the industry.) The system programmer knows that, for almost all consistency problems, the backup processes (described in the next section) will continue to provide service to the customer. This consistency checking has two direct effects:

1. When contamination of system data structures is detected, the processor is immediately shut down, reducing the chance that the database can become contaminated.
2. All significant errors in system software become very visible, and since the entire processor state is frozen and dumped when an error is detected, it is easier to uncover the cause of the error. Thus, errors that affect system stability and data integrity are found and corrected in a very timely manner. The result is higher-quality software and fewer failures that need to be tolerated.

Process pairs provide fault-tolerant execution of programs. They tolerate any single hardware fault and most transient software faults. (Transient software faults are those caused by some untimely combination of events, such as a resource shortage that occurs at the same time that an I/O error must be handled.) Most faults in production software are transient [Gray, 1985], since the simpler programming errors are weeded out during testing. Process pairs allow fail-fast programs to continue execution in the backup process when the software bug is transient.

Process Pairs

The key to both hardware and software fault tolerance is the *process pair* [Bartlett, 1981]. A process pair consists of a *primary* process, which does all the work, and a *backup* process, which is passive but is prepared to take over when the primary process fails. (See Figure 8–61.) This arrangement is analogous to having a standby processor or computer system, except that, if properly arranged, the cost of the backup process is far less than the cost of the primary process. Generally, the memory requirements and processor time consumption is a small fraction (usually about 10 percent) of the primary process.

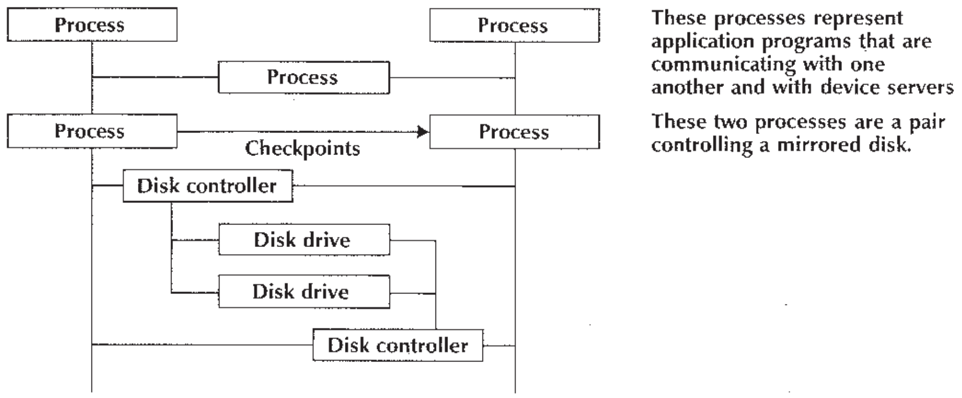
A typical process pair is started in the same way as an ordinary process: as a single process executing in a single processor. The process then notifies the operating system that it would like to create a clone of itself, using the same program file, in another processor. The second process is also started in a very ordinary fashion, but with two small differences:

1. The second process receives an initial message that informs it that it is a backup process; the process then goes into a passive mode, accepting messages only from the primary or from Guardian (to notify the backup that either the primary process or its processor has failed).
2. Both the primary and backup process share a single name, and for each name, the destination control table registers both the primary and backup processes; all communication to the process pair is routed to the primary process.

While the primary process executes, the backup is largely passive. At critical points, the primary process sends *checkpoint* messages to the backup.

Checkpoint messages have two different forms; a process pair will normally use either one or the other, but not both. For application software, a simple form of checkpointing is provided. Checkpoints copy the selected process state, at carefully chosen takeover points, from the pri-

FIGURE 8–61
Diagram of system structure


Process
Process
Process
Process
Checkpoints
Process
Disk controller
Disk drive
Disk drive
Disk controller
These processes represent application programs that are communicating with one another and with device servers
These two processes are a pair controlling a mirrored disk.

mary to the backup. Updating the process state of the backup is performed solely by the system software at the command of the primary process; the backup process is completely passive.

At a checkpoint, usually immediately before or after some important I/O operation, the primary and backup are functionally identical. If the primary fails, the backup begins executing at the point at which the last checkpoint occurred. For most system software, such as I/O processes, checkpoint messages contain functional updates to the process state. The backup processor must interpret these messages and apply appropriate updates to its own local data structures; this is sometimes referred to as an *active* backup. Although the program code is identical, the contents of memory can be entirely different. A typical checkpoint would indicate that a file has been opened or closed, but, in the case of the disk process, most updates to important files also will involve a checkpoint.

The active backup approach is more complicated, but it has several advantages. Less data is transmitted on each checkpoint, and information that the backup can obtain from other sources (such as a disk file) need not be checkpointed. For software fault tolerance, the active backup is better because the backup must manage its own state, and errors in the primary process are less likely to contaminate the backup [Borr, 1984].

Since the process pair shares a single name, it appears to all other processes as a single process. When a message is sent to it, the message system automatically consults the destination control table to determine which process is primary and routes the message to that location. When the primary process fails for some reason, the backup becomes the primary, the destination control table is changed, and all messages are directed to the new primary process. The new primary has already received checkpoints that describe the current openers of the process, so those openers need do nothing to re-establish communication.

There are potential race conditions in which a server process pair has performed some request and has not replied to the client when a failure occurs. When the outstanding request is resent to the new primary process, it might perform the same operation a second time, possibly causing an inconsistency. This problem is eliminated by associating sequence numbers with each request and having the new primary simply make duplicate responses to already-processed requests.

Similar problems could occur if a client process pair performs a checkpoint and then makes requests to a server. If a failure occurs, the backup client takes over and makes duplicate requests to the server. These requests are handled using the same sequence numbering scheme. During normal operation, the sequence numbers are used for duplicate elimination and detection of lost messages in case of transmission error [Bartlett, 1978].

I/O Processes

Most processes can execute on any processor. The exceptions are I/O processes because they manage devices, and each device is connected to two processors via dual-ported device controllers. When a device is configured, an I/O process pair is automatically defined for the device; one member of the pair resides in each processor connected to the device. The configuration parameters may state which processor is the normal primary access path to the device, but this choice can be altered by the operator for testing and performance tuning. In the case of a processor, channel, or controller port failure, the process that still has access to the device will take over control, and all application requests will be routed to it.

When a request for an operation such as a file open or close occurs, the primary process sends this information to the backup process through the message system. These checkpoints ensure that the backup process has all information needed to take over control of the device. Process pairs provide a uniform way to access I/O devices and all other system-wide resources. This access is independent of the functions performed within the processes, their locations, or their implementations. Within the process pair, the message system is used to checkpoint state changes so that the backup process can take over in the event of a failure. A process pair for a mirrored disk volume appears in Figure 8–61.

Disk Process

Although the overall design of an OLTP application may be very complex, it is essentially composed of many simple servers that require a minor amount of computation and perform a large number of database accesses and updates. To achieve high performance in this environment requires a great deal of sophistication in the design of the database management software. In this section, we are able to give only a broad outline of the myriad responsibilities and functions of the disk process, but it clearly has the most demanding task of any component of the system.

Each disk process pair must manage a pair of mirrored disks, each connected to two controllers, which are in turn connected to two processor channels. Thus, there are eight possible paths to the data, any component of which may fail; even in the rare case of a multiple failure, the disk process must attempt to use all available resources to provide continuous access to the data.

Mirrored Disks. Mirrored disks are, as the name implies, identical images of one another. It would appear that this is a situation in which fault tolerance requires a redundant (and expensive) resource that contributes nothing to system capacity. (The value of data integrity in most cases, however, justifies the expense of the redundant disks.) Fortunately, however, even though the redundant disk does not contribute to storage capacity, it usually contributes significantly to processing capacity. When data must be read from disk, the disk process can use either of the two disks, usually the disk that offers the shorter seek time. Multiple read requests can be processed concurrently, and, if one disk is busy, then the other disk can be used. Because duplexed disks offer shorter seeks, they support higher read rates than two ordinary disks [Bitton, 1989].

Any write operation must be made to each of the mirrored disks and requires that both disks seek to the same cylinder (thereby reducing the chance of having a short seek on the next read). Consequently, disk writes are considerably more expensive than reads, but, when performed in parallel, they are not much slower than a write to a single disk. The proper use of disk cache, particularly when protected by transaction management, can eliminate a large majority of disk writes without sacrificing data integrity.

Customers who consider all or part of their database to be a noncritical resource may have unmirrored disks on a disk-by-disk basis. Modern disks are very reliable, and, even when drives fail, it is exceedingly rare that the data are lost. Many activities, such as software development, would not be seriously impacted by the rare unavailability of a disk.

Disk Cache. Each disk process can be allotted many megabytes of main memory for disk cache. In 1990, the upper limit was 56 MB per disk volume, but this will steadily increase, along with the size of main memory. The disk process uses the cache to hold recently accessed disk pages and, under normal circumstances, can satisfy most read requests without performing any physical I/O operation. The disk process can service many requests concurrently, so it is not unusual for it to satisfy a half-dozen requests from cache while it is performing a single real disk I/O.

The worst case for cache management is an application that performs random accesses to a very large database. The probability that it re-uses a recently accessed page in cache is quite low. Experience, however, shows that the typical application usually has, at most, one such file in addition to numerous smaller files that are also accessed in each transaction; thus, the ratio of cache-hits to physical I/Os remains quite good. Even in the case of the large file, traversing the B-tree index structure to find a data record might normally require three physical I/Os that can be saved by cacheing the index blocks.

Although cache has an obvious benefit in reducing read I/O operations, the situation is not so clear with write operations. There are many situations in which disk file updates are made to blocks that were recently updated; if only part of a block is being updated, this clearly saves a read to get the unchanged parts of the block. More significantly, if updated blocks could be kept in main memory and written to disk only when convenient, many disk writes could be eliminated. On a conventional system, we couldn't do this because a processor failure would lose a large number of disk updates and corrupt the entire database.

In Tandem systems, we might consider checkpointing disk updates to the backup disk process, which is much cheaper than performing an actual I/O. If a processor failed, the backup could make sure that the updates were written to disk. This approach, however, would not be safe enough, since it is possible for a long power failure, or simple bad luck, to cause a multiple failure and loss of an unacceptable number of database updates. Luckily, there is a solution to this problem, but we must postpone its discussion until we have covered the basics of transactions.

Partitioning. If an OLTP application has a high transaction rate and each transaction accesses a particular file, the load on the disk process, even with perfect cacheing, may be too large to sustain on a single processor. As with the application, it is necessary to be able to distribute the database access load across the processors easily. (In general, dynamic load redistribution has not proved necessary, but it must be easy to redistribute in a static manner.)

The concept of file partitioning is not new, but the disk process and file system cooperate to provide a simple and flexible solution. Any file can be simply partitioned by issuing a few commands specifying the key ranges of each partition. To the applications, the partitioned file appears as a single file, and the file system redirects any read or write request to the proper partition, depending on the key of the record. If a partition becomes too full, or too busy, it can be split by subdividing its key range and moving the appropriate records to the new partitions.

The partitions of a file can be distributed throughout a network, and, thus, a distributed database can be created and maintained in a manner that is completely invisible to the applications, while maintaining excellent performance when accessing local data. For example, one could easily construct a 50-partition file, one for each of the United States, and physically locate each partition on a separate Tandem system in each state. On each system, local state data could be processed very efficiently, but every application process could have access to the full database as if it were a single file, subject only to the inevitable communication delays.

Locking and Transaction Protection. An OLTP application may have hundreds of independent servers performing concurrent accesses to the same set of files; the disk process must support file and record locks, for both shared and exclusive access. Locks can be obtained through explicit application request or through the operation of transaction management, which automatically provides atomicity and isolation of all database accesses in a transaction. More about transactions appears later.

File System. The file system is a set of system routines that execute in the application process and manage communication with I/O processes and other application processes. For access to I/O devices, the file system hides the process-and-message structure, and the application program appears to be issuing direct requests to a local I/O supervisor. That is, the file system provides a procedure call interface to the remote I/O processes, masking the fact that they are remote procedure calls. The application is unaware of the distributed nature of the system. In order to implement partitioned files, the file system automatically manages requests. It implements buffering so that many sequential operations can be satisfied with one request to the I/O process.

The file system implements the first level of system security, as it does not allow an application to send a message to an I/O (or application) process unless the application has first identified itself with an Open message that contains an authenticated user name. If the server process denies access to the object, the file system will not permit further messages (except for Open messages) to be sent.

The file system manages timeout and retransmission of requests; it sends the message to the backup server process if the primary fails. In addition, if the client is a process pair, the file system manages checkpointing of the process state to the backup process.

NonStop SQL

The file system and the disk server cooperate to process Structured Query Language (SQL) database operations in an integrated and efficient manner [Tandem Data Base Group, 1988]. The file system manages the SQL processing in the client and performs all the high-level operations such as sort, join, and aggregation. The disk process understands simple constructs, such as table, field, and expression, and will do low-level SQL operations on the data. Thus, operations can be performed at whatever level promotes the best efficiency. For example, the SQL statement

```
UPDATE ACCOUNT SET BALANCE = BALANCE + :DEPOSIT-AMOUNT
  WHERE ACCOUNT_NUMBER = :ACCOUNT-ID;
```

can be processed with a single message to the disk process. There is no need for the application to fetch the information, update it, and send it back to the disk process. In another example, the statement

```
SELECT FIELDA, FIELDE FROM TABLEX WHERE FIELDA + FIELDB > FIELDC;
```

allows filtering to be performed at the disk process, minimizing the transfer of data to the application.

NonStop SQL is designed specifically to handle OLTP applications while achieving good performance. Because it is integrated with other system software, it can be used for OLTP applications in a geographically distributed network of systems. SQL tables can be partitioned across systems in a network. Also, applications can run at one network node while accessing and updating data at another node. Furthermore, the applications themselves can be distributed. With NonStop SQL, fault tolerance derives from the basic mechanisms of process pairs, mirrored disk, and geographically distributed systems, along with node autonomy and transaction support. All transactions, local and network, are protected for consistency and error recovery.

From a fault-tolerance perspective there are two novel things about NonStop SQL. First is the design goal of node autonomy. The system is designed so that if the client and server can communicate, then the client can access the data. This simple requirement implies that all the metadata describing the file must be replicated with the file. If the file is partitioned among many nodes of the network, the catalog information describing the file must be replicated at all those nodes.

The second requirement is that no administrative operations on the data are allowed to take the database off line. For example, taking an archive dump of the database must be done while the data are being accessed, reorganizing the data must be done on line, and so on. Many administrative tasks are not yet on line, but a major focus of current efforts is to make all administrative operations completely on line.

SQL allows data administrators to attach integrity constraints to data; these may take the form of *entity constraints* that limit the values a record may have or *referential integrity constraints* that constrain the relationships among records in different tables. Placing the constraints on the data is more reliable than depending on the application program to make such checks. Updates to the data that would violate these entity constraints are rejected.

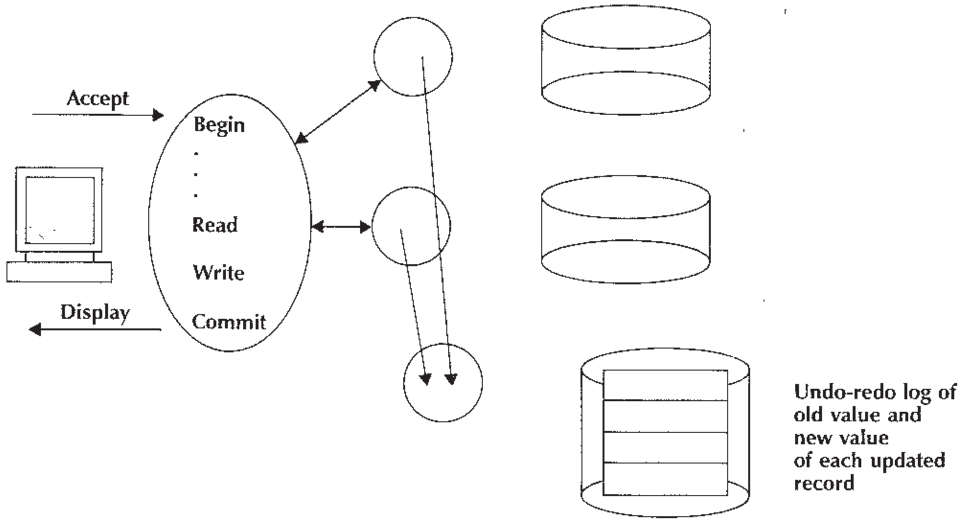
Transactions

The work involved in a computation can be packaged as an atomic unit by using the transaction monitoring facility (TMF) (Figures 8–62, 8–63). This facility allows an application to issue a Begin-Transaction request, make numerous database accesses and updates in multiple files, on multiple disks, and on multiple network nodes, and then issue an End-Transaction request [Borr, 1981]. The system guarantees that the work of the transaction will be ACID, defined as follows [Haerder and Reuter, 1983].

- *Atomic*: Either all of the database updates will be performed, or none of them will be; for example, if a transaction moves money from one bank account balance to another, the end result will never be more or less money on the books.
- *Consistent*: Each successful transaction preserves the consistency of the database.
- *Isolated*: Events within a transaction must be hidden from other transactions running concurrently; otherwise, a failing transaction could not be reset to its beginning.
- *Durable*: Once committed, the results of the transaction must survive any failure.

Should the application or Guardian (that is, the disk process or TMF) detect a problem that compromises the transaction, either one may issue Abort-Transaction, which will cause any

FIGURE 8–62
Structure of the transaction monitoring facility


Accept
Begin
Read
Write
Commit
Display
Undo-redo log of old value and new value of each updated record

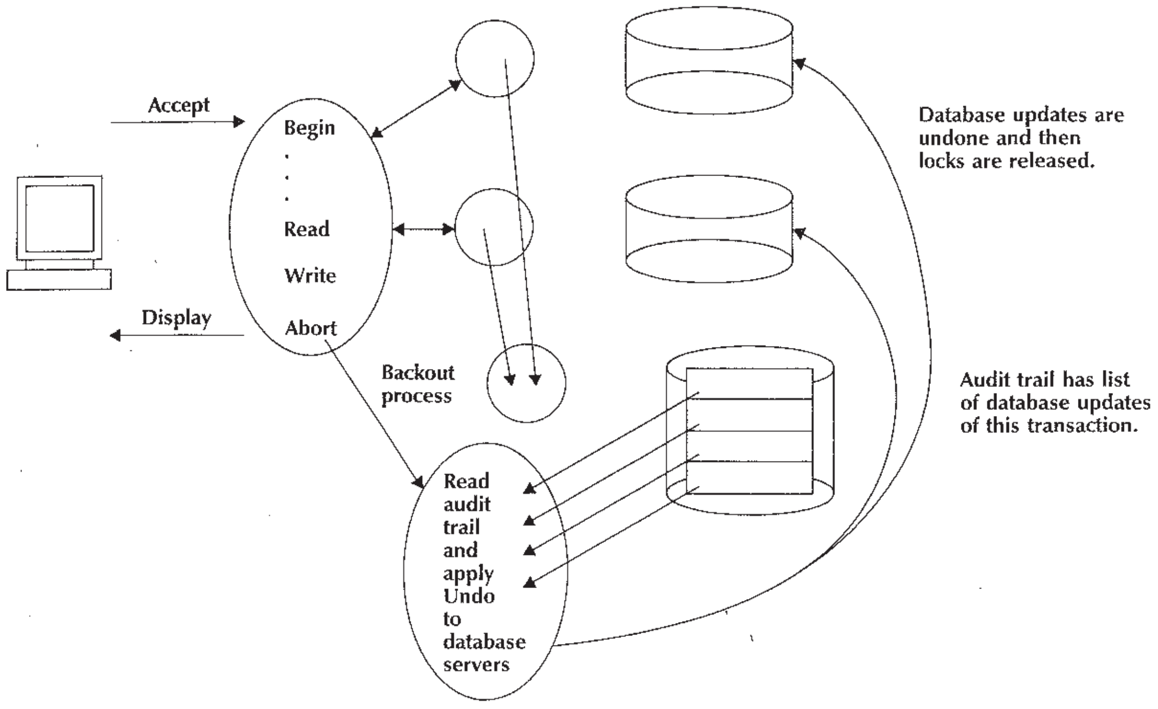

Accept
Display
Begin
Read
Write
Abort
Backout process
Read audit trail and apply Undo to database servers
Database updates are undone and then locks are released.
Audit trail has list of database updates of this transaction.

FIGURE 8–63 *TMF transaction backout structure*

database updates to be undone. The system can manage thousands of concurrent transactions, keeping them all isolated from one another. The application program need not concern itself with the locking protocol, as the required locking operations are well defined and performed automatically.

The work of a transaction can be distributed to multiple processes. As we shall see, it is normal to structure an application as client processes directing numerous server processes, each designed to perform some simple part of the transaction. Once the client has issued Begin-Transaction, all requests to servers are marked as belonging to the transaction; all database updates by the servers become part of the single transaction until the client issues the End-Transaction or Abort-Transaction request.

When a client issues Begin-Transaction, the system creates a unique *transaction identifier* and uses it to tag all messages and all database requests. If a tagged message is accepted by a server, all of its messages and database requests receive the same tag. Database locks acquired by the client or its servers are also tagged by the transaction identifier.

During a transaction, disk processes hold the updates in their caches. If there is insufficient cache, the disk process may update the database before the transaction completes, but first it must generate *undo* and *redo* records that allow the transaction to be either undone in case it aborts or redone in case it commits and a later failure occurs. When the client issues End-Transaction, each disk process with updates for the transaction must generate undo and redo log records before it updates the disk.

The undo and redo records are written to specially designated transaction log files called the *audit trail.* Normally, these files are on disks that are separate from the database, and the undo and redo records must be written to the audit trail disk before the main database is updated. Thus, even a total system failure will not lose transactions that are committed or allow the database to become inconsistent. Even if a mirrored disk pair is destroyed, the database and all transactions can be recovered from an archival copy of the disk and the transaction log. Any process participating in the transaction can unilaterally abort it. The system implements two-phase locking and uses a nonblocking, grouped, presumed-abort, two-phase commit protocol.

It might appear that support of transactions adds considerable overhead to the basic operations of accessing and updating the database. Surprisingly, TMF *improves performance,* while enhancing fault tolerance and data integrity. As described previously, updates to a database should be written to disk immediately in order to prevent their loss in case of a failure. This increases disk traffic and lengthens transaction response time.

When database operations are protected by TMF, a description of all updates is written to the audit trail; it becomes unnecessary for the disk process to write the updates to the database, except when it is convenient to do so. As soon as the audit trail records are reliably stored on disks, the application can be notified that the updates have been permanently recorded and will survive any failure. Writing the updates to the audit trail is considerably more efficient than writing the updates to the database because many updates from a single transaction (and, in a busy system, from multiple concurrent transactions) are blocked together and written in a single I/O operation. Further, the audit trail is written sequentially and writing is performed with a minimum of seeks, while database updates are random-access and imply numerous seeks. Finally, the disk process performs less checkpointing to its backup because uncommitted updates do not need to be protected against processor failures.

The result of these effects is that the logging and recovery of TMF is a net savings over the less functional store-thru-disk cache. TMF converts random main memory database access to sequential accesses, dramatically reducing the density of I/O transfers. Benchmarks have demonstrated that I/O density can be reduced by a factor of two or three when TMF is used [Enright, 1985].

While the Tandem system provides high availability through single-fault tolerance, TMF provides multiple-fault tolerance for the critical element of transaction processing systems: the database. Although multiple faults are exceedingly rare, the consequent cost of database loss is very high.

Before the introduction of TMF, application programmers relied on process pairs and forward error recovery to provide fault tolerance. Whenever an error was detected, the backup process resumed the computation from the last checkpoint. Process pairs were difficult to design and implement and reduced the productivity of application programmers. Applications implemented with TMF are much simpler to program and achieve the equivalent level of fault tolerance. Because transactions imply an automatic locking protocol, it is much easier to maintain a consistent database.

Process pairs are still an important concept and are used for system and I/O processes, as well as for specialized utility processes such as the print spooler. They are the fundamental basis on which TMF and other system software are built so that the customer can write fault-tolerant applications without regard for fault tolerance.

Transaction Processing Monitor

Applications are structured as client (requestor) and server processes. The clients are responsible for presentation services and for managing the user interface. Such user interfaces range from a forms-oriented interface to an electronic mail or home banking system, to real-time interfaces where the user is an automated warehouse, gas pump, or telephone switch. The servers are programmed to perform specific functions, usually a set of related database accesses and updates. In the electronic mail example, one server looks up names while another is responsible for routing messages to other network nodes and to gateways. In the gas pump example, one server does user authentication while another does billing. Typically, applications are structured as hundreds of services. Breaking an application into requestors and servers promotes software modularity, allows on-line change and growth of applications, and exploits the multicomputer architecture. With the advent of intelligent terminals (workstations, automated teller machines, and other computers acting as clients and servers), the client is migrating to the workstation, and the client-server architecture is becoming the standard structure for all transaction processing applications.

The application designer specifies the programs and parameters for each client and server. The servers can be programmed in any language, but the clients have traditionally been programmed in a COBOL dialect called Screen COBOL. This interpretive language automatically manages transactions and process pairs. In case the client process fails for any reason, the backup process takes over, reprocesses the input message if necessary, and redelivers the output message. This gives exactly-once semantics to transaction processing. Screen COBOL relieves the application programmer from needing to understand how to write process pairs. It is the most common way that customers get message integrity.

The transaction processing monitor, Pathway (Figure 8–64), is responsible for managing the application's requestors and servers. It creates requestors and servers at system startup, maintains a configuration database that can be altered on line by operator commands, and load balances the system by creating and deleting server instances as the load changes and as processors come and go from the system.

Process Server Classes

To obtain software modularity, computations are broken into several processes. For example, a transaction arriving from a terminal passes through a line-handler process (for instance, X.25), a protocol (for example, SNA), a presentation services process to do screen handling, an application process that has the database logic, and several disk processes that manage disks, disk buffer pools, locks, and transaction audit trails. This method breaks the application into many small modules, which serve as units of service and of failure. If one unit fails, its computation switches to its backup process.

If a process performs a particular service (for example, acting as a name server or managing a particular database), then traffic against this server is likely to grow as the system grows. Gradually, the load on such a process will increase until it becomes a bottleneck. Such bottlenecks can be an impediment to linear growth in performance as processors are added. The concept of process *server class* is introduced to circumvent this bottleneck problem.

A server class is a collection of processes that all perform the same function, typically spread over several processors. Such a collection can be managed by Pathway. Requests are sent to the class rather than to individual members of the class. As the load increases, members are added to the class. If a member fails or if one of the processors fails, the server class migrates into the remaining processors. As the load decreases, the server class shrinks. Hence, process server classes are a mechanism for fault tolerance and for load balancing in a distributed system [Tandem, 1985]. The application designer specifies the program, parameters, minimum size, maximum size, and distribution of the server class. Pathway reads this configuration database at

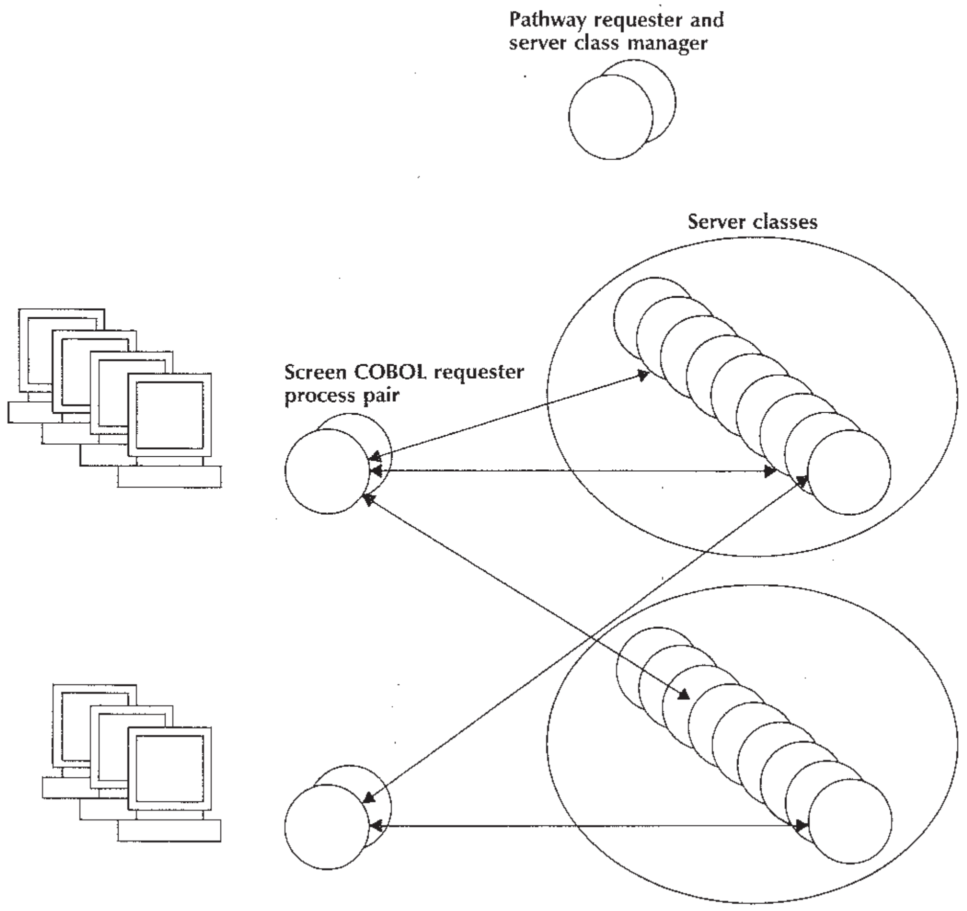

Pathway requester and server class manager
Server classes
Screen COBOL requester process pair

FIGURE 8–64
The structure of the Pathway transaction processing monitor

system startup and manages the server class, growing it as the load increases and shrinking it as the load decreases.

Networking

The process- and message-based structure of Guardian naturally generalizes to a network operating system. A proprietary network, called Expand [Tandem, 1987], enlarges the original 16-processor design to a 4080-processor (255-node) network. Expand uses a packet-switched, hop-by-hop routing scheme to move messages among nodes; in essence, it connects all of the IPBs of remote systems. Expand is now widely used as a backbone for corporate networks or as an intelligent network, acting as a gateway among other networks. The fault tolerance and modularity of the architecture make it a natural choice for these applications. Increasingly, the system software supports standards such as SNA, OSI, MAP, SWIFT, TCP/IP, Named Pipes, and so forth. These protocols run on top of the message system and appear to extend it.

The fault tolerance provided by the system extends to the network. Networking softwa allows a session to continue even if a communication link breaks. For example, SNAX, Tandem implementation of IBM's System Network Architecture, provides continuous operation by tran parently checkpointing at key points the internal information needed to sustain operation. Th enables SNAX to maintain all active sessions in the event that a single processor or line fail Similar provisions exist in the open system interconnection software.

Fault tolerance also underlies the distributed systems management products for global managing NonStop systems and Expand networks. For example, with the event manageme subsystem, an event recorder process executing as a NonStop process pair provides for gracef recovery from single-process failures. That is, if the primary event recorder process fails or stopped, the backup process continues recording in the appropriate event logs.

Disaster Protection

In conventional disaster recovery operations, when a disaster happens, people at a standby sit retrieve the database tapes from archival storage, transfer them to the standby site, establish compatible operating system environment, restore the data from tape to disk, switch commun cation lines to the backup site, and restart application programs. This is a complex, labor-intensive and error-prone process that commonly takes from 12 to 48 hours. The issues surrounding disaste recovery change dramatically when one moves from a traditional batch environment to the worl of on-line transaction processing. OLTP customers require recovery within a matter of seconds o minutes with little or no lost transactions or data. Symmetric network and application design based on the remote duplicate database facility (RDF) software give this kind of disaster protection

Operations personnel can use RDF to get applications back on line within five minutes afte a disaster. RDF stores the database at two systems, typically at two distinct geographic sites. Fo each data item there is a primary copy and a backup copy of record. All updates are made to th primary copy, and the TMF log records for the primary are sent to the site that holds the backu copy of the record, where they are applied to the backup copy of the data. The customer ca select one of three options for the sending of these log records:

- **2-Safe:** No lost transactions at takeover. In this case, the transaction is recorded at bot sites prior to committing the transaction to the client. It implies slightly longer response time because of the added network delay.
- **1-Safe:** The last few transactions may be lost at takeover because they were not recorde at the backup site. 1-safe has better response time and may be appropriate if each transaction i of little value or is easily reconstructed at a later time.
- **Electronic Vaulting:** The log of the system is simply transmitted to a remote site and stored there. It is not applied to the remote database until there is an actual disaster. This is similar to the standby site scheme, but avoids the movement of data to the standby site when the disaster happens.

RDF makes it possible to maintain a current, on-line copy of the database at a secondary node, as illustrated in Figure 8–65. The secondary database can be located nearby or across the nation. The use of RDF is completely transparent to the application programmer. Any TMF application can be converted to an RDF application without change. Only the database configuration and operations procedures change [Lyon, 1990].

To support its backup capabilities, RDF monitors and extracts information from TMF audit files and sends this information over the Expand network to a corresponding RDF process on the

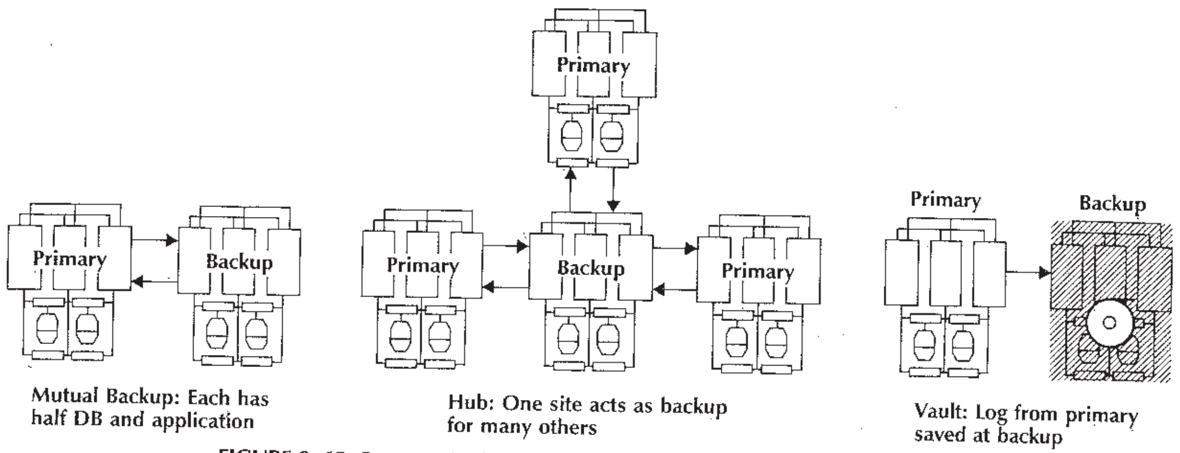

Primary
Backup
Mutual Backup: Each has half DB and application
Primary
Backup
Primary
Primary
Hub: One site acts as backup for many others
Primary
Backup
Vault: Log from primary saved at backup

FIGURE 8–65 *Remote duplicate database facility*

second node. This extraction usually takes place within seconds of the audit's being created by TMF. The RDF process on the second node receives the audit transactions and stores them in disk buffers. Another RDF process on the second node then applies these transactions to the database, thus maintaining the duplicate database. The second copy of the database is usually current within seconds of the primary database. Updating activities on the second database can be temporarily suspended without compromising the integrity of the on-line backup process. The audit transactions accumulate in disk buffers at the secondary site until updating is resumed. Then all accumulated updates are automatically applied in the correct sequence.

RDF has some additional fault-tolerance benefits. If one of the sites needs to be taken off line for a software upgrade, hardware upgrade, facilities move, or just a fire drill to test the disaster recovery facility, the load can be switched to the other node without interrupting service.

Operating System Summary

The innovative aspects of the Guardian operating system do not entail new concepts; instead, they are a synthesis and integration of preexisting ideas. Of particular importance are the low-level abstractions: processes and messages. These abstractions allow all processor boundaries to be hidden from both application programs and most system software. These initial abstractions are the key to the system's ability to tolerate failure. They also provide the configuration independence that is necessary in order for system and application software to run on systems of many sizes. Process pairs are a natural extension of the process concept to fault-tolerant execution. Transactions have been integrated with the operating system and appear as a natural part of the execution environment. Spooling the transaction log to a remote site is the basis of the disaster recovery facility. Extending the message-based system to a long-haul network makes it geographically distributed.

The operating system provides the application programmer with general approaches to process structuring, interprocess communication, and failure tolerance. Much has been documented about structuring programs by using multiple communicating processes, but few operating systems support such structures.

Finally, the design goals of the system have been demonstrated in practice. Systems with from 2 to 200 processors have been installed and are running on-line applications. Many of these systems are members of multinode networks. They are recovering from failures and failures are being repaired on line, with little or no impact on the system users.

Application Software

Application software provides a high-level interface for developing on-line transaction processing applications to run on the low-level process-message-network system described in the preceding sections. The basic principle is that the simpler the system, the less likely the user is to make mistakes [Gray and Anderson, 1985]. For data communications, high-level interfaces are provided to paint screens for presentation services. Furthermore, a high-level interface is provided to SNA to simplify the applications programming task. For database management, the relational data model is adopted, and the NonStop SQL software provides a variety of easily implemented functions. A relational query language integrated with a report writer allows quick development of ad hoc reports.

System programs are written in the transaction application language (TAL), which is a high-level, block-structured language similar to ALGOL or Pascal; TAL provides access to machine instructions where necessary. Most commercial applications are written in COBOL85 or developed through application generators. In addition, the system supports FORTRAN, Pascal, C, BASIC, MUMPS, and other specialized languages. A binder allows programmers to combine modules from different languages into a single application, and a symbolic debugger allows them to debug in the source programming language. The goal, however, is to reduce such low-level programming by moving to application-specific, fourth-generation languages.

A menu-oriented application generation system, Pathmaker, guides developers through the process of developing and maintaining applications. Whenever possible, it generates the application code for clients and servers based on the contents of an integrated system dictionary. The application generator builds most clients from the menu-oriented interface, although the user can tailor the client by adding COBOL statements. The template for the servers is also automatically generated, but customers must add the semantics of the application, generally using COBOL. Servers access the relational database either through COBOL record-at-a-time verbs or through set-oriented relational operators. Using automatically generated clients and the transaction mechanism, customers can build fault-tolerant distributed applications with no special programming required. Pathmaker provides maximum protection against failures through its reliance on TMF.

Ongoing investigations support the hypothesis that, as programmers migrate from language to language, human error rates remain nearly constant. That is, a programmer will produce about the same number of errors for every 1000 lines of code, regardless of the language being used. Thus, the higher the level of language a programmer uses, the smaller the number of errors in the object code. On this basis, Tandem urges its customers to use high-level tools like the Pathmaker application generator and other products that incorporate fourth-generation language concepts for OLTP program development. These tools include SQL Forms from Oracle, Applications By Forms from Ingres, and Focus from Information Builders. These tools greatly simplify the programmer's work.

New application software is under development that takes advantage of the growing number of personal computers and other workstations in the business world. This influx of desktop computers has dramatically influenced the way that business people do their computing. As these machines become faster and provide more memory and disk storage, businesses are expected to want system software for them that parallels software running on mainframes and minicomputers. This software, in turn, will generate a growing number of applications.

As stressed earlier in this section, customers demand good price/performance from fault-tolerant systems. Each Cyclone processor can process about 25 standard transactions per second. Benchmarks have demonstrated that 32 processors have 32 times the transaction throughput of one processor; that is, throughput grows linearly with the number of processors, and the price per transaction declines slightly [Tandem Performance Group, 1988]. Tandem believes a 50-processor Cyclone system is capable of 1000 transactions per second. The price per transaction for a small system compares favorably with other full-function systems. This price per transaction demonstrates that single-fault tolerance need not be an expensive proposition.

OPERATIONS

Errors that originate with computer operators are a major source of faults. Operators are often asked to make difficult decisions based on insufficient data or training. The system attempts to minimize operator actions and, where required, directs the operator to perform tasks and then checks the operator's actions for correctness [Gray, 1990]. Nevertheless, the operator is in charge and dictates what orders the computer must follow. This relationship poses a dilemma to the system designer: how to limit the actions of the operator. First, all routine operations are handled by the system. For example, the system automatically reconfigures itself in the case of a single fault. The operator is left with only exceptional situations. Single-fault tolerance reduces the urgency of dealing with failures of single components. The operator can be more leisurely in dealing with most single failures.

Increasingly, operators are given a simple and uniform high-level model of the system's behavior that reflects physical real-world entities, such as disks, tapes, lines, terminals, applications, and so on, rather than control blocks and other abstractions. The interface is organized in terms of actions and exception reports. The operator is prompted through diagnostic steps to localize and repair a failed component.

Maintenance problems, discussed earlier in this chapter, are very similar to operations. Ideally, there would be no maintenance. Single-fault tolerance allows hardware repair to be done on a scheduled basis rather than as soon as possible, since the system continues to operate even if a module fails. This approach reduces the cost and stress of conventional maintenance. The areas of single-fault-tolerant operations and single-fault-tolerant maintenance are major topics of research at Tandem.

SUMMARY AND CONCLUSIONS

Single-fault tolerance is a good engineering tradeoff for commercial systems [Horst and Gray, 1989]. For example, single disks are rated at an MTBF of five years. Duplexed disks, which record data on two mirrored disks connected through dual controllers to dual processors, raise the MTBF to 5000 years (theoretical) and 1500 years (measured). Triplexed disks would have a theoretical MTBF of over one million years, but because operator and software errors dominate, the measured MTBF would probably be similar to that of duplexed disks.

Single-fault tolerance through the use of fail-fast modules and reconfiguration must be applied to both software and hardware. Processes and messages are the key to structuring software into modules with good fault isolation. A side benefit of this design is that it can utilize multiple processors and lends itself to a distributed system design. Modular growth of software and hardware is a side effect of fault tolerance. If the system can tolerate repair and reintegration of modules, then it can tolerate the addition of brand new modules. In addition, systems must tolerate operations and environmental faults.

Fault tolerance can also be applied to open standard-based systems. These systems provide the benefit of application transparent fault tolerance at the cost of additional hardware resources and a decreased resilience to software faults.

REFERENCES

Bartlett, 1978, 1981; Bitton, 1989; Bitton and Gray, 1988; Borr, 1981, 1984; Brinch Hansen, 1970; Burman, 1985; Carr, 1981, 1985; Chan and Horst, 1989; Dijkstra, 1968; Eddy, 1987; Englert, 1989; Enright, 1985; Garcia, 1988; Gray, 1979, 1985, 1990; Gray and Anderson, 1985; Haerder and Reuter, 1983; Homan, Malizia, and Reismer, 1988; Horst, 1989; Horst and Chou, 1985; Horst and Gray, 1989; Horst, Harris, and Jardine, 1990; Horst and Metz, 1984; Jewett, 1991; Jouppi and Wall, 1989.

Katzman, 1977; Lenoski, 1988; Lyon, 1990; Mourad and Andrews, 1985; Nellen, 1985, 1986; Sohi, Franklin, and Saluja, 1989; Tandem, 1985, 1987; Tandem Database Group, 1988; Tandem Performance Group, 1988; Tom, 1988; Troisi, 1985; Uren, 1986; White, 1987.

THE STRATUS CASE

The Stratus Architecture

STEVEN WEBBER*

The fault-tolerant computer industry uses many terms that have different meanings for different people. The following definitions are of particular importance in this report on Stratus architecture.

• *Hardware fault tolerance* is the technique of applying hardware alone to effect fault-tolerance. Commercially available hardware fault-tolerant solutions include voting systems and systems such as those developed by Stratus that utilize two levels of duplexing—one for checking and one for fault-tolerance. Successful hardware fault-tolerant solutions require no effort on the part of application programmers to effect fault-tolerance. Hardware fault-tolerant computers are programmed as if they were simple, nonfault-tolerant computers.

• *Software fault tolerance* is the technique of applying software programs to effect fault-tolerance. These techniques typically include checkpointing information between different computers within a computer network so that some other back-up computer can take over when the primary computer fails. The majority of commercially available fault-tolerant products rely on software fault tolerance.

• *Software fault recovery*, or the ability of a system to recover from software faults, refers to the techniques brought into play when a software failure is recognized. These techniques preserve the integrity of the execution environment. They are independent of hardware fault tolerance, providing an additional level of protection for a different class of failures.

• *Critical on-line computing* refers to systems that run a company's most important business operations and manage the delivery of their most important products and services to customers. The system directly contributes to a business's profitability, revenue growth, or competitive advantage; it is the direct interface between a business and its customers; it provides constantly changing data relied upon for decisions in real time by many users; it is responsible for individual transactions of enormous financial value or business importance.

* The author acknowledges Kippi Fagerlund, who edited the draft manuscript, and also Marc Sill and Greg Baryza, who provided some of the artwork. The evolution of Stratus products is the result of the efforts of many people, including Bob Reid, Ken Wolff, Kurt Baty, Ron Dynneson, Joe Samson, Gardner Hendrie, Mike Grady, Larry Johnson, Jerry Stern, Janice Lacy, Bob Freiburghouse, Richard Barnes, Paul Green, Otto Newman, Herb Robinson, Neil Swinton, Doug Steinfeld, Jim Murry, Jim Bush, John Bongiovanni, and Jim Filreis.

• *Data integrity* refers to the consistency of multiple database records and files. Data integrity is usually a concern only after a system interruption. Fault tolerance does not guarantee data integrity. Data integrity is typically achieved through transaction protection.

• *Data corruption* is the contamination or alteration of data without any indication that the data is no longer correct. The kind of fault tolerance provided by Stratus systems protects against data corruption in nearly all cases. Most other systems, even fault-tolerant ones, provide minimal means of protection against or detection of corrupted data, and often the data is irrevocably corrupted before the detection hardware can stop it.

• *Transaction protection (TP)* is the ability to perform a sequence of database and communications operations in an *atomic* way such that if any changes are made to a database or sequence of databases, all associated changes that are part of the transaction are instituted. TP is independent of fault tolerance. A complete TP system must include its own recovery capabilities so that the integrity of the database after a system failure is guaranteed. Some TP systems do not provide fault-tolerant features. When hardware failures occur, such systems go down, but when service is restored, the database is made consistent such that each transaction prior to the failure is applied or lost. Even hardware fault-tolerant TP systems must provide mechanisms in addition to the normal system fault tolerance to guarantee database integrity against nonhardware-related system crashes.

• *On-line transaction processing (OLTP)* refers to a TP system's performing sequences of transactions in real time as on-line users, machinery, or other computers wait. Typical OLTP applications include 24-hour banking networks, airline reservation systems, manufacturing support programs, and point-of-sale terminal networks.

System downtime is typically broken up into the following categories.

• *Hardware failures*—failures of hardware that cause the system to stop running: As a result, the application (solution for which the computer was purchased) is not available. Usually, hardware failures lead to some degree of system degradation or failure. The degradation often manifests itself as poorer performance (response time and/or throughput) and occasionally results in the unavailability of selected data. The degradation may also result in the inaccessibility of selected devices.

• *Operating system software crashes*—failure of the software resulting in a system crash: After a failure of this type, the system must somehow reinitialize all or part of itself. Often, the failure is data-dependent and will crash multiple systems or continue to crash a system until some manual corrective action is taken. In other cases, the failure is unreproducible and is brought out only by the juxtaposition of relatively rare events (often related to I/O or communications error situations that are difficult to test). If the failure is unreproducible, retrying or restarting the system usually works, although there may be no insight as to why the system crashed.

• *Operational downtime*—SYSGEN, missing or improper operator intervention: Operators usually have high-level privileges (access or permission not granted to most users) and can inadvertently cause considerable damage.

• *Application software problems*—failure of the application software leading to the inability of the end-users to do their work: The operating system may remain operational, but the application must reinitialize itself or be reinitialized by explicit operator intervention. The fact that the system remains up is of little consequence to the end-users.

• *Database maintenance and backup*—the maintenance and support of database systems often results in periods of time when the database is not available for normal use: Registering new users, adding new structures, and backing up a database often requires that the database be made unavailable. As with application system software problems, the availability of the system without the database is of limited value.

• *Environmental problems*—problems with electrical power, air conditioning, earthquakes, floods or other storms, smoke, contaminants, or sabotage.

• *Communications failures*—failure of communications lines used to interconnect systems.

• *Field service*—removal of parts of the system or the disabling of the entire system for such activities as repairing failed components, running preventive maintenance, and running diagnostics.

• *Software installation*—the unavailability of part of the system or an entire system while new software is installed: Software installations fall into two main groups, installation of basic operating system software (which usually requires a reboot) and installation of application software (which rarely requires a reboot but usually requires the application to be quiesced or stopped).

• *Hardware installation*—the installation of new hardware to expand or upgrade a configuration.

The design of a modern computer system must minimize or eliminate downtime in all of these areas.

The statistical breakdown of the occurrence of downtime as a result of the various factors listed previously is different in nearly all published reports. Two trends are clearly important, however. First, hardware (including that of nonfault-tolerant systems) is becoming more reliable, resulting in less downtime. Second, operational downtime (including operator errors and system downtime for performing such activities as installing new hardware or software and running preventive maintenance on some component of the system) seems to be on the increase.

Another important trend is that software tends to become more stable and reliable the longer it is in the field without significant functional improvement. Software stability is one of the most important factors leading to software reliability. Modern software practices (structured design techniques such as design reviews, documentation, and code; use of modern development tools; object oriented programming; more extensive testing) have led to software products that are initially significantly more reliable, but the highest levels of reliability of most software are achieved only after fixing problems as they are encountered.

Another important issue relating to the availability of systems is environmental quality. For example, it is becoming increasingly difficult to get high-quality, continuous electrical power. Today's computer systems must not only be more forgiving of power fluctuations but must also be able to survive (ride through) brief power outages.

;TRATUS ;OLUTIONS TO)OWNTIME

Stratus has addressed potential downtime problem areas as follows.

• **Hardware Failures:** The Stratus hardware fault-tolerant architecture isolates users from almost all hardware failures.

• **Operating System Software Crashes:** System crashes are minimized (but not completely eliminated) with the use of software fault recovery procedures.

• **Operational Downtime:** Operational downtime is almost completely eliminated due to the lack of need for a SYSGEN; that is, hardware and software configuration occurs automatically. An operator needs to do little to run the system.

TABLE 8–7 *Introduction of major Stratus hardware products*

Year	Product	Significant New Features
1981	FT200	2-CPU 68000-based (2 logical CPUs/board); up to 16 MB of memory; user and executive CPUs (not symmetric); 20-slot main chassis; powerfail recovery
1984	XA400	4-CPU 68010–based (4 logical CPUs/board); symmetric multiprocessing
1984	XA600	6-CPU 68010–based (6 logical CPUs/3 boards); symmetric multiprocessing; 8-KB cache/CPU; floating point assist in hardware; 40-slot main chassis
1987	XA2000 110-160	1- to 6-CPU 68020–based (1 logical CPU/board); up to 96 MB of memory; 64-KB cache/CPU; floating point coprocessor; dynamic processor upgrades; enhanced powerfail ride-through
1988	XA2000 50–70	4-CPU 68020–based (4 logical CPUs/board); generalized I/O controller; 10-slot chassis; fault-tolerant I/O communications bus
1989	XA2000 30	Single-CPU 68030–based (1 logical CPU/board) "midplane" eliminates need for many cables and simplifies service; 6-slot chassis; integrated peripheral package; increased customer serviceability
1990	XA2000 210-260	1- to 6-CPU 68030–based (1 logical CPU/board); up to 256 MB of memory; 128-KB intelligent cache/CPU; bus-watching for cache consistency

• **Application System Software Problems:** Stratus provides highly structured, powerful interfaces for transaction processing and forms management. These interfaces can simplify application development and lower the risk of the application's introducing errors.

• **Environmental Problems:** Using sophisticated powerfail recovery procedures, powerfail ride-through, and battery backup, Stratus systems avoid or minimize most power-related environmental problems. Stratus systems usually require no special air conditioning.

• **Field Service:** Nearly all Stratus hardware can be replaced while the system is fully operational, minimizing the impact of field service on system availability. Boards, disks, fans, power supplies, and line adapters (interfaces to peripherals of all types) can be replaced on line while the system is running. Most replacements can be installed by customers, although some assemblies require trained field service personnel. Self-diagnosing boards clearly indicate broken parts. Specific failures are reported automatically to the Stratus Customer Assistance Center (CAC) through the remote service network (RSN) (using autodialing modems). Field service is simple and reliable.

• **Software Installation:** Installing basic operating system software (excluding device drivers) requires a system reboot. Most other software (including most device drivers) can be installed while the system is fully operational. To install new application software, however, it is often necessary for the application itself to quiesce or reach a clean point.

• **Hardware Installation:** Table 8–7 lists the major hardware products Stratus has introduced since 1981. Most Stratus hardware can be replaced on line while the system runs at full capacity. Further, any configuration expansions (including adding disks, memory boards, communications lines, and additional processors anywhere within a Stratus computer network) can be performed while the system is fully operational. The ability to dynamically expand or change the hardware configuration of a system is becoming increasingly important due to the need for frequent changes in many critical on-line systems.

)ES OF FAULT .ERANCE

A fault-tolerant system must be designed to withstand failures of all types. This means that the architecture and design of system software must be fault resilient and that every hardware component of the system must be, in some way, redundant.

Software Issues

Stratus has taken several approaches to solving the problem of system software reliability. First, from the beginning the Stratus architecture has included a mechanism that gives the system software designers powerful analysis tools and capabilities. The Stratus system automatically records information about every system crash and forwards this information or makes it available to the CAC or operations system personnel through the RSN. Stratus learns of every customer system crash automatically, and therefore has more knowledge of software problems than other companies. The system continuously logs system and error activity. These logs are valuable for analyzing many system outages.

Second, Stratus uses software fault recovery techniques to enable the operating system to recover from software bugs within the system. These techniques are described in detail later.

Finally, Stratus has used modern software development techniques, including use of high-level languages, extensive design and code reviews, testing for both quality and performance, and significant customer involvement in new hardware and software products. The use of tools and compilers avoids many classical errors (for example, forgetting to recompile all programs when an include file changes or forgetting to initialize a variable).

Hardware Issues

Most fault-tolerant systems provide some method of recovery from the failure of a major hardware component, but basic components are often overlooked. Specific critical components include power, buses (often printed circuits), disks, printed circuit boards, and the clock.

Table 8–8 describes the redundancy techniques of Stratus system components. The following paragraphs describe how Stratus has addressed failures of these components.

Power. Power failures occur within the computer itself as well as outside of it. Uninterrupted power sources solve only the external problem. The failure of a power supply or the cables or buses that distribute the power to the active components will crash most systems. A complete solution requires that power from separate sources be fed through different power supplies to separate logic boards over different buses.

Techniques that use special chips to provide duplicate, self-checking circuitry do not satisfactorily address the problem of power failures. Such chips use a single power source. This solution to protect against internal chip logic errors is becoming less practical as a means for achieving fault tolerance as computer chips become more reliable. Most failures today occur above the chip level. Similarly, triply redundant voting systems do not solve the power problem unless each voting circuit and logic unit derives power from a separate source.

Buses. Architectures that use a bus must have a duplicate bus to protect against failure. A bus can fail because of a mechanical problem, such as a short in the bus interface logic. Systems that use some form of local area network must have duplicate media and interfacing logic. Furthermore, local area network systems must provide software to automatically manage failures in the

TABLE 8–8 *Stratus system components*

Component	Type/Width	Bandwidth/Speed	Redundancy Technique
Processor	68000	8 MHz	Self-checking; lockstep pairs
	68010	8 MHz	
	68020	16 MHz	
	68030	24 MHz	
Memory	8 MB	—	Self-checking logic for control; ECC for data; lockstep pairs
	16 MB	—	
	32 MB	—	
StrataBus	32 bits data	64 MB/sec	Duplicated; parity on groups of signals
StrataLink	—	1.4/2.8 MB/sec	Self-checking; duplicated
I/O controllers	Communications	600 Kb/sec	Self-checking; lockstep pairs
	IOP	4 MB/sec	Self-checking; lockstep pairs or duplicated
	Disk	1.2 MB/sec	Self-checking; duplicated
	Tape	600 KB/sec	Self-checking
Devices	Disks	—	Duplicated
	Terminals	—	None
	Tapes	—	—

media. Without such software, application programmers must take on what should be a system function.

Disks. All fault-tolerant computers provide some form of disk redundancy. Some systems provide the option of mirroring entire volumes; others allow mirroring of partial volumes. Since disks are far less volatile than main memory, many applications and users rightfully feel secure only after their data is safely stored on one or more disks.

Boards. Failure of a single component on a computer board usually has one of two effects: the entire board breaks in some way recognizable by the rest of the system, or worse yet, the error goes unnoticed. In most systems, either situation eventually leads to a crash of the module containing the board. A truly fault-tolerant computer must protect against **any** error of this type.

Clocks. Although very reliable, clock circuits which generate and distribute clocking signals to the various components of the system must be considered when designing a fault-tolerant system.

SYSTEM ARCHITECTURE OVERVIEW

A Stratus computer *system* consists of up to 32 *modules* connected through a Stratus intermodule bus (*StrataLink*). The StrataLink, a redundant, coaxial, proprietary interconnect mechanism, is used for system expansion. Using StrataLink, the Stratus operating system provides a global system view for all devices in the system. It also provides "message passing" between cooperating processes. The StrataLink restricts modules to a geographic proximity of a few miles, but typically the modules are linked into a system in a single building or a few adjacent buildings. The StrataLink consists of two independent coaxial "links." Each link runs at 1.4 megabytes per second, providing an intermodule throughput capability of 2.8 megabytes per second. The system software continuously uses both links, thereby recognizing immediately if one of the links fails. If a link does fail, the throughput drops to 1.4 megabytes per second.

A Stratus *network* consists of several systems connected using an X.25 packet switched network or the StrataLink hardware. Figure 8–66 illustrates an X.25 packet switched network of four systems (4 groups of 14 modules total connected with StrataLinks).

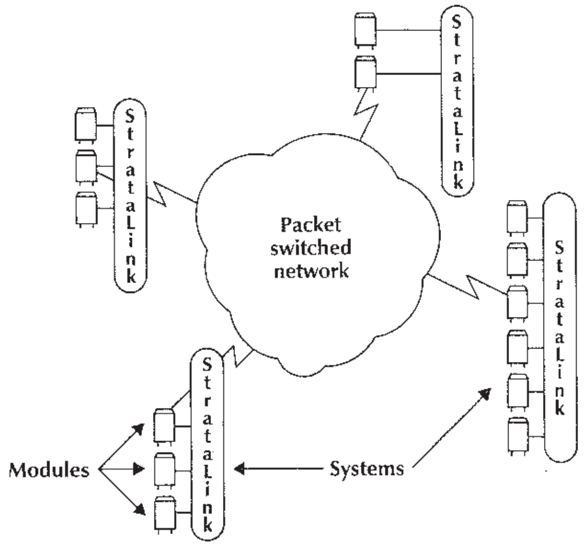
Each module within a system is a complete computer; Figure 8–67 illustrates a typical Stratus module. Each module is bus-oriented, containing a Stratus backplane or midplane that implements a proprietary fault-tolerant module bus called a *StrataBus*. A module also contains one or more processor board pairs, one or more memory board pairs, and pairs of I/O controllers connected to various peripheral devices, including disks, tapes, and communications interfaces. A module contains its own redundant power supplies, battery backup subsystem, and various card cages for I/O line adapters and I/O controlling logic.

The Basic Module Bus

The major boards of a Stratus module interface with the StrataBus. The StrataBus has 32 logical slots and is implemented in various Stratus modules with 6, 10, 20, or 40 physical slots. (The 40-slot chassis supports logical board stacks that contain multiple physical boards.) The StrataBus uses an arbitration scheme that, in its simplest form, provides bus priority as a function of bus slot number. Even-numbered slots within the bus derive power from a subsystem that is totally independent of the power subsystem used by the odd-numbered slots of the bus. This partitioning of slots into disjoint, isolated sets enables a module to survive the complete failure of either of its power subsystems.

The power for the boards derived from these power supplies consists of unregulated 24 V_{DC}. Each board regulates this power to the necessary levels. By placing the power regulation function on each board, it is possible to remove and insert boards from the StrataBus while the

FIGURE 8–66
A typical network



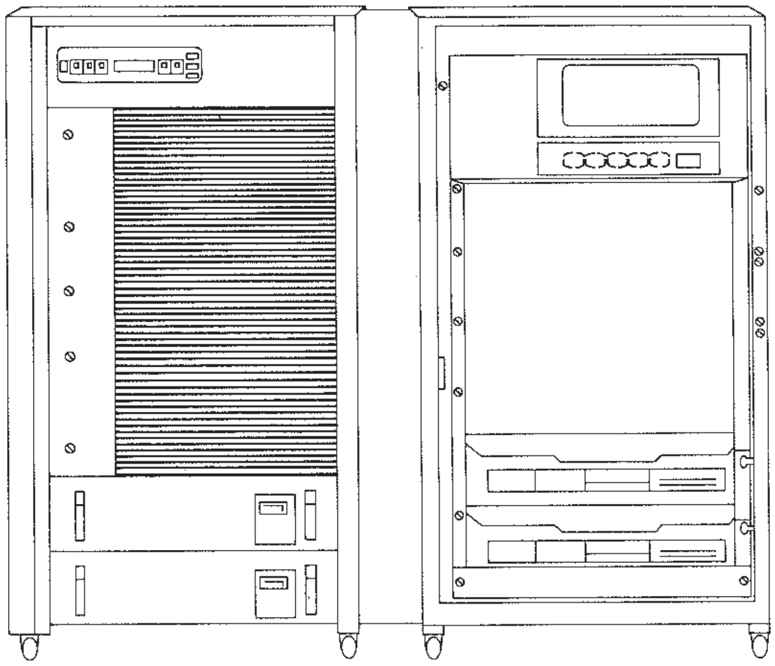


FIGURE 8–67
A Stratus 40-slot module

bus is fully powered. No components of the system need to be disabled to remove or insert a board.

The StrataBus is a synchronous bus. The system clocking function drives all circuitry in the system and is distributed using a single bus signal. This clock signal generates a standard bus tick of 125 nanoseconds (8 MHz). The bus protocol provides for extensive overlapping/pipelining, allowing a new bus request of up to 64 bits of data on every 125-nanosecond tick. This yields a maximum bus bandwidth of 64 megabytes per second. (The system clocking function generates higher frequencies for boards running at rates faster than 8 MHz.)

Nearly every signal on the bus is duplicated. The bus is actually viewed as two independent buses, bus A and bus B, each with its own power, ground, data, address, and control lines. Parity protects the lines of each bus.

Each major system board simultaneously interfaces with both buses. A board drives the same data on both buses and reads data from both buses if they are both enabled. (The hardware automatically instructs all boards to ignore a suspect bus.)

A power supply for each bus powers pullup resistors to effect an *open collector* technology. Unless driven to a 1 state by at least one board, each bus signal represents a logical 0.

Bus Monitoring by Memory Boards. The various parity signals on the StrataBus detect bus failures or bus-related failures of boards that interface with the bus. Controller logic performs the detection within the memory boards. If multiple pairs of memory boards exist within a module, all such boards monitor the bus for problems. If a memory board detects a problem on either bus, it declares that bus broken and instructs all boards to stop using the bus (by asserting signals on bus lines). This bus monitoring detects such failures as an open bus line, a shorted bus line, or the failure of a bus driver on one of the boards plugged into the bus. Such bus driver failures cannot be detected by the on-board checking logic, as the drivers are enabled and disabled by the comparator logic and are thus logically beyond it (see Figure 8–69). The bus drivers are the only logic on a major board not covered by the self-checking logic of the board.

If the memory subsystem detects a subsequent failure of the single working bus, that bus is declared broken and the originally broken bus is declared good. The buses alternate in this manner until system software tests the buses and places them both back in service. Alternating between failing buses provides a simple recovery strategy for transient bus errors. If the original failed bus has a hard failure, implying that both buses have simultaneously failed, the module is inoperable.

Power Subsystem

Each power supply has an associated battery backup system. The battery system works in two modes: It either powers all boards within the module, or it powers only the memory boards. The power supplies monitor the AC power and interrupt the system when they detect a power failure. The system software reacts to a power failure indication by saving in main memory all information needed by boards in the system. In 90–95 percent of all power failures, power returns within a few seconds. If the AC power-sensing hardware detects a failure of this type, the system simply continues with what it was doing before the power failure was first detected.

The battery systems have enough capacity to power the entire module (excluding disk and tape drives) for up to six seconds during a power outage. After six seconds, if the AC power has not returned, the system quiesces all boards and instructs the hardware to supply power only to the memory boards. In this extended battery backup mode, the system is not operational, but its complete state is preserved. If power is restored within an hour or two, depending on the amount of memory that must be backed up by battery, the system software restores the system to its state at the time of the power interruption. All I/O that did not complete is restarted.

During the first few seconds of a power outage (while the system is still operating), the disks are not powered and begin to cycle down. This typically leads to normal, recoverable disk errors that are retried when the power is fully restored. No data can be damaged.

With the large memory sizes supported by Stratus modules, there is not enough time to save all of memory to disk before it becomes necessary to switch to the low power usage mode, which supports only memory. The battery system has insufficient power to support all disks in an operational state. Additional batteries can be configured to provide recovery from arbitrarily long power outages.

System Boards

The boards that interface to the StrataBus have several common features. First, they all operate synchronously at a simple multiple of the common system frequency. First-generation boards run at 8 MHz, but newer boards operate at up to 24 MHz.

Second, all boards are self-checking and auto-isolating. The boards check themselves for component errors on every clock tick and will not place data on the bus if a board finds itself to be in error. This self-checking is typically performed by duplicating the logic on the board and running both sets of logic independently but synchronously. The outputs of these independent logic networks then run through onboard comparator circuits that enable the bus drivers. The self-checking logic automatically causes a board to break. A failed board is said to be *broken* or *red-lighted.* (When a board breaks, a red LED on the front of the board lights to identify it.) A broken board never drives data onto the buses.

Third, each board must provide its own power regulation to convert the unregulated 24 V_{DC} available from the bus to usable TTL levels.

Fourth, each board is self-identifying, providing system software with coded information describing the board's type, the revision level of the board, the revision level of the PROM software on the board (if any), and a limited amount of board repair history. This information is critical to the software that puts reinserted boards on line. If the boards are incompatible, the system does not accept the new board.

Fifth, all boards that interface to the StrataBus must obey a set of common interface conventions used by the Stratus maintenance and diagnostic software for testing, initializing, and enabling boards in the system. The common interface allows the operating system software to monitor and diagnose boards plugged into the bus through a common mapped I/O space—a special range of virtual memory addresses—which is interpreted similarly for all boards. (Each slot within the backplane has a set of addresses associated with it that control the board in that slot. These addresses are referenced to enable, disable, test, remove, and restore the board.)

One specific requirement of this self-checking feature is that both halves of a board must behave totally deterministically. Any logic must progress from state to state with each clock tick totally deterministically. In particular, *don't care states* (bits that can apparently harmlessly assume either a 0 or a 1) are not allowed, since they may yield conflicting values in the comparator logic on the boards.

Figure 8–68 depicts the general layout of a Stratus self-checking board. The A connector connects to bus A, the B connector connects to bus B, and the C connector connects to external

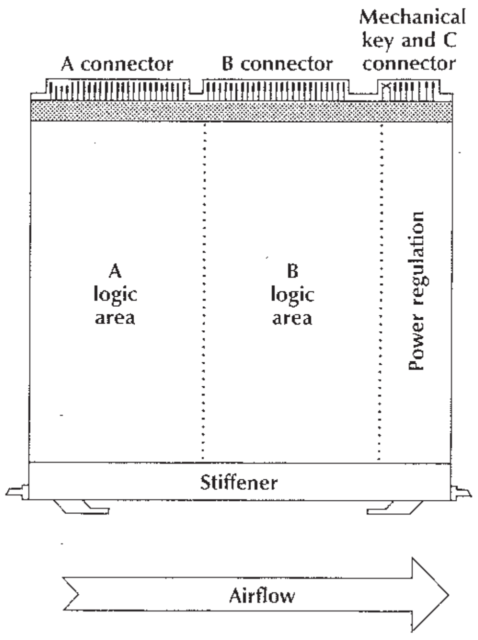


FIGURE 8–68
A self-checking board

logic, such as an I/O bus, the maintenance panel (in the case of CPU boards), or to other boards of the same type (as is the case for memory boards).

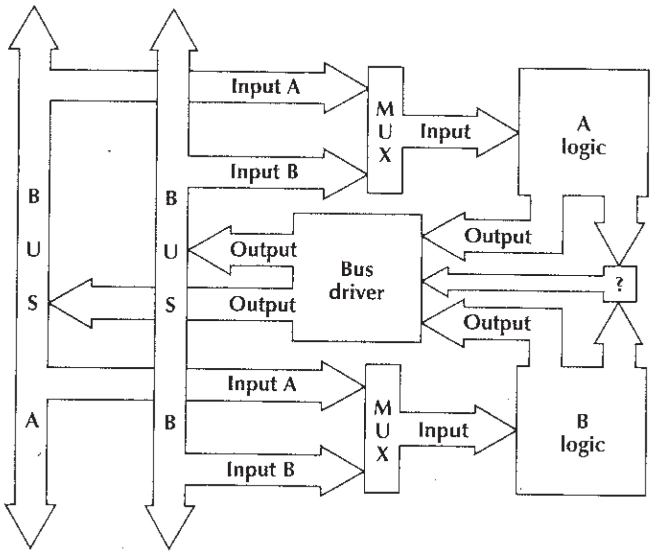
Figure 8–69 is a block diagram of the typical checking logic for a self-checking board. The comparator logic (represented by the question mark in Figure 8–69) enables and disables the bus drivers. If a compare mismatch occurs, the bus drivers are immediately disabled, and no data is allowed out on the buses.

Boards within a Stratus system nearly always occur in pairs. Such boards are referred to as *partners*. Major boards operate in two ways: either in synchronous lockstep with a partner board or only logically paired with a partner board. When operating in synchronous lockstep, proprietary Stratus hardware and software synchronize both self-checking boards of the pair; thus, both boards (including a total of four sets of logic) do exactly the same thing on any bus cycle. This requires that the boards behave completely deterministically with respect to any conditions that can arise. In essence, the same requirements that apply to self-checking boards (mainly total determinism of the logic) apply to boards in synchronous lockstep.

Two boards running in synchronous lockstep read the same values from the StrataBus simultaneously. Output signals are logically ORed on the StrataBus. For example, two client partner boards ask for a bus cycle (arbitration for the bus) at exactly the same time (on the same cycle) by asserting their intention to use the bus. If the arbitration network grants the bus to the pair of boards, they both place the address for a read or write on the bus at the next clock tick. Two ticks later, the client boards or the responding boards (usually memory controllers) place the data on the bus. The data is taken off the bus one tick later.

Boards that interface with such devices as disks, tapes, and the StrataLink operate in a logically paired state. Such boards are not synchronized in lockstep with their partners; rather, they rely on operating system software for an equivalent function. (The two halves of the board must be synchronized.) Disk mirroring with dedicated disk controllers uses this method. Software ensures that mirrored disks contain duplicate, consistent copies of data. This software is com-

FIGURE 8–69
The logic of a self-checking board



pletely invisible and inaccessible to application code (and most operating system code) and therefore need not concern application and operating system developers. The placement of partner boards in even-odd slots of the StrataBus chassis is not required by the architecture, but it is required to protect against power supply failures.

Off-Board I/O Interface Buses

The I/O controller boards usually connect to some form of I/O bus. The type of bus is a function of the type of controller. The dedicated disk controllers have several interface options such as Stratus proprietary and SMD. The tape and StrataLink controllers each interface to a single device and do not have an associated bus. The programmable StrataBus interface (PSI) controller has a Stratus proprietary bus. The communications controllers interface to Stratus proprietary buses, some of which are fault tolerant. The I/O processor (IOP) boards interface to a Stratus proprietary fault-tolerant bus.

Any controllers that run in lockstep with a partner and interface to nonfault-tolerant buses must have special logic to interface to the bus. This guarantees that all four sets of logic (two sets on each board of the partnered pair) see the same data. This scheme is typically implemented using (1) latches on the logic interfacing to the bus, (2) conservative timing assumptions so that clocking signals from the bus lead to all four sets of logic seeing the same data, and more recently, (3) reflexive checking logic on the controller boards. The reflexive checking logic trades special signals between the partnered boards at each clock tick to make sure that the boards are still synchronized. The signals traded are basic board cycle signals, the earliest indicators that the boards are out of synchronization. If the boards do go out, the hardware stops one of the boards automatically. This reflexive checking guards against failing devices connected to one of the nonfault-tolerant buses.

Line adapters, or I/O adapters (IOAs), plug into the various I/O buses and interface with devices or communications lines. These come in many types: full modem asynchronous, null modem asynchronous, synchronous, or high-speed synchronous. The IOAs usually contain microprocessors and significant amounts of memory for loading protocol-specific code to drive that particular device or communications line.

The IOP utilizes a sophisticated, self-checking, duplexed I/O bus, called the *P/Q bus,* to interface to the IOAs. This is a duplicated address/data multiplexed bus. The same bus signals are used for both address and data, and there are two separate sets of signals. The bus is limited to 20 feet in length and supports 4 megabytes per second transfer speeds. Both the P and the Q buses use parity for checking. Each bus has four function code lines, again protected by parity. The bus design is expandable to 8 megabytes per second.

The P/Q bus protects against any single-bit asynchronous glitch and any multibit failure on a single bus. The bus uses loop-back checking to ensure that a bus has what was placed on it, bus-to-bus comparison (P versus Q) to see if both buses have the same data, and continual checking for errors by every board on the bus. Each IOA has a pair of custom gate arrays to implement the special bus protocols and checking logic. The P/Q bus can accept white noise on any data line and continue to run. Figure 8–70 illustrates the logical configuration of I/O.

Major Board Types

Stratus supports three major board types: processors, memory boards, and I/O controllers. The processor and memory boards run in lockstep. Some I/O controllers run in lockstep, and some run logically paired.

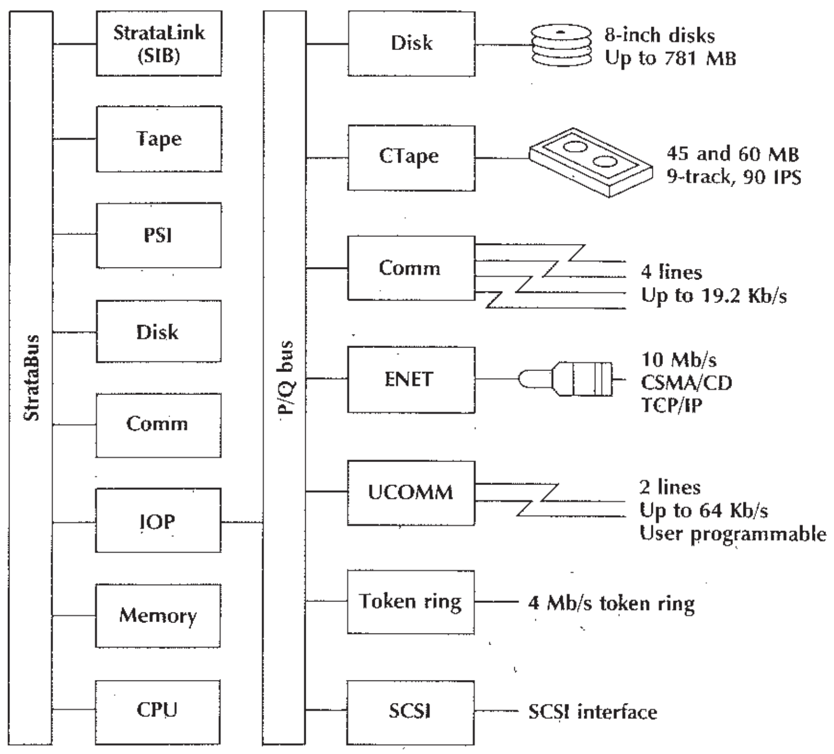


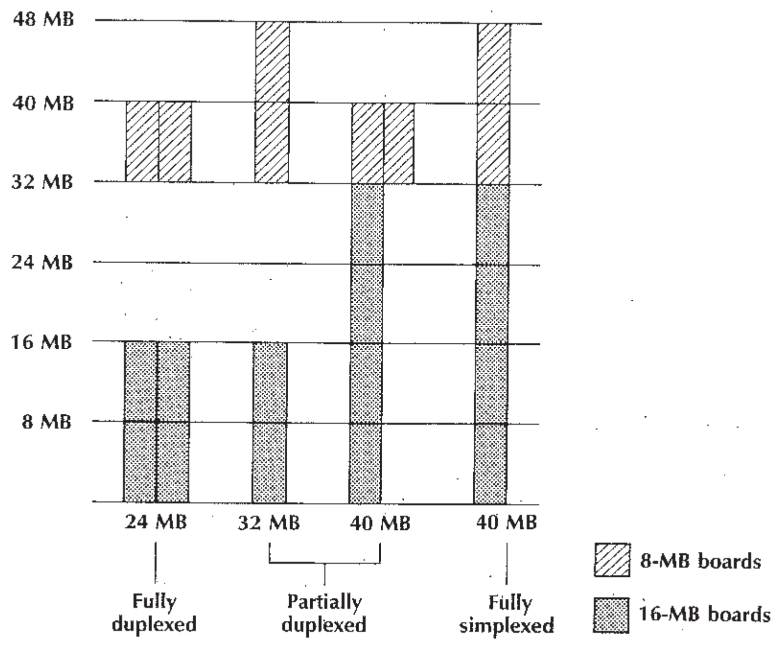
FIGURE 8–70 *Logical configuration of I/O components*

Processors. Stratus has several processor board products, all based on the same principles. The operating system software supports multiple processors executing out of a common shared memory system (often referred to as *symmetric multiprocessing*). This provides significant performance improvements over single-CPU configurations. For applications that require more CPUs than the maximum supported by a single module, Stratus resorts to the same architecture other computers use for expansion: loosely-coupled configurations linked together over some high-speed interconnect, using a combination of message passing and remote procedure calls.

Memories. Stratus system memory is contained on memory packages of one-, two-, or three-board stacks. Such stacks are called a board since they behave as a single unit and use one logical slot in the StrataBus chassis. A memory board contains controller logic that interfaces to the StrataBus and drives the memory chips. Memory currently comes in 4-, 8-, 16-, and 32-megabyte packages. All memory is protected by single-bit error-correcting, double-bit error-detecting (ECC) logic.

Memory can be configured in one of two ways: *simplexed* or *duplexed.* When configured in a simplexed manner, the memory is not fault tolerant, although it can survive single-bit errors. When configured in a duplexed manner, the memory is fault tolerant. The simplexed configuration is provided for customers who want to configure more memory at certain times while knowingly sacrificing fault tolerance. Memory can be switched between fully duplexed and simplexed and back to fully duplexed again while the system operates. Figure 8–71 illustrates various configu-

FIGURE 8–71 *Methods of duplexing and simplexing memory*



rations of the same set of four memory boards, two of which are 16 megabytes and two of which are 8 megabytes.

The ECC logic on the memory boards behaves differently, depending on whether or not the memory is duplexed. If the memory is duplexed, a board detecting an ECC error immediately declares itself temporarily broken, allowing the remaining partner board to continue operating. Operating system software tests the broken board, and if the error appears to be transient, the board is reduplexed with its partner. If the memory board is running simplexed when the ECC error occurs, the error is corrected, and the client board receives the correct data two cycles later. The board is not reported as broken in this case.

When a memory board is reduplexed with its partner, the additional memory controller is placed into a listening state; it does not drive data onto the bus. This state allows a special bus cycle to read from the good memory and immediately write back to both memory boards. A background system process sweeps through all memory using this cycle, allowing normal operations to continue while a memory board is added to the system. More recently manufactured memory boards can perform this update operation automatically (after diagnostic system software instructs them to do so).

I/O Controllers. As previously mentioned, I/O controller boards operate in one of two ways. Synchronously lockstepped boards include the various models of communications controllers, the IOP, and the PSI. If one of the boards of such a pair fails or is removed from the system physically, the other board continues to process what the two boards were doing in parallel.

Logically paired I/O controller boards perform different physical tasks simultaneously for peripherals such as disk and tape. The operating system software ensures that applications are presented with a fault-tolerant base on which to work. The most interesting board type handled in this way is the *disk controller.* (Stratus supports disks with the IOP, as well as with dedicated disk controllers. The basic algorithms that follow apply to either.)

Disks. Disks are grouped into logical volumes of up to 10 pairs of disks. The disks in a logical volume need not all be of the same type. Each disk in the logical volume is usually, but not necessarily, duplexed or mirrored with another disk. A disk and its corresponding partner disk are called a *duplexed pair.* Duplexed pairs must be of the same disk type.

Standard configurations assign the two disks of a duplexed pair to different disk controller boards. This configuration provides two completely independent hardware paths for accessing the two copies of data. The controlling logic, electrical power to drive the logic, bus interfaces, and cabling system are also duplicated. The operating system software uses several techniques to ensure that both disks of a duplexed pair contain exactly the same data and that if one disk fails, its replacement is brought up to date automatically.

CRC Checks for Disk Blocks. Each block of data written to disk is checksummed, and the checksum is written to the disk with the data. This allows the hardware to detect problems related to cables and connectors and failures that were not detected by the disk hardware. The checksums are examined by logic in the disk controller, which is itself self-checking, providing an additional guarantee of data integrity.

Bad Block Remapping. Stratus software keeps a list of blocks that are unusable—either because the manufacturer declared them unusable or because Stratus diagnostic software determined they were unusable during the manufacturing process or normal operation. In either case, the operating system allocates another block on the disk from a pool (on each disk) reserved for this purpose and uses this alternate block whenever references are made to the failed block. The software uses the alternate disk block address for all accesses to the disk. The remapping is done whenever a read or write fails and retries are unsuccessful. The knowledge of this remapping is isolated to limited software in the disk system.

When a write failure occurs, the data is simply kept in memory and written to the newly allocated block. When a read fails, the data is read from the other disk and then written back to an alternate block on the disk that had the read failure.

SCSI disk manufacturers now build this type of facility (or a variant of it) into the drives that Stratus purchases and incorporates into its systems.

Reads from a Duplexed Pair. The operating system can read from either of two disks containing the same data. Depending on load, activity, and reference patterns, appropriate disk selection algorithms can lead to an effective disk performance improvement. One algorithm that Stratus uses is simple: If both disks are idle (no real or queued activity to the disk), the operating system selects the disk positioned closest to the cylinder on which the data resides. If one disk is busy, the operating system selects the other disk. If both disks are busy, a disk is selected at random and the operating system uses a finer-grained disk sorting algorithm.

Fast and Normal Disk Recovery. Stratus hardware and software can diagnose a particular failure when it occurs. Two cases are significant: a controller (which can be controlling several disk drives) fails, or a single drive fails.

In the case of a controller failure, the system continues to operate by reading from and writing to the disks connected to the controller partner that is interfacing with the mirrored disks. When the failed controller is replaced (on line, while the system is fully operational), the operating system automatically brings the disks connected to that controller in synchronization with the disks that were in use while the controller was out of service. Stratus calls this updating operation *disk recovery.* The operating system performs it automatically upon recognition of the replacement controller in the module. Basically, disk recovery consists of copying data from the good (consistent, complete) disk to the bad (inconsistent, incomplete) disk.

Stratus provides two forms of disk recovery. The first, *fast disk recovery,* is used when a controller, rather than a drive, fails. In this case, most of the data that exist on the drives connected to that controller are valid. As soon as the controller fails, the operating system software begins tracking all writes to the disks connected to that controller. When the controller is later replaced, the system need only update the blocks that were modified while the controller was missing or broken. Any data that may have been modified while the controller was broken are copied from the good disk, thereby guaranteeing consistency of data on both disks.

The second form of disk recovery, *normal disk recovery,* is used when the entire disk must be updated from its partner. Normal disk recovery is necessary, for example, if the drive itself must be replaced. The system software copies all used blocks from the valid disk to the new disk while the system operates. During the recovery period, the software forces writes to both disks (while reading only from the good disk), guaranteeing that, when the recovery pass is complete, the disks are synchronized.

Disk Writes. Most operating system developers do not worry about failure scenarios to the degree that developers of fault-tolerant systems must. Two issues are of particular interest because they demonstrate Stratus's concern for guaranteeing data integrity.

The first relates to serial writes; the second relates to verified writes. The Stratus disk software must, as noted, manage the writing of each data block to two separate disks. Writing could be done in parallel (overlapping the I/O), since totally separate I/O paths are available. Instead, the operating system does not usually begin the second write of the two until it knows that the first write has completed without error. This strategy protects against an extended power failure while a parallel write is done to both disks. Such a power failure, in rare cases, could lead to blocks being written incorrectly as the write logic within the disk drives loses power. Incorrectly written blocks, in turn, could lead to the destruction of both copies of a piece of data.

Writes are also verified to guarantee that the data is reliably on disk. The second check, referred to as *disk verify,* is provided on most vendors' disk hardware.

Stratus provides the option of allowing selected disks to run without verifying writes. A parallel write option will be available in the future.

Synchronizing Boards

When a Stratus module is initialized, all boards that are running in lockstep must be synchronized. The concept is fairly simple: get the four sets of logic to do exactly the same thing. The actual implementation is a bit tricky. The system starts up with all intelligence isolated to one of the main CPUs of the system. Tie-breaking hardware effectively runs exactly one CPU at first. All other boards initialize themselves but remain off line. The running CPU surveys the hardware in the system, using the mapped I/O space that is part of the common interface to each board that interfaces with the StrataBus. The CPU brings the other boards within the system on line one by one.

Before a CPU can run, it must synchronize its two halves: All registers, caches, memory cells, and microprocessors must be provided with the same data. The CPU initializes these items before enabling the comparator logic that would otherwise force the board off-line as broken. Memories must be brought on line by the initializing CPU before any I/O (other than primitive character I/O to a terminal) can be accomplished.

Reflexive Checking

The term *reflexive checking* describes the interboard communication performed by certain pairs of boards to guarantee they are synchronized. If the boards get out of synchronization, one of them is forced off line automatically by the hardware so that the boards' signals, which are ORed on the bus, do not confuse other boards in the system. This logic typically exchanges selected control state indicators that change frequently (at least every tick) during operation of the boards.

Periodic Tests

The comparator logic on self-checking boards can fail. Such a failure has two manifestations: either a false failure can be reported, or a true failure can be missed. A false failure indication is harmless to the system, but even so, it is immediately detected and indicated—the board goes broken. Failure to diagnose a real problem is also harmless as long as the partner board is working. If a single failure occurs, one of the buses is receiving the correct data from both boards. The failing board can fail in one of two ways. First, it can place a 0 on the bus when it should place a 1, in which case the partner board overrides it by placing the correct data on the bus with an OR signal. Second, it can place a 1 on the bus when it should (and its partner does) place a 0 on the bus. In this case, the memory controllers detect the problem using the parity-checking logic and declare that bus broken.

The board can also be broken in such a way that no data at all are being placed on the bus; for example, the on-board power system could be broken. If this is the case, the bus monitoring software run by the maintenance process detects the problem. The board does not appear to be on line. Note that the state of the bus logic is such that a 0 is assumed if nothing is driving the bus (as would be the case if the board had no power).

RECOVERY SCENARIOS

With this background, it is easy to understand how Stratus achieves its fault tolerance while meeting so many of its corporate product goals. Once an entire system is initialized and all duplexed components are synchronized with their partners, the following failures are insignificant both to system availability and capacity.

- **Board Failure:** When a component or materials failure associated with a board occurs, the failing board automatically drops out while its partner continues to run. The partner does not take over, as would be the case with a backup component; instead, it continues the actions it was performing synchronously with its partner. One board, rather than two, is ORing signals to the bus and responding to other bus requests. The operating system software hides the unavailability of the failed board for boards not running in lockstep. After the failed board takes itself off line, the system increments a fault count associated with the board and recalculates the board's mean time between failures (MTBF) based on any earlier failures. If the MTBF is less than an administratively set value, the board is marked for replacement, and the RSN is invoked to report the board as a failed component. If the calculated MTBF is still greater than acceptable parameters, the board performs a series of self-tests. If it fails these tests, the failure is not transient, and the replacement process is initiated. If the board passes these self-tests, it is either reduplexed with its partner board (for synchronously lockstepped boards) or logically brought back into service (for logically paired boards).

• **Power Supply Failure:** If one of the two Stratus power supplies within a module fails, all of the boards driven by that power supply drop out while their partner boards continue to run at full capacity. If either power supply is operational, no battery backup hardware is needed.

• **Operational Downtime:** Operational downtime because of improper action as a result of a hardware failure is almost unheard of. The board that failed is self-identifying, so it is unlikely that the wrong board will be removed for repair. Further, all board replacements are performed while the system is fully operational. It typically requires no more than a few seconds to integrate the replacement board into the module. No time is lost waiting for someone to notice that the system has failed in some way. When a failure does occur, it results in no downtime, and notification (including the CAC, usually responsible for initiating the correct repair action) is automatic.

• **Field Service:** The classical outages due to field service (shotgunning by trying to guess what might be wrong and restarting the system to see if the guess was correct) do not exist with Stratus systems. The self-diagnosing and self-identifying nature of the boards makes this approach obsolete. No preventive maintenance is required for the central system components and therefore no downtime results. As mentioned previously, repair of the system by replacing boards does not affect system operation.

• **Hardware Installation:** Installing new hardware to expand or upgrade a module or system does not require any downtime. In fact, when new hardware is dynamically added, the system automatically makes such hardware available to already running applications. For example, additional processor boards can be added to a system, and the software can immediately benefit from the increase in processing power (due to the symmetric multiprocessing architecture of the operating system). Similarly, memory boards and disks can be dynamically added to a live configuration and are then immediately available for application use (without rebooting the system).

ARCHITECTURE TRADEOFFS

The entire Stratus architecture reflects tradeoffs between simplicity, ease of maintenance, lifetime system costs, logistics concerns, and technology trends on one hand and slight increases in product cost on the other hand. The resulting products clearly justify such tradeoffs.

First, the Stratus solution requires more hardware. Any truly fault-tolerant system requires more hardware than a conventional system, but a Stratus system needs, in some cases, four times the amount of hardware. However, the cost of hardware (primarily logic chips) that Stratus must quadruplicate is low and is decreasing, becoming a less significant part of the total cost of a computer system, particularly when measured over the lifetime of the system. The components with the most significant hardware costs (mainly on-line memory and peripherals) need only be duplicated (or, in the case of disks, redundant in some way), and any truly fault-tolerant computer has these same requirements.

Second, to design circuits that use Stratus concepts, logic chips must be totally deterministic. This has been problematic in the past, but it is becoming less so, as chip manufacturers become more aware of the needs of architectures built on Stratus concepts. Stratus has received commitments from several chip manufacturers for the future production of chips with totally deterministic behavior.

Third, since the logic circuits depend on total synchronization, anything that detracts from this synchronization can be a potential problem. The primary sources of difficulty in this area are new revisions of chips (new masks, usually to fix bugs). Chips that behave slightly differently from earlier revisions are harmless to all other architectures, but to the Stratus architecture, they can be devastating. (If the differing chips are on the same board, the board automatically diagnoses the difference and stops itself. If the differing chips are on separate boards, the boards cannot be synchronized or stay synchronized.) Therefore, Stratus must be sensitive to the revision level of the chips that are placed on boards. Extensive onboard self-identification hardware allows the operating system software to reject boards that do not meet synchronization requirements.

STRATUS SOFTWARE

The Stratus approach to fault tolerance does not depend on a particular operating system; however, software must be capable of managing disk mirroring, redundant intermodule connections, and the diagnostics and maintenance functions necessary to test, remove, and install replacement or expansion hardware. Further, as noted earlier, extensive work is needed to improve the quality and fault tolerance of system software. This work includes a well-structured development process, of course, but Stratus has been able to transcend normal quality levels with its extensive knowledge of system failure scenarios (through the RSN) and with the use of software fault-recovery techniques described later.

Stratus chose to develop its own proprietary operating system, the Virtual Operating System (VOS). When VOS was developed, no other commercially available operating system provided the necessary features. These features include support for the following: multiple processors running in a tightly coupled, shared main memory configuration; disk mirroring; a file system that supports all high-level features of COBOL, as well as transaction-protected files; worldwide networking, providing a single system view of thousands of modules; and the previously mentioned features needed for fault tolerance.

Stratus has since decided to support the UNIX operating system, as well as VOS. The following discussion refers specifically to VOS, but all of the concepts leading to high-quality software are also used in the Stratus UNIX offering, FTX.

Processes

VOS is a process-oriented operating system based on a procedure-call model. It differs from other general-purpose operating systems in its support for multiple processors, emphasis on the client/server model, ability to provide transaction-protected distributed databases, management of fault-tolerant issues such as disk mirroring, and its relative newness.

Programs run within processes that call upon the operating system to complete tasks. A typical call to the operating system does not switch processes (as would be the case with a message-based operating system); rather, it verifies that the arguments to the operating system entry are valid and optionally switches the process to a higher level of privilege. For intermodule or intersystem calls, call arguments are placed in a message that is sent to the appropriate server module. Message-passing primitives are also provided for interprocess communication, particularly for applications that may grow to require multiple modules. These message-passing primitives are vital to the recommended method of developing applications: a client/server process organization.

Processes play a key role in application development but also are a means of implementing some of the basic concepts of Stratus fault tolerance. System processes are created, usually at module initialization time, to manage the diagnostics and maintenance of the hardware. Similarly, system processes are created to interface to the remote service network (RSN).

System processes also implement VOS networking. Server processes perform remote file operations for the benefit of client processes. This client/server relationship is similar to the client/server relationship recommended for applications. Any process, however, can be a client process merely by requesting some remote operation that must be processed by a server on some other module. The difference between VOS message passing and pure message passing in other operating systems is that local requests do not require the client/server process switch and other overhead. VOS becomes a message-passing (distributed) operating system only for off-module requests. Finally, an important use of processes is to implement the distributed transaction management required by any transaction processing system.

Access Control

VOS provides extensive access control facilities to protect data and access to modules and systems in general. Access control lists, passwords, data encryption, and privilege levels protect data.

Distributed Transaction Protection

An important attribute that contributes to the success of VOS in the OLTP marketplace is its ability to support the concept of a *distributed transaction*. A distributed transaction requires interaction between client processes that typically start and commit transactions, server processes acting on behalf of the client processes, and transaction processing (TP) overseer processes that manage transaction processing, including the TP log files and TP recovery after a module interruption.

The TP overseer processes on each module communicate using message passing and remote procedure calls to effect a *two-phase commit protocol*. This protocol guarantees that the distributed databases involved in a transaction are kept consistent, even when some modules involved in the transaction may be unavailable for extended periods of time.

The methods used by Stratus involve TP log files that contain afterimages of the database, flushed to disk at commit time. These log files are applied to the real database files as time permits. VOS also provides the ability to back up transaction-protected files during operation and to perform database roll-forward after a system interruption.

Software Fault Recovery

Since software problems are a primary cause of unavailable applications, Stratus has devised methods of software fault recovery that make its operating system more resilient than most other systems. The extensive data structure locking needed to implement full support for multiple tightly-coupled CPUs is the basis for the operating system strategy. Since every shared data structure within the operating system must be protected from simultaneous updates by multiple CPUs, an extensive locking protocol is necessary. If a data structure is not locked, it is in a consistent state. If a data structure is write locked (where a distinction exists between read locks and write locks), it is potentially inconsistent.

If a fault occurs within a process that is executing within the kernel, or if the process detects a problem through the use of its own checking software, fault recovery procedures that check which data structures might be inconsistent are automatically invoked. The check simply sees which data structures are write locked by the faulting process. Each lock has an associated procedure to call in case the fault recovery software detects the lock set at recovery time.

By establishing recovery procedures for each locked data structure and by defining the data structures so that the called recovery procedures can determine how to make the data structure consistent, many operating system software bugs can be rendered harmless. Many key software packages within the operating system have been designed to operate using this software fault recovery strategy.

Diagnostic Software

Several processes are created to manage the diagnostics, reporting, and maintenance of the hardware. The key processes are the maintenance process and the diagnostic process. The maintenance process determines if a significant event (such as a board failure or the removal or insertion of a board) has occurred. When the maintenance process detects a significant event, the diagnostic process is notified and directed to test the hardware involved.

The maintenance process uses two distinct techniques to determine when events of interest occur. The first technique uses a hardware interrupt signal generated by a board when it breaks. This interrupt is referred to as a *red-light interrupt*. Since a board could break in such a way that it might not be able to set this interrupt, the maintenance process also uses another technique. This second technique polls all boards plugged into the StrataBus to determine if any boards have been inserted, removed, or broken. The typical polling interval is 10 seconds.

When the maintenance process notices a change, it places a request in the diagnostic queue serviced by the diagnostic process. The queue entries typically request the diagnostic process to test a piece of hardware. However, software must occasionally be downloaded into a line adapter before it can be fully tested. The diagnostic process also performs this action. If a piece of hardware is diagnosed as broken, the hardware is left off line and the RSN notifies the Stratus Customer Assistance Center (CAC). If the board error appears to be transient, it is placed back on line and synchronized with its partner if necessary. If too many transient errors are encountered in a specified period of time, the board is declared broken, and the software initiates the process of calling for a replacement board.

System Log Files

One last system process that plays a role in the overall diagnostic management of a module is the overseer process. Among other things, it maintains various log files that record events of interest to system operation. The log files of primary interest are the *system error log* and the *hardware log*. These files are written to by the system at appropriate times by buffering messages in operating system space. The overseer process then copies the messages into actual files. The hardware log file consists of hundreds of message types, all adhering to a standard, formatted structure. All log messages are time stamped. Application software can be notified whenever a message is placed in these files and can therefore reflect these messages to a terminal or some other file. When the RSN packages a message to send to the CAC about a failed piece of hardware, the log files are scanned, and any entries within the file related to the failed hardware are copied into the package sent to the CAC.

As noted earlier, the RSN plays an important role in improving the reliability of Stratus software by notifying the Stratus CAC whenever a system interruption occurs. This gives Stratus engineering personnel extensive insights into the types of problems that occur in the field. Whenever a module is booted, it sends a message to the CAC over the RSN indicating why the reboot occurred. If the reboot was triggered by a software crash, the reason for the crash is available through the remote dump analysis software available through the RSN. In general, the reboot messages provide a complete and accurate tracking of the availability of Stratus systems. This information is used to continuously improve upon the target of continuous availability.

SERVICE STRATEGIES

One of the important design issues for VOS and for Stratus hardware was the strategy for servicing the computers. The following paragraphs compare the Stratus approach to the then current approaches of other computer companies. (Since the introduction of Stratus systems, some of the techniques described have been adopted by other companies.)

The Classical Repair Approach

A classical approach to computer repair follows a scenario such as the following:

1. Someone notices that something is wrong. Either the computer does not seem to be giving the right answers, or it seems to have crashed.
2. Someone, usually a vendor field engineer, tries to determine what might be broken. The problem may be hardware- or software-related.
3. If hardware is suspected, a shotgun approach is often used, whereby boards are replaced singly or in groups in an effort to determine where the problem lies. To aid in determining which boards may be problematic, diagnostic programs are often run, usually requiring a dedicated machine.
4. An attempt is made to check whether or not the fix has been effective. Often, this amounts to little more than seeing if the system seems to work. Such a method is rarely exhaustive, scientific, or conclusive.
5. If the problem still exists or recurs within a few days or weeks, the process loops back to step 2 again.

The disadvantages with this scheme are obvious. The process is expensive, time-consuming, error-prone, unconvincing, and usually makes the system unavailable for extended periods of time.

The Stratus Service Approach

The Stratus approach contrasts with the classical approach in many ways. Hardware errors and software errors may be distinguished with complete confidence. If there is no red light on any board, the problem is a software problem. If there is a red light on a board, that board is broken. If the board failed because of a transient error, the system has already tried to reinstate it into the system several times before giving up and leaving the red light on. (There is no need to diagnose what is wrong with the board; it will not be repaired until it is brought back to the Stratus manufacturing facility.)

The RSN (Remote Service Network) plays a key role in Stratus maintenance. The RSN software provides a means for computers at Stratus customer sites to inform the Stratus CAC of broken or failing hardware automatically. It also provides a communications path for such activities as remote maintenance by CAC personnel, exchange of files, on-line updates, and bug fixes.

The CAC periodically uses the RSN to poll sites; to gather configuration information, error statistics, and system release information; and to check on the consistency of software versions. Customer sites use the RSN to ask both technical and nontechnical questions of CAC personnel. The RSN also informs Stratus publications personnel about problems with or suggestions for manuals.

Many customers are concerned about the implicit access to their files that the RSN might provide to CAC personnel. Extensive protection is built into the RSN software facility, and most customers are completely satisfied with the level of protection provided.

When a hardware error does occur, the RSN immediately informs the Stratus CAC. Appropriate local operations personnel can arrange for immediate notification on their terminals as well.

The advantages to the Stratus approach are clear. First, errors do not result in emergency situations; the system continues to run at full capacity. Second, since the repair can usually be made by onsite, untrained staff personnel, there is a considerable customer savings. The parts necessary for the repair are typically sent to the site automatically from the Stratus manufacturing facility using overnight courier. After repair, these parts undergo a complete testing cycle as rigorous as the initial manufacturing of the parts.

SUMMARY

The introduction of computer systems based on the Stratus architecture provides businesses of all types with new levels of system availability and serviceability. The architecture has proven portable to new I/O systems and faster chip technologies. It provides a platform on which software not cognizant of issues of fault tolerance can provide continuously available solutions. Finally, it provides relief to the ever-increasing threats of rising service costs and quality degradation. Stratus systems and their customers enjoy the highest satisfaction for service and quality.