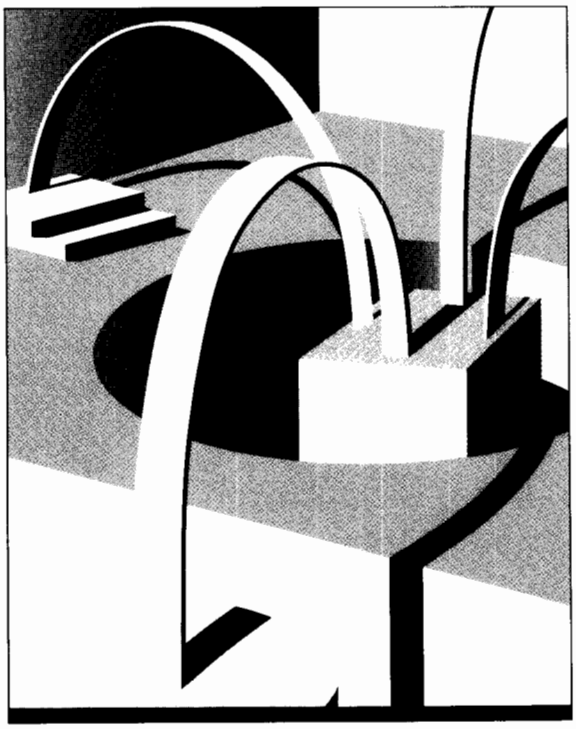
T A N D E M

SYSTEMS REVIEW

VOLUME 3, NUMBER 2

AUGUST 1987

BRANDIFINO



Correction:
Please note the following correction to the article titled "Performance Measurements of an ATM Network Application," which appeared in the December 1986 issue.

Page 103, Table 3. The numbers under the heading *Transactions per month* were calculated in millions. For example, Bank of America averaged 13.0 million transactions per month (as of September 1985).

Volume 3, Number 2, August 1987

Editor
Ellen Marielle-Tréhoüart

Associate Editors
Wendy Osborn
Carolyn Turnbull White

Assistant Editor
Sarah Rood

Technical Advisors
Mark Anderton
Bart Grantham
Dick Thomas

Cover Art
Stephen Stavast

Production and Layout
Niklas Hallin
Janet Stevenson

Typesetting
Tandem Typography

The *Tandem Systems Review* is published by Tandem Computers Incorporated.

Purpose: The *Tandem Systems Review* publishes technical information about Tandem software releases and products. Its purpose is to help programmer-analysts who use our computer systems to plan for, install, use, and tune Tandem products.

Subscription additions and changes: Subscriptions are free. To add names or make corrections to the distribution data base, requests within the U.S. should be sent to Tandem Computers Incorporated, *Tandem Systems Review,* 1309 South Mary Avenue, LOC 5-04, Sunnyvale, CA 94087. *Requests outside the U.S. should be sent to the local Tandem sales office.*

Comments: The editors welcome suggestions for content and format. Please send them to the *Tandem Systems Review*, 1309 South Mary Avenue, LOC 5-04, Sunnyvale, CA 94087.

Enhancements to TMDS

The smooth functioning of the hardware in Tandem™ systems is a shared responsibility between the customer's system manager and Tandem's customer engineer (CE). The Tandem Maintenance and Diagnostic System (TMDS), a diagnostic software system for Tandem hardware, helps them perform their jobs. Although TMDS is known primarily for its abilities on the NonStop VLX™ system, TMDS runs on all NonStop™ systems. It is distributed automatically to all Tandem service customers and runs under the GUARDIAN 90™ operating system. For a description of TMDS's basic capabilities, refer to an earlier issue of the *Tandem Systems Review* (Troisi, 1985).

In addition to providing information on the state of Tandem hardware, TMDS contains an automatic fault analysis (AFA) portion. This most recent enhancement uses the artificial intelligence techniques of expert systems.[1] Though the VLX has had fault analysis since the B30 software release, starting with the C00 release, fault analysis will be available on all NonStop systems, diagnosing the peripheral equipment, such as disks and tapes.

This article describes the features provided by TMDS, its components, and uses. It also gives an overview of the new AFA portion of TMDS and concludes with future directions.

[1]An expert system is a program that emulates the knowledge of a human expert in a particular field. The human expertise is represented in a knowledge base, and an inference engine uses that base and other state information to reach the needed conclusion.

Features

TMDS delivers its diagnostic benefits within the on-line processing environment while observing certain design goals that ensure the continued health of the system. Those goals include minimizing the consumption of system resources, causing the least possible impact on customer applications, and maximizing safety during the diagnostic process through careful failsafe mechanisms.

TMDS offers the following features:

- Diagnostic tools are consistent, easy to use, and do not require an intimate knowledge of hardware subsystems, their status codes, or specific error values.
- Hardware subsystem problems are diagnosed on-line with a minimum disruption of system activity.
- Diagnostics yield meaningful output (such as the isolation of a faulty field-replaceable unit).

Two Distinct Functions

TMDS's hardware and software can be divided into two distinct functions: the diagnostic portion, used by the Tandem CE to run device diagnostics on the system hardware; and the monitoring portion, which continuously monitors and analyzes the system. (Figure 1 shows the architecture of TMDS with fault analysis.)

The diagnostic portion of TMDS contains a core of commands that apply to all subsystems and additional commands that apply

Figure 1

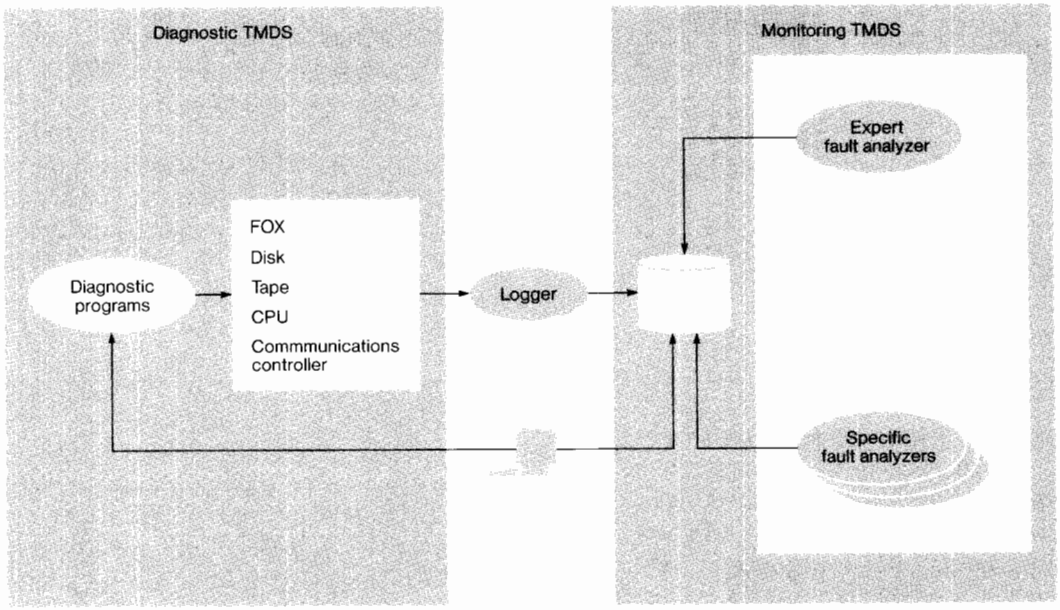


Figure 1.
TMDS architecture.

to specific subsystems. For example, the SWITCHTO command is used to switch to a specific subsystem. That subsystem contains tests and diagnostics for devices specific to that subsystem. (See Figure 2 for more detail.)

TMDS's monitoring portion is designed to track the system's hardware status. (See Figure 3.) This portion contains AFA to facilitate the diagnosis process. Fault analysis is discussed in more detail later in this article.

Figure 2

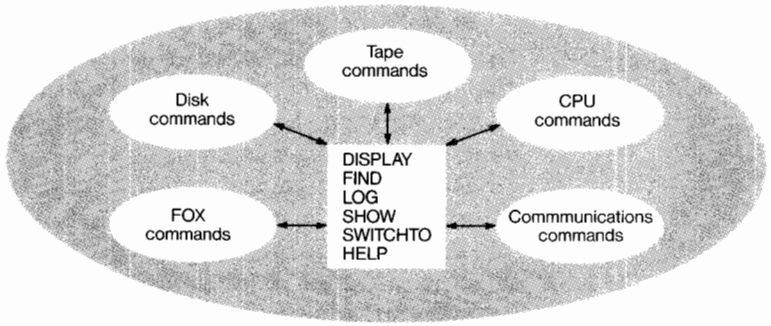


Figure 2.
TMDS commands.

A System Management Tool

Frequently, TMDS notices equipment problems before they affect the operation of the system. This makes it a valuable system management tool. It is useful not only to the CE doing a job, but also to the system manager overseeing the health of the hardware. The following features are useful in system management:

- *Event log*, a historical document that contains entries of hardware status events. Fault analysis uses the data in the event log to make its diagnoses.
- *DISPLAY*, a command that shows the user all system problems determined by TMDS.
- *FIND*, a command that allows users to browse the log for various occurrences.

In addition to these features, the CE also uses diagnostics to identify problems that fault analyzers (FAs) don't cover. TMDS allows the CE to perform diagnostics on-line. In fact, many diagnostics can be performed while the device itself is on-line, the worst case being that only the device needs to be downed. Under the previous diagnostic systems, the device and its controlling CPU both needed to be down.

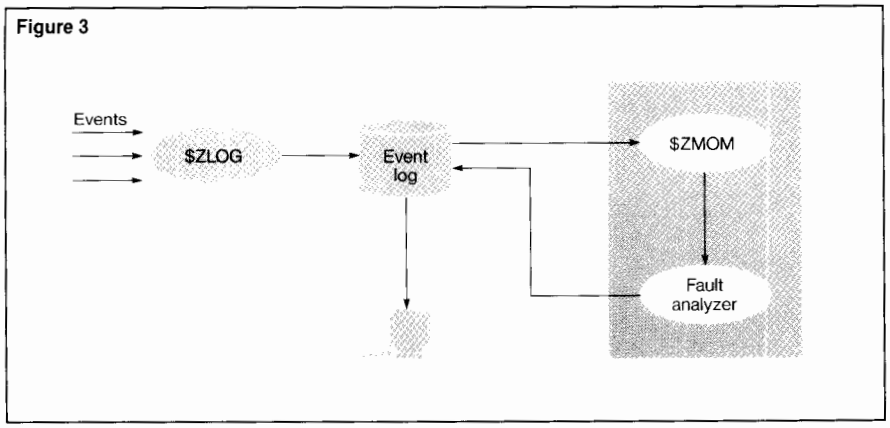


Figure 3.
TMDS monitoring.

Improved Service

TMDS gathers information required by CEs. In fact, CEs were involved in the early design of TMDS to ensure that the information they need most gets collected. CEs can now access information about hardware involved in the occurrence of specific events—information such as specific product type, controller addresses and paths, and the current operation. TMDS also aids the CE by capturing intermittent errors that are difficult to recreate in a diagnostic environment.

Retrieving relevant data from the TMDS event log is now faster and easier. The CE can request event history by device name starting at any time on any date (e.g., yesterday at noon or two weeks ago at 9:35 a.m.).

By viewing a device's event history, the CE can examine the specific actions leading up to the problem being investigated. The CE can sometimes forestall a potential problem because the early signs are detected and corrected. Trends in the type of event being logged and the frequency of the occurrence are often signs of potential trouble.

When the system is linked to the Tandem National Support Center (TNSC), the history can also be monitored remotely. Remote diagnosis can sometimes prevent small problems from developing into serious problems, without requiring a CE on-site. If a problem detected by the TNSC cannot be solved remotely, a CE will be assigned to troubleshoot the problem on-site. However, the CE confers with the TNSC on the circumstances surrounding the problem before leaving the support office, and is therefore better able to determine the equipment or parts required to solve the problem.

The customer's system manager is able to help monitor system performance by running the DISPLAY, FIND, and STATUS commands on the system on a regular basis. As the person most intimately concerned with the system's daily operations, the system manager monitors the TMDS event log as a natural extension of ensuring that the system is running at its best. If the system manager has any questions or doubts about the system's performance, a Tandem representative can be contacted for resolution.

Flexible On-line Diagnostic Testing

After examining the system's event log, a CE or a TNSC specialist initiates the necessary diagnostic tests to aid in the evaluation. The CE can run the entire series of tests or select exactly which segments of the device or system require more in-depth attention. This flexibility saves time by running only the pertinent tests. And by allowing the CE to customize the testing pattern, the problem can be evaluated more accurately.

In addition to flexibility, TMDS's testing and system verification are performed on-line with minimal effect on system performance. No system resource other than the device in question need be engaged to run tests or verification after repair. Processing continues unhindered by the diagnostic tests, unless the processor itself is being tested. This means that CEs can run their tests immediately, without having to wait for an opportunity to dedicate the processor to diagnostics. In the past, waiting for the processor often meant a second trip for the CE, which left the system running in a degraded mode.

Automatic Fault Analysis (AFA)

Traditionally, hardware and software failures on Tandem systems have been reported via messages sent to a console device. It has been the responsibility of the operator to monitor these messages, interpret them, and take appropriate actions. TMDS automates much of this work for the system operator by providing AFA. AFA provides the automatic detection, analysis, and reporting of hardware and software failures. Results of the analysis are reported locally to the event log. On NonStop VLX and NonStop CLX™ systems the analysis is reported remotely to the Tandem On-line Support Center (OSC) (Allen, 1987).

TMDS fault analysis is based on packets of information known as event signatures. These machine-readable buffers are generated by low-level software products to report hardware failures, state changes, and other statistical information vital to fault diagnosis. By examining these signatures in detail, TMDS is able to isolate failures in the system.

Components and Implementation

Tandem hardware and low-level software have been designed so that when an unusual event occurs on the system, information about the event is captured and included in an event signature that is shipped to the TMDS process $ZLOG. An unusual event is any occurrence about which information must be retained for examination by a CE or diagnostic (e.g., when a disk drive goes into the down state, or when the temperature in a CPU cabinet goes out of range). The $ZLOG process receives event signatures and stores them in a TMDS event log on disk.

The $ZMOM process examines each event signature logged by $ZLOG. If the signature requires examination, $ZMOM opens a TMDS investigation, called a *case*, and starts the appropriate fault analyzer (FA) to investigate the event in more detail. (Cases are described in more detail later in this article.)

The FA performs detailed analysis on a very narrow range of events. Any conclusions reached are placed back into the TMDS event log and, in this way, are made available to the user and the TMDS system.

The entire AFA system is monitored by the TMDS DISPLAY command. This command provides information to the user on completed and ongoing fault analysis.

Cases

A case is a failure investigation. It is identified by a unique number that is included in all results related to the investigation.

A case is always in one of three states: open, solved, or closed. When an interesting event is noticed and an investigation started, a case is opened. The case stays in the open state until the results of fault analysis are placed into the log. At this point, the case enters the solved state, indicating that some analysis has been performed. The case remains in the solved state until it is closed by either a person or a machine. A closed case remains in the TMDS log for some time, thereby making the machine's failure history available to the user.

Opening Cases. All cases are opened by the $ZMOM process. $ZMOM is a rule-based program that contains a number of individual rule sets, each of which can interpret a small subset of events. A rule-based program is one in which the rules about how to perform some function—in this case deciding when to open a case and start an investigation—are kept separate from the code that applies these rules. This separation makes it easy to expand $ZMOM's capabilities as more subsystems are added to the TMDS system and as fault analysis becomes more comprehensive.

When an event signature (one that a $ZMOM rule set is interested in) is logged, the rule set is executed. Based on its conclusions, it takes one of several actions, including storing partial conclusions for later use, incrementing a counter, or opening a case. Partial results and counters are stored in a private memory associated with each rule set and can be used to extend analysis over several event signatures.

$ZMOM checks each new case opened by a rule set to see if the failure is related to an existing open or solved case. If it is, $ZMOM includes the new failure information in the existing case. If not, $ZMOM assigns a unique case number to the failure, and a special TMDS OPEN event is logged, announcing the opening of a new case. At this time, any FAs associated with the rule set are started. $ZMOM monitors the FAs to ensure that they run to completion, restarting them if necessary.

Once an event is completely processed, $ZMOM examines the next event in the TMDS log, and the cycle continues.

Solving Cases. Cases are *solved* by FAs. When an FA is started by $ZMOM, the appropriate event signature is identified in the TMDS log that caused it to be started. The FA processes this and other information in the log necessary to analyze a failure. When the cause of the failure is determined, the FA places a TMDS SOLVED event into the log. This event contains the results of the diagnosis, including the names of any broken parts in both a textual and programmatic form. This information is made available to users through the TMDS DISPLAY command.

Closing Cases. A case is *closed* when the incident it describes needs no further attention. Typically this is done by the user using the TMDS CLOSE command. The CLOSE command prompts the user for a case number and a synopsis of why the case is being closed. This information is included in a CLOSED event signature that is placed into the TMDS log where $ZMOM and other interested parties can access it.

CASES can also be closed programmatically by an FA under the following circumstances: when analysis indicates that there is no substantial failure; or when the failure can be corrected programmatically, such as in the case of automatic disk-sector sparing.

Monitoring AFA

AFA activity is monitored with the TMDS DISPLAY command. This command gives the user several views of the AFA system with varying levels of detail. Refer to Figure 4 for more information on the DISPLAY command.

Remote Support

In VLX and CLX systems, TMDS optionally relays the results of AFA to the OSC.

All solved events placed into the TMDS log by FAs are "dialed-out" to the OSC. In addition, open and closed events can be dialed-out to the OSC as requested by the generators of these events.

Dial-out is accomplished using the TMDS FA, DIALFA. Just as for all FAs, $ZMOM has a rule set that specializes in the events DIALFA is interested in. (See Figure 5.)

Open, closed, and solved events are examined to see if conditions justify starting DIALFA. If these conditions are met, a case is opened by $ZMOM for this dial-out attempt, and DIALFA is started. When started, DIALFA is notified of the event in the TMDS log that caused it to be invoked. DIALFA takes the information from this signature, adds some system-specific information, and ships the signature to the OSC. If the transfer is successful, the case related to the attempted dial-out is closed by DIALFA. If unsuccessful, a solved event describing the failure is logged in the TMDS event log. This notifies the user that a dial-out was attempted and failed.

Figure 4

(a)
The command below is used to display a list of cases that are in the SOLVED state, giving the user a list of known active problems on the system. The user enters:

```
TMDS>display *,detail on

Display for time 4 FEB 87 15:24 onward on system \TSII.
Print time is 18 FEB 87 15:24.

Case  State   Time                                   Subsystem/Device
2     Solved  18 FEB 87 14:17                        TAPE $TAPE2
                Probable FRU: 3206 Controller Board. Controller
                address: %3, Term code: %126 (Z80 test hard failure).
```

(b)
The following command is used to display all AFA activity that has occurred in the last two weeks. This information is useful for determining failure trends or problem areas. The user enters:

```
TMDS>display *,case all,detail on

Display for time 4 FEB 87 15:25 onward on system \TSII.
Print time is 18 FEB 87 15:25.

Case  State   Time                                   Subsystem/Device
1     Open    18 FEB 87 14:17                        TAPE $TAPE
                A TAPEES^SOFT^SELFTEST^ERROR event was received.
1     Solved  18 FEB 87 14:17                        TAPE $TAPE
                Probable FRU: 3206 Controller Board. Controller
                address: %3, Term code: %106 (Soft Z80 test failure).
1     Closed  18 FEB 87 15:00                        TAPE $TAPE
                Board was replaced.
2     Open    18 FEB 87 14:17                        TAPE $TAPE2
                A TAPEES^HARD^SELFTEST^ERROR event was received.
2     Solved  18 FEB 87 14:17                        TAPE $TAPE2
                Probable FRU: 3206 Controller Board. Controller
                address: %3, Term code: %126 (Z80 test hard failure).
```

Figure 4.
TMDS commands for monitoring AFA. (a) The command for displaying a list of cases in the solved state. The user is given a list of known active problems on the system. (b) The command for displaying all AFA activity that has occurred in the last two weeks. This information is useful for determining failure trends or problem areas.

Future Directions

TMDS is constantly extending its coverage of existing products, and as new products emerge, they are being incorporated into the diagnostic subsystem.

Since the basis for the entire TMDS system is event generation, there is ongoing work to improve event reporting. Tandem is using its experience to retrofit event reporting into older products as we develop new devices. It has been found that products must be designed with diagnosability in mind to ensure that information critical to diagnosis is transferred from hardware through controlling software to the TMDS log.

TMDS is also developing interactive diagnostic aids to direct and help the CE or user in troubleshooting the system. These aids will be a new rule-based system, separate from $ZMOM, that will be run as commands. The CE will consult with the system to diagnose a particular system problem. Producing these aids will help Tandem gather the rules and procedures that will enable more complete automatic FAs to be built.

Figure 5

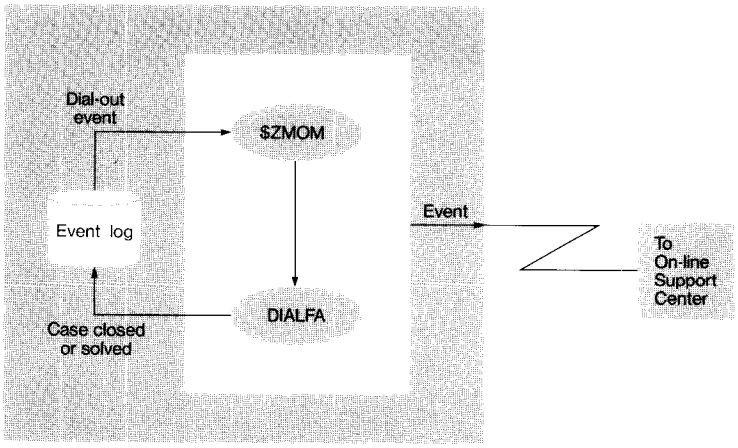


Figure 5.
TMDS dialing out to an event.

Tandem is also making products with improved on-line serviceability. For example, in the new CLX system, on-line repair aids guide *local assistants* interactively through the steps to repair the machine.

A local assistant is a customer staff member who has had minimal training on system operation and maintenance. The level of help and detail offered by these aids is adjustable by the user, so that the novice receives much more instruction than the experienced CE. TMDS leads the local assistant through each step and assures correct procedures are followed. For example, before instructing the assistant to remove a disk drive, the device is set in the DOWN state and powered OFF. After the unit is replaced, diagnostics are automatically run to ensure that the problem was corrected. The new unit is then set in the UP state.

Finally, TMDS is working to automate the machine-service process by allowing software to make explicit requests of the user. This is called *task* capability. For example, if an FA concludes that a part is broken and needs to be replaced, it can generate a task that calls for the replacement of that part. The task is available to the user through the TMDS DISPLAY command. When the user picks this task to work on, the proper repair aid is automatically run. This in turn can generate other tasks, such as part ordering, etc. In this way, the various facets of the TMDS diagnostic and service environment are integrated into one easy-to-understand user interface.

Conclusion

The TMDS fault analysis capability is the result of commitment from all levels of the Tandem product: hardware, low-level software, the operating system, and TMDS itself. This cooperative effort has introduced for the first time a coherent diagnostic and serviceability standard for present and future Tandem products.

References

Troisi, J. 1985. Introducing TMDS, Tandem's New On-line Diagnostic System. *Tandem Systems Review*. Vol. 1, No. 2. Tandem Computers Incorporated. Part no. 83935.

Allen, J. 1987. The VLX: A Design for Serviceability. *Tandem Systems Review*. Vol. 3, No. 1. Tandem Computers Incorporated. Part no. 83939.

Cindy Anderson Blain, a service marketing analyst, joined Tandem in 1984 as a production supervisor in the Cupertino systems integration and test facility. Later she worked in Corporate Materials Planning. Since joining the Service Marketing Group, she has worked primarily on the service aspects of new product introductions.

Leslie White is product manager for TMDS and AI. She joined Tandem seven years ago and has worked in various technical support positions as well as in product management. Prior to Tandem, she worked in database software development. Leslie has a bachelor's degree in Mathematics from the University of Colorado at Boulder.

Wes Witte joined Tandem in 1983 as a software designer for the TMDS group. His main focus for the last year has been the evaluation and integration of AI tools for Tandem use. Wes received his master's degree in Computer Science from Stanford University.

A class of magnetic tape drive called *streaming tape drives* or *streamers* is available. Streaming tape drives are less expensive, quieter, and more reliable than conventional tape drives. Because of this, the recent trend is toward streamers. Eventually, streamers will replace conventional tape drives in most applications.

Streamers have operational characteristics that must be accommodated to achieve good performance. If tape system software is not well suited to streaming operation, actual performance may be much lower than the streamers' potential. A tape system attempting to use a streamer in the same way it used a conventional drive may show poor performance, even if the streamer is nominally faster than the conventional drive it replaces.

This article describes how streamers differ from conventional drives and explains how to use them effectively.

Conventional Tape Drive Hardware

In a conventional tape drive, three parts move tape over the read/write head: the capstan, the reel motors, and the vacuum columns. (See Figure 1.)

Figure 1

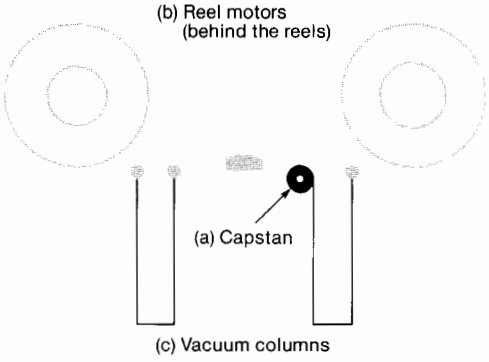


Figure 1.
A conventional tape drive has three parts that move the tape: (a) the capstan, (b) the reel motors, and (c) the vacuum columns.

The capstan is a wheel, usually covered with rubber. As it turns, it pulls the tape over the read/write head. The capstan is connected to a motor, a tachometer, and control electronics. These components form a feedback system that controls the tape speed to within a few percent.

The reel motors wrap and unwrap the tape on the supply and takeup reels. They can move the tape as fast as the capstan motor does, but they can't accelerate as quickly.

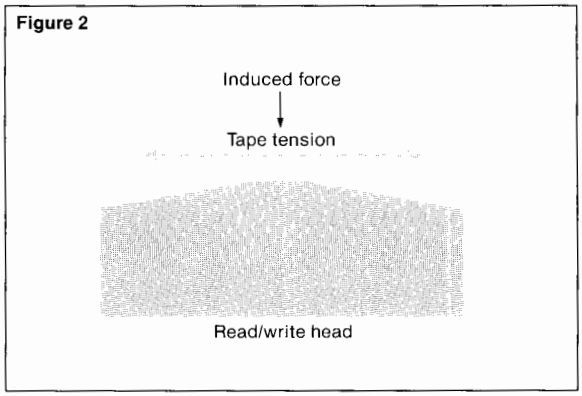


Figure 2.
The tape tension and the wrap angle around the head induce a force which holds the tape against the head.

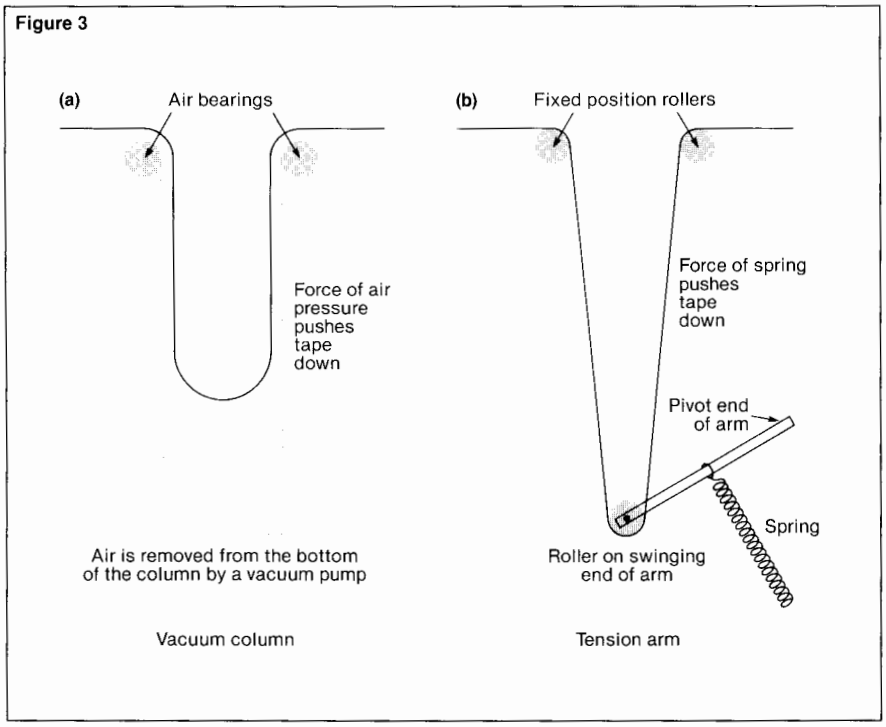


Figure 3.
(a) A vacuum column uses air pressure to put tension on the tape. (b) A tension arm uses a spring-loaded arm to put tension on the tape.

The vacuum columns regulate tape tension by providing a steady pull on the tape on both sides of the head. Tape tension induces a force against the head. (See Figure 2.) If the tape is under insufficient tension it will not hold to the head as the tape moves. Excessive tension may stretch the tape, causing data loss. Excessive tape tension may also wear out the read/write head prematurely.

Vacuum columns also act as buffers between the quickly accelerating capstan and the slowly accelerating reels. A capstan motor may accelerate the tape to 100 inches per second (ips) in milliseconds, while reel motors may take hundreds of milliseconds. Tape is pulled from one vacuum column by the capstan and pushed into the other vacuum column by air pressure. Before one column has emptied and the other has filled, the reel motors will have come up to full speed.

Some tape drives (usually less expensive, lower performance drives) substitute tension arms for vacuum columns. Figure 3 shows a vacuum column and a tension arm in detail. The vacuum column can supply and take up tape faster than the tension arm. It is also gentler on the tape at high rates of acceleration than the tension arm. The tension arm is preferred for slower speed tape drives because it is less expensive, and having no pneumatic components, it is quieter, more reliable, and easier to maintain.

Conventional Tape Drive Operation

With a conventional tape drive, system commands go to the capstan. Capstan commands are:

- Accelerate tape and run forward.
- Accelerate tape and run backward.
- Stop tape.
- Rewind tape.

Sensors in the vacuum columns control the motion of the reel motors. As the capstan fills a vacuum column, the nearest reel motor empties the column. As the capstan empties a vacuum column, the closest reel motor refills the column. The reel motors generally keep both vacuum columns about half filled with tape.

A feedback system controls the capstan speed. A separate feedback system controls the reel motor speed. Most conventional tape drives read and write at a tape speed of 25 to 200 ips. They typically rewind at two or more times the speed at which they read and write.

A conventional tape drive operates in either start/stop mode, or in streaming mode. Figure 4 illustrates the difference.

Figure 4a shows tape speed as a function of time. One data block is written. At time W, the drive receives a write command and accelerates the tape (to 100 ips in this example). By time X, the tape is at full speed. In the time required to bring the tape to full speed, part of an interblock gap has been put on the tape.

In this example an interblock gap is 0.6 inch. Of this, 0.150 inch is put on the end of each record by the head during a read-after-write check because the read gap is 0.150 inch behind the write gap. Between times W and X, 0.225 inch of interblock gap is put on the tape.

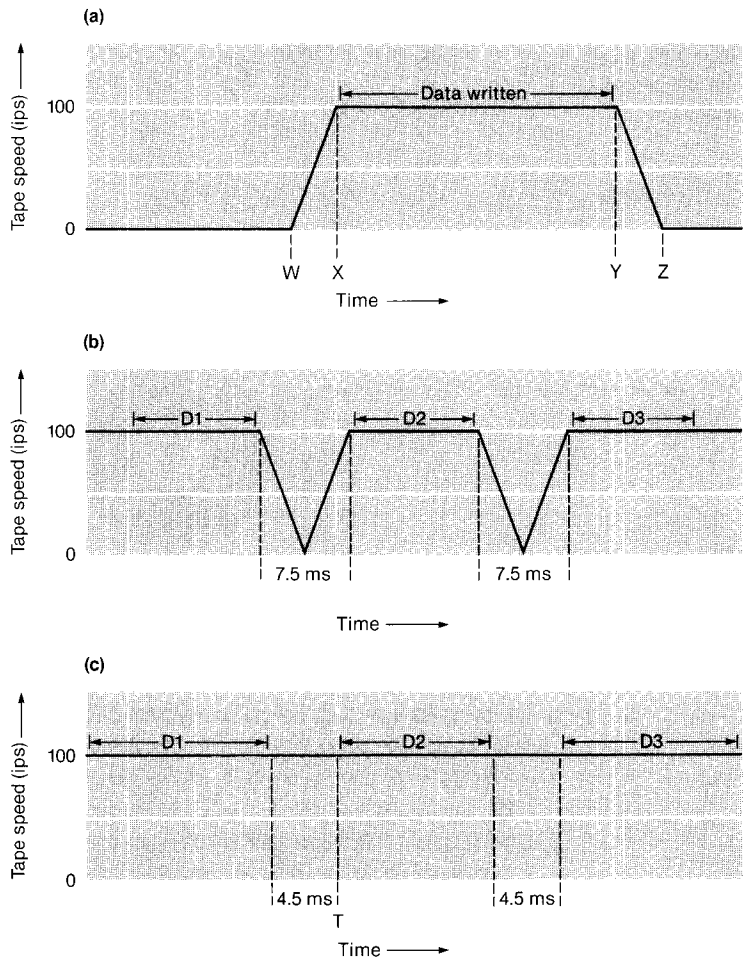
When the tape is at full speed, the data block is written. As soon as all of the data block has been written (at time Y), the tape is slowed to a stop. This takes until time Z. In this example, 0.225 inch of the next interblock gap is put on the tape between Y and Z.

For a 100-ips, nine-track tape drive, the time from W to X is about 3.750 ms. The time from Y to Z is also about 3.750 ms.

Figure 4b shows the drive writing several blocks. For every block there is a 7.5-ms overhead for starting and stopping the drive in each interblock gap.

Figure 4c shows how to eliminate 40% of this overhead. The drive keeps the tape moving at full speed between writing blocks. This is called *streaming mode operation*. It increases the average data rate but can only be used if a data block is always ready to be written following each interblock gap. The computer must have the data for block D2 ready by time T. If time T is reached and the data is not ready, the drive must be slowed to a stop after a complete interblock gap has already been put on the tape. Beginning to stop the drive at time T results in an extra long interblock gap following block D1.

Figure 4



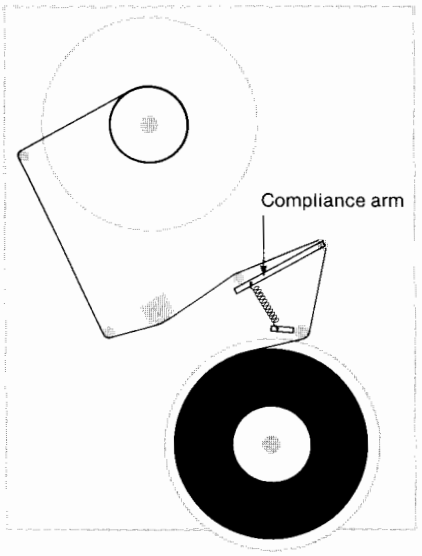
To maintain a standard interblock gap on the tape, the drive will assume that it is to stop after each block unless it is told very soon after writing a block that another is to be written. This delay is called the drive's *reinstruct time*.

Figure 4.

Conventional tape drives operate in both start/stop and streaming mode. (a) Writing a single record in start/stop mode. (b) Writing several records in start/stop mode. (c) Writing several records in streaming mode.

Figure 5.
A streaming tape drive. Unlike conventional drives, it has no capstan drive, vacuum columns, or tension arms.

Figure 5



How Streamers Work

A streamer is a conventional tape drive that has had the most expensive (and least reliable) portions of the tape path removed. It can *only* operate in streaming mode. It does not have a capstan or vacuum columns (or tension arms). In addition to improving reliability and lowering cost, this reduces size and noise. (See Figure 5.)

The reel motors (usually under microprocessor control) move the tape at precise speed over the head. The motors cannot start or stop the tape quickly, nor can they start or stop in an interblock gap. Instead of starting in 3.750 ms, they may require 200 ms to bring the tape to full speed.

Many open reel streamers have small compliance arms to maintain tape tension. Quarter-inch cartridge streamers regulate tape tension by pulling the tape through the cartridge with an elastic belt. Tandem's model 5120 half-inch cartridge tape drive uses microprocessor control of reel motor torque to regulate tape tension.

Repositioning

Earlier it was noted that streaming mode is only possible if a block of data is always available when the device needs it. If the computer system does not meet the reinstruct time limit, the drive should recover in some way that does not lose user data. Streamers usually use a technique called *repositioning*.

Figure 6a illustrates repositioning with a graph of tape speed as a function of time. When the drive finishes writing the first data block (time = A), the computer system has until the reinstruct time limit to issue the next write command. If the system fails to do so within the reinstruct time limit (time = B), the drive slows to a stop (time = C), runs the tape in reverse, and stops (time = F). The drive will wait with the read/write head positioned on the block previously written until the next write command arrives. When it gets another command (time = G) the drive accelerates the tape, spaces over the data block D1, and writes the next data block D2 (time = H).

Figure 6b is another way of looking at the same time sequence. It shows how the tape is positioned over the read/write head during repositioning.

Because the time between A and F is so long (hundreds of milliseconds to several seconds), the computer system usually has more data ready to write by time F. Assuming that time between F and G is zero, it is possible to define the time from A to H as the *reposition cycle time* of the drive. The time from B to G is called the drive's *reposition time*. The time from F to H is called the drive's *access time*. Typical reposition times are 600 ms to 3 seconds. Typical access times are 100 ms to 300 ms.

There is one case where repositioning does not reduce system throughput. If a program is infrequently logging data to the drive (e.g., one write every minute) and the program does not wait for the reposition to complete before continuing processing, the reposition time is overlapped with processing of the next write and does not affect system performance.

Reduced Interblock Gaps

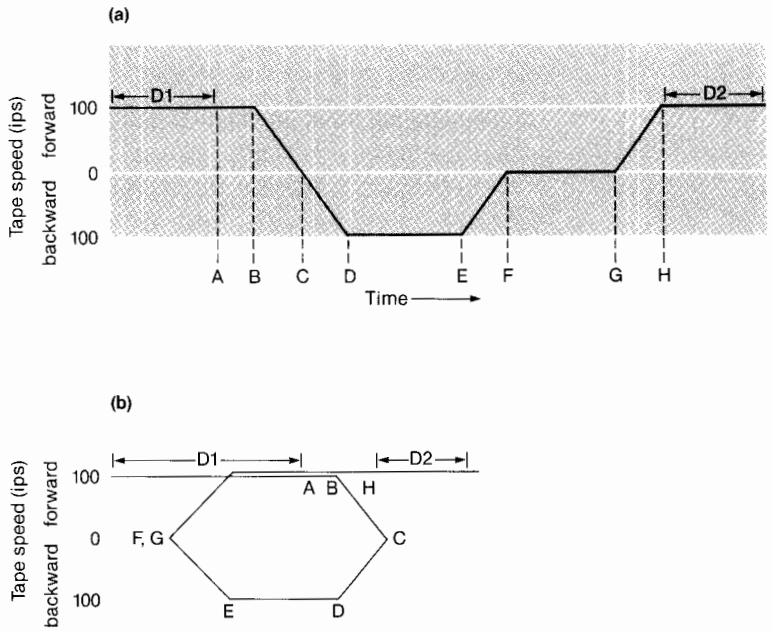
Streaming mode makes very small interblock gaps possible. If the system always has data ready for the next block while writing the current block, it isn't necessary to write any interblock gap at all. Even though streaming drives could write open reel tapes with very small interblock gaps, this is not done because these tapes could not be read by conventional tape drives. The standard interblock gap for NRZI (800 bpi) and PE (1600 bpi) tapes is 0.6 inch; GCR (6250 bpi) tapes have 0.3-inch interblock gaps as a standard.[1]

Most cartridge tape drives are designed for use in streaming mode with repositioning. With no requirement for interchange with start/stop drives, these tape formats have a very small interblock gap. (One format for the quarter-inch cartridge has a 0.03-inch interblock gap.) This increases tape capacity and overall data transfer rate.

The Tandem 5120 tape subsystem has a 4.5-inch interblock gap in start/stop mode and a 0.8-inch interblock gap in streaming mode where repositioning is used.

[1]NRZI = Non-return to Zero Inverted; PE = Phase Encoded; GCR = Group Code Recording.

Figure 6



Getting Good Performance with Streaming Tape Drives

Using a streaming tape drive in a system designed for a conventional drive is likely to result in poor performance. The streamer may be slower in the system environment than a lower speed conventional drive because it spends time repositioning. For example, a 45-ips conventional drive may give better performance than a streamer rated at 100 ips if the streamer must reposition often. Following are some techniques—some available to application programmers, others limited to operating system or hardware developers—for getting good performance with streaming tape drives. The list at the end of this article gives recommendations for getting the best possible performance from streamers.

Figure 6.
Repositioning. (a) Tape speed changing with time. (b) Tape position changing with time.

Improving I/O System Speed

Repositioning can be avoided by making the computer I/O system fast enough to keep up with the drive. If the reinstruct time limit is always met, repositioning is unnecessary.

Bypassing File System Overhead. One method used to improve I/O system speed when backing up disks is to bypass the overhead of the file system. A program doing such a "physical" or "image" backup reads disk sectors and writes them to tape. This method of disk backup eliminates file open and close operations, as well as most disk seek time. It maintains a high, uniform data transfer rate from the disk which is well suited to transferring data to a streaming tape drive.

Because the checking, protection, and security mechanisms of the file system are bypassed, physical I/O is generally not available to application programs. In the Tandem environment it is only available to system utilities, such as the volume mode operation of Tandem's BACKUP and RESTORE tape utilities.

Transfer Overlap. Another approach to improving I/O system speed is to overlap disk and tape transfers. Two areas in main memory are used as buffers. The disk process fills one as the tape process empties the other. This "double buffering" technique is used by Tandem's BACKUP and RESTORE tape utilities to increase device and I/O process utilization.

Data Buffering. Even if the system is able to sustain the average data rate of the drive, there may be a problem with short-term variance in the system's data rate. It is not sufficient to have an average data rate equal to the drive's data rate. Every reinstruct time limit must be met, or repositioning will occur. If a program produces data for the tape in bursts, some type of buffering should be used to even out the flow of data and match it to the streamer's data rate.

Buffering can be done in many places in the system. A program can combine output records into blocks before writing them. Programs can write data blocks to a disk file and periodically transfer the file to tape. The operating system can provide multiple buffers so that physical I/O is performed by the controller while the program prepares another block for output. An I/O controller can contain multiple block buffers.

If a fixed number of block buffers are available, it is better to write large blocks into them. Increasing the block size reduces the amount of time spent on overhead relative to the time spent doing data transfer. Improvements may involve rewriting programs or simply specifying different block sizes to utility programs. For example, the Tandem BACKUP program defaults to an 8-Kbyte block length but permits the user to increase the block length with command line parameters.

A related issue is file size. Typically, end of file processing will require a reposition cycle. Regardless of block size, tapes that hold many small files require more time to read or write than tapes that hold a small number of large files.

Increased Reinstruct Time. Another way to keep a drive streaming is to increase its reinstruct time. A common technique is interblock gap extension. On a 100-ips nine-track streamer, every 0.1 inch added to the interblock gap gives an additional millisecond of reinstruct time. Some drives permit an extended gap as a programmable option, while others extend the gap automatically each time the computer fails to meet the standard reinstruct time limit. This can substantially reduce the tape capacity if done frequently, particularly if the tape has many small blocks written on it.

Reduced Tape Speed. A final technique that may be used to balance the system data rate with the tape data rate is tape speed reduction. A low-speed drive without repositioning may give better performance than a high-speed drive with repositioning. For this reason, a 50-ips streamer may outperform a 75-ips streamer under some conditions.

Combining Techniques. Frequently a combination of these approaches is used. For example, it may be better to implement buffering and accept a small number of reposition operations rather than to reduce the tape speed.

Data Integrity Considerations

If data buffering is used to keep a drive streaming, there are data integrity considerations. The program will continue processing as soon as a data block is written into the buffer, rather than waiting until it is written to tape. Consider the case where the program has written three blocks to the buffer and only the first is written to tape. If there is a write failure in the tape drive, the program must somehow be informed about which blocks are on the tape. One approach is to define how the data is committed to tape and let each program manage the buffering. There are three typical modes of operation.

Implicit Record Commit
The implicit record commit makes programs wait for data write completion on every record. This allows the same error-recovery methods as conventional start/stop drives but will be slow because the drive will reposition every time a record is written.

Explicit Record Commit
This requires programs to explicitly commit data blocks to tape with a Synchronize command. Data streaming is possible, and programs may use record commit checkpoints to guarantee that the buffered data blocks are written to tape (at the cost of a reposition cycle time).

Implicit File Commit
The implicit file commit writes all buffered data blocks and repositions the tape when a file mark is written. This provides file level recovery and is usually acceptable for most applications.

The Tandem 5120 drive operates with implicit record commit when in start/stop mode and combines implicit file commit with explicit record commit when in streaming mode.

General Recommendations

Avoid using the tape in streaming mode when the system is very busy. If other jobs are doing a lot of disk accesses, they may slow I/O down to the point where the drive must reposition.

Consider changing programs to write to disk files which are later backed up to tape.

Avoid using the BACKSPACE RECORDS control operation in programs. This is not a characteristic of all streamers, just the Tandem 5120.

When Using Tandem BACKUP/RESTORE Utilities

Use a large block size for BACKUP when possible. This applies to start/stop mode as well as streaming.

BACKUP on large files is better than on small files when in streaming mode. Experiment with start/stop and streaming mode to find which works better for your files.

Try volume mode backup. This feature of BACKUP/RESTORE does a physical backup of a disk which bypasses the file system. When run in streaming mode with a large block size, it is faster than file-by-file BACKUP.

Use VERIFYREEL rather than VERIFYTAPE, especially when in streaming mode.

Conclusion

Streaming drives offer the potential for high performance at low cost. They require some changes in the way tape is used to get maximum benefit. The advantages in cost and reliability will promote the use of streamers over conventional tape drives. Developers of new programs that use tape should consider how well their programs would run with a streaming tape drive.

References

Bashe, C.J., Johnson, L.R., Palmer, J.H., and Pugh, E.W. 1986. *IBM's Early Computers*. The MIT Press.

Harris, J.P., Phillips, W.B., Wells, J.F., and Winger, W.D. 1981. Innovations in the Design of Magnetic Tape Subsystems. *IBM Journal of Research and Development*. Vol. 25, No. 5.

Acknowledgments

The author would like to thank all the reviewers of this paper, particularly Charles Levine and Dan Watson.

John Blakkan has worked in magnetic tape hardware development since joining Tandem in 1984. He holds B.S. and M.S. degrees in Computer Science.

Tandem's new 5120 tape subsystem is used in the NonStop EXT10™ and NonStop EXT25™ computer systems. It features low cost, small size, and cartridge media with a storage capacity more than twice as large, and data integrity 100 times higher, than that of comparable open-reel tape drives. The subsystem (drive and formatter) connect to the tape controller via a small computer system interface (SCSI).

Although the 5120 uses the same basic ferric oxide media as conventional open-reel drives, increased storage density and reliability are achieved by application of state-of-the-art recording technology. This article discusses some of the techniques used:

- Basic operation.
- Serpentine format.
- Head design.
- Data formatting.
- Storage capacity.
- Error retry and recovery.

Basic Operation

The cartridge contains a spool with approximately 600 feet of tape. Cartridge insertion engages the spool with a motor hub. In the load operation, tape is threaded past the head and wrapped around a take-up reel in the drive until the beginning of tape (BOT) marker is found. This takes 75 seconds and allows the drive to measure the length of tape as well as providing a retensioning pass.

The drive operates as a reel-to-reel unit with motion controlled only by the two reel motors. (For more information on drive operation, refer to the accompanying article, "Streaming Tape Drives," by John Blakkan.) The drive is capable of both streaming operation and start/stop operation by means of a longer interblock gap (IBG) in start/stop mode. Data is recorded in MFM self-clocking code at 12,000 bits per inch (Ng, 1986).

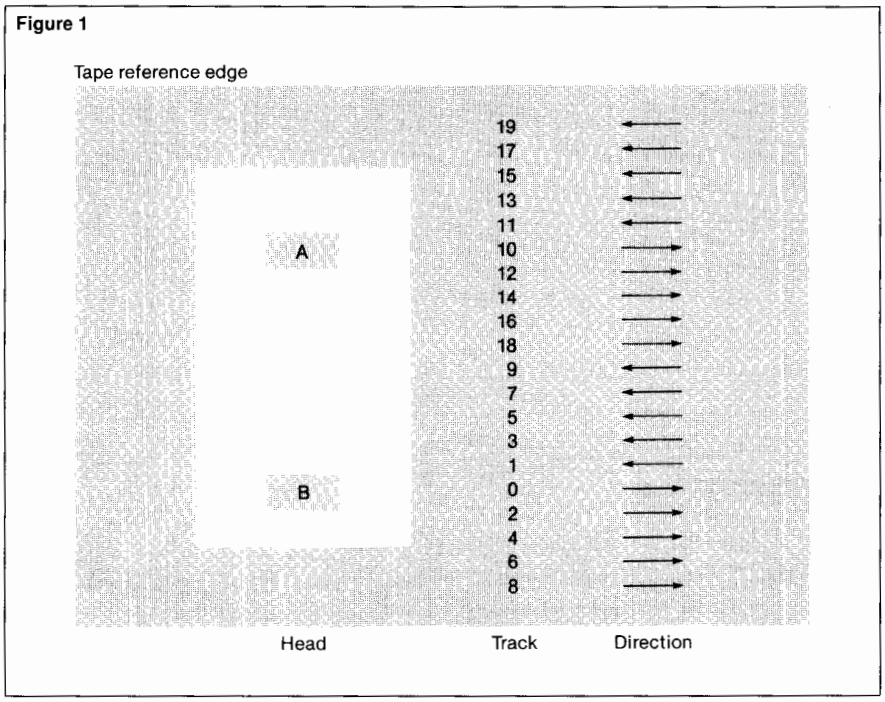


Figure 1.
Track layout.

Serpentine Format

Reducing the number of active data channels reduces the cost of a tape drive. Originally, Tandem tape products had nine active data channels (tracks) covering the full width of the tape. The 5120 subsystem has only two active data channels reducing the head cost as well as the amount of read and write circuitry. While a moving head and its associated stepping motor mechanism is needed to use the full tape width, the cost is minor compared to a fixed, multitrack head for the same number of tracks on tape.

Increased storage capacity results from the high linear recording density as well as the number of tracks across the width of the tape. The use of a moving head with appropriate track widths allows as many tracks as desired, subject to mechanical and electrical design trade-offs. The 5120 drive has ten head positions (20 physical tracks). The serpentine format makes the 570-foot recording area of the cartridge appear to be a 5700-foot two-channel tape.

The serpentine format is created by recording a tape from one end to the other. The head is then moved to a new track position, and the tape is recorded in the opposite direction. This is repeated until the tape is full.

The physical tracks 0 through 9 correspond to the "A" data channel and the physical tracks 10 through 19 are the "B" data channel. (See Figure 1.) Since channels A and B are used simultaneously, the drive is considered a ten-track, dual-channel device.

All recording starts on track 0 and proceeds sequentially to track 9. Recording starts near the center of the tape and proceeds outward towards the edges. This track layout was chosen to maximize the tape lifetime. The tape is positioned in relation to the head by guides that touch the edges of the tape. Therefore, using the tape wears the edges. When the edge wear affects the recording area of the outside tracks, the tape is considered worn out. In applications where the full cartridge capacity is not required, the outside tracks are not used and the effective cartridge lifetime is improved.

Head Design

The head configuration used in nine-track tape drives is an erase head that erases the full width of the tape, followed by a read/write head with a gap for writing and a gap for reading for each data channel. (See Figure 2.) This allows writing only when the tape is moving in one direction (forward) so that the read channel may verify that the data was written without error. Reading of data is also usually done in the forward direction.

The serpentine format requires writing with the tape moving in either direction. For good data integrity, the tape should be erased prior to writing, and a read gap should follow the write gap for error detection. Some cartridge drive manufacturers satisfy this requirement with a head configuration using a full-width erase head energized only when writing track 0, and two sets of gaps for each data channel—one set for each direction. One manufacturer of a widely used quarter-inch cartridge drive provides no erase and post-write check at all. A single-gap head is used for both write and read, relying on a fixed small record size and a 50% redundancy data format to recover read data.

The 5120 subsystem drive has a more elegant solution: the three-gap head. Each data channel has a write gap in the center of the head with a combined function read/erase gap on each side. (See Figure 3.) During forward tape motion the leading gap is driven to erase and the trailing gap reads. When the direction is reversed, the gap that was read now becomes erase and vice versa. The erase gap is degaussed whenever the erase current is turned off to prevent residual magnetism from biasing the read gap.

Data Formatting

All information recorded on tape may be categorized as either records or file marks. Nine-track drives use a unique track pattern and IBG size to differentiate a file mark from a record. The 5120 only reads one track at a time and so must use the information contained in the recorded format to identify records and file marks.

Figure 2

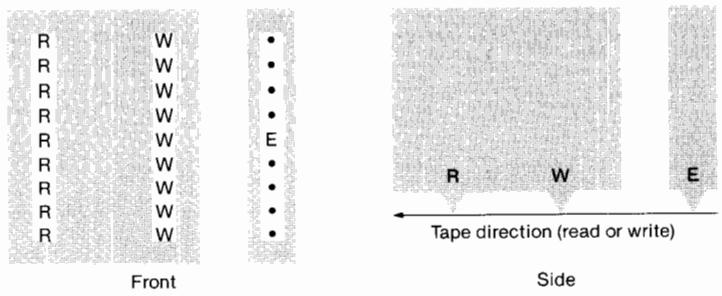
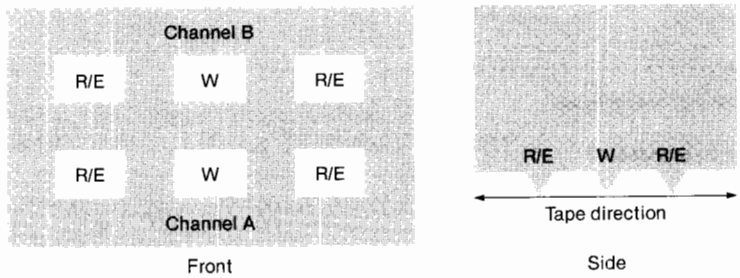


Figure 3



Data records are formatted into either one or two physical blocks. Records of 16,384 bytes or less fit in one physical block per data channel, while records between the ranges of 16,385 and 32,768 bytes require two physical blocks. Each physical block contains eight data sub-blocks and two error correcting code (ECC) sub-blocks that are variable in size. All data and ECC sub-blocks are the same size for a given record.

Figure 2.
Nine-track head.

Figure 3.
Three-gap head.

Table 1.
A comparison of tape recording densities.

	NRZI tape	PE tape	GCR tape	5120 tape	IBM 3480 tape	IBM 3380 disk
BPI	800	1600	6250	12,000	38,000	15,000
TP(1/2)I	9	9	9	20	18	400
Areal density (Kbytes/sq. in.)	14.4	18.8	112.5	240	684	12,000
Capacity/unit media (Mbytes)	20 (2400-foot reel)	40	160	130	200	1260

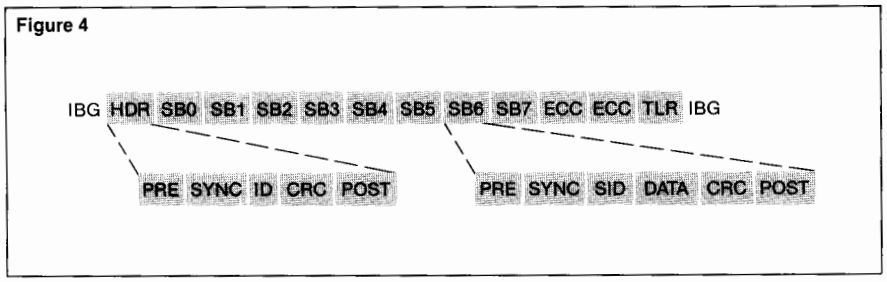


Figure 4.
Physical data block.

Within each physical block, data is divided equally between 16 sub-blocks, padded as required to fill all sub-blocks, and then unpacked between the two data channels on a sub-block basis. A single physical block for one data channel is shown in Figure 4.

The header (HDR) sub-block contains information on the size of the data and ECC sub-blocks, the record size, and the position of the record on the tape. It also indicates if the record required one or two physical blocks.

All sub-blocks contain cyclic redundancy check (CRC) bytes used for error detection when the sub-block is read.

The ECC sub-block data is the exclusive-OR of the data in four data sub-blocks on a byte-by-byte basis. The ECC is generated across the sub-blocks in data channels A and B as follows:

A ECC 0 = XOR (A0, B2, A4, B6)
A ECC 1 = XOR (A1, B3, A5, B7)
B ECC 0 = XOR (B0, A2, B4, A6)
B ECC 1 = XOR (B1, A3, B5, A7)

Spacing the ECC generation between separated sub-blocks decreases the possibility that tape defects such as debris particles could affect more than one sub-block involved in the ECC generation.

File marks look like data records with all the data and ECC sub-blocks removed. The HDR identifies the block as a file mark. To make the file mark more certain of recognition, the physical block is written twice.

Storage Capacity

The 5120 significantly improves recording density by increasing both linear density and the number of tracks across the tape. (See Table 1.) The recording density increase offers a potentially higher storage capacity for a given length of tape.

Tape storage capacity in the 5120 is highly dependent on the record size and the chosen operating mode. The IBG necessary to identify records reduces the amount of tape available for recording data. Therefore, the longer the record, the better the recording capacity. The IBG between records depends on the operating mode of the drive. Start/stop mode has an IBG of 4.5 inches while streaming mode has an IBG of 0.8 inch.

As an example, a 4-Kbyte record takes 2.1 inches of tape; a 16-Kbyte record takes 7.2 inches. In start/stop mode the 4-Kbyte recording has over twice as much gap as data. In the 16-Kbyte recording, the gap is less than 40% of the recorded area. Figures 5 and 6 show the effect of record size on capacity and effective transfer rate.

Error Retry and Recovery

Tape operates in a relatively harsh, uncontrolled environment compared to disk storage. Defects in the oxide surface as well as debris particles (such as dust) contaminate the media during use. All cause write and read errors, yet the drive must be able to tolerate a reasonable number of such errors with its retry procedures.

The error recovery scheme used in the 5120 tape subsystem relies on writing a record without error. This is assured when the regenerated CRC on data read from tape matches the CRC read from tape on each sub-block during write operations. During subsequent read operations, a data sub-block with error can be reconstructed by exclusive-ORing the ECC and associated data sub-blocks. Multiple data errors can be corrected as long as only one of the four sub-blocks for a given ECC has an error detected. This format and error recovery yields substantially improved error rates. The uncorrectable read error rates compared to open reel tapes are:

5120 = 1 in 10^12 bits
GCR = 1 in 10^11 bits
PE = 1 in 10^10 bits

Error retry sequences for either write or read errors are automatically performed by the drive/formatter. Up to 16 retrys are attempted before an uncorrectable error is reported.

Write error retry in start/stop mode is similar to that used in earlier tape products. The tape is spaced back over the record in error, then erased several inches in the forward direction, and then the record is rewritten. If there is still an error, the operation is repeated.

Figure 5

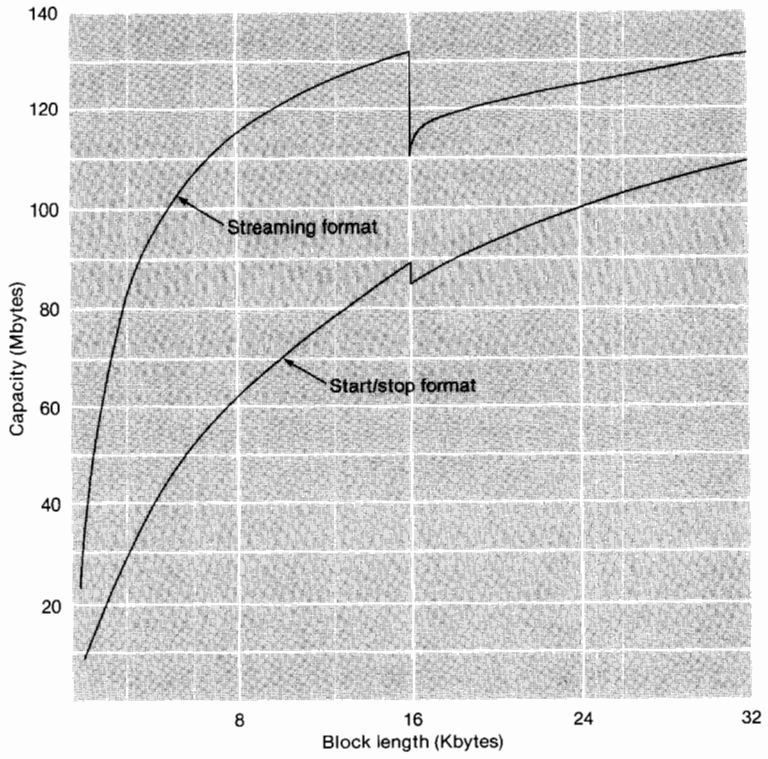


Figure 6

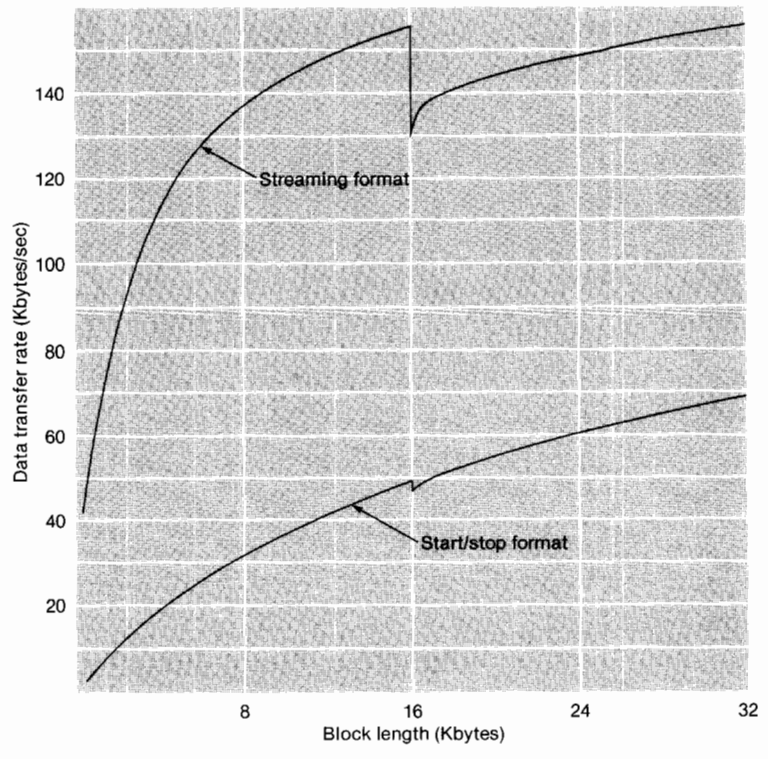


Figure 5. *Capacity vs. block size.*

Figure 6. *Transfer rate vs. block size.*

Write retry in streaming mode wastes tape instead of time. When a write error is detected, the record is rewritten immediately without erasing the defective copy of the record. This operation is repeated until no error is detected. With a maximum record size, 14 inches of tape are used for each retry (considerably more than are used in start/stop mode). The trade-off is speed, since a record can be written in less than 0.2 second; a reposition cycle to erase and rewrite takes more than 2 seconds.

In start/stop mode, read error recovery is attempted from the ECC data regeneration. If this is not successful, the tape is backspaced and the record reread with the ECC correction applied again.

Streaming mode read recovery also uses ECC regeneration. If this is not successful, the next record is read. If this record has the same block number as the record with error, this record is used for error recovery. If, however, the record has a higher block number, the tape is backspaced to a record with a block number less than the block in error, then spaced forward to the correct block number and the read operation repeated.

Conclusion

State-of-the-art recording technology is the key to the high capacity, high reliability, data integrity, and low cost of the 5120 cartridge tape subsystem. Tandem benefits from an improved price/performance, while the user benefits from improved storage capacity in a small, easily handled cartridge.

Reference

Ng, D. 1986. Data-Encoding Technology Used in the XL8 Storage Facility. *Tandem Systems Review*. Vol. 2, No. 2. Tandem Computers Incorporated. Part no. 83937.

Wesley Phillips has been with Tandem since 1984 and is currently a development engineer in the Peripherals department. He has over 30 years of engineering experience, 15 of them in tape controller and drive design and evaluation.

Ada: Tandem's Newest Compiler and Programming Environment

The Ada programming language was developed in response to several problems faced by any large systems user with multiple computer languages and systems to support:

- The difficulty and cost of training programmers in several languages, development environments, and application environments.
- The difficulty of porting a program written in a given language from one computer to another.
- The increased cost to support those languages.
- The problem of getting those languages to work together.

Ada was designed to handle many different programming tasks; a single language could now address requirements that previously required several languages. In addition, Ada was designed to be portable so that Ada programs could run on almost any computer, reducing software costs to the customer.

Evolution of Ada

In an environment where application software lasts from 10 to 20 years or more, the overall cost of the software becomes increasingly important. Initially, the primary emphasis in the development of computer languages was to improve the speed and efficiency of programmers in getting an application up quickly. Little analysis was done about the ongoing maintenance cost for the application software. If new hardware were to be installed for new applications and processing requirements, conversions and/or emulators were viewed as new costs, seldom tied to the life-cycle cost of the existing applications software.

These problems were especially true in the U.S. Department of Defense (DoD), where teams of hundreds of programmers write thousands of programs for a single application. To meet future needs for large scale real-time systems, the DoD needed a single high-level language that could be used for all military contracts.

In 1975 the DoD commissioned the High Order Language Working Group to investigate such a language. The group consisted of representatives from the military services, other DoD agencies, and several other countries. In 1977, after several years of formal study and review, the DoD announced that it would accept bids from private companies to develop an appropriate language.

In 1979 an international team from Cii Honeywell Bull in France developed the language now known as Ada.[1] The first standard for Ada was approved in December 1980 by the DoD and became MIL-STD-1815. In February 1983 the Ada Joint Program Office (AJPO) asked for standardization from ANSI (American National Standards Institute), which was granted, resulting in the current ANSI/MIL-STD-1815A.

Ada has a reputation for being expensive in terms of resources (CPU cycles, memory, and disk space) and challenging to learn in the sense that it encourages a different, more modern programming style. It was designed to reduce costs over the entire software life cycle, not necessarily the costs during just the coding phase.

Early problems included the scarcity of both trained people and validated compilers. In 1984 there were just 10 validated Ada compilers. By 1986 there were over 55 validated compilers. Currently, more than 20 companies offer education in Ada-related areas, and several major universities offer Ada programming classes. As the number of Ada compilers and products is increasing, the rate of introduction is accelerating, and the price/performance for the products is improving.

[1]The language was named after Augusta Ada Byron, a mathematician who worked with Charles Babbage and who is thought by some to be the first programmer.

Ada Application Areas

There are actually several arenas for the use of Ada within the DoD. When the DoD wanted a single language for all its contracts, that initially included what is known as "embedded systems." These are computers which are a part of larger systems, such as a guidance computer on a missile, a business communications network, or a microprocessor to control a VCR. The computer in an embedded system could be a processor such as the Intel 80286, the Motorola 68020, the military's 1750A, a Tandem NonStop system, or a network of Tandem NonStop systems. Embedded systems cover a wide range of applications, but they do tend to have similar requirements that match well with Tandem's strengths: parallel processing, real-time control, and high reliability.

Ada is also well suited for general applications and systems programming, and its use is now required in all DoD mission-critical systems.

Ada Compiler Validation

Ada is defined by a strict standard and has a large suite of test programs used to determine conformance to the standard. The name Ada is a trademark of the AJPO, and for a compiler to use the name "Ada," it must pass the test suite.

This test suite, part of the Ada Compiler Validation Capability (ACVC), contains approximately 2400 tests. (Certification for most other languages may require from 300 to 600 compiler tests.) The addition of new ACVC tests and the modification of existing ones is an ongoing process, and Tandem is committed to passing the ACVC test suite as each new version is released. In June 1987 a new test suite which included approximately 800 new tests became mandatory. Compiler validation expires one year after the certificate is issued. Each year, a compiler must pass the updated ACVC test suite in order to be issued another validation certificate. Of course, testing, no matter how rigorous, is no guarantee of correctness except in the most trivial situations. But one can be sure that a compiler passing all of the ACVC tests very closely conforms to the standard.

In addition to validation testing, a second factor eases portability and helps ensure consistency among different vendors' compilers: the AJPO will not allow a compiler for a subset or superset of Ada capability to use the Ada trademark. The subset prohibition means that something called an "Ada compiler" can process all Ada language features; the superset prohibition helps guarantee that a program compiled by one Ada compiler can be compiled by all Ada compilers.

Ada Programming Support Environment (APSE)

A compiler alone does not provide a complete programming environment. The DoD recognized this and sponsored the development of requirements for such an environment. While the APSE is not as completely defined as the Ada language, it is intended to support the development, maintenance, and enhancement of applications produced in Ada. It consists primarily of "tools" needed for a large development effort with a long software life cycle. Typical APSE tools are the compiler, linker or binder, text editor, loaders, configuration managers, a program database or library manager, and debugging facilities. According to the requirements, most of these tools should be written in Ada.

For Tandem Ada, some of the APSE tools are provided as a part of the product itself (the compiler, binder, and program library manager), while other components are provided by other Tandem products (e.g., a text editor and symbolic debugger).

Ada Language Features

Ada, like Pascal, is a structured language with strong type checking. However, Ada has more features than Pascal. Some of the major features are packages, tasks, and generic units; these are discussed in this section. The source list at right contains more information on Ada features as well as requirements, standards, and programming.

Packages

The Ada package feature allows a programmer to group together various logically related entities, for example, related type declarations, variables, and subprograms. Packages are divided into two parts: the specification and the body. The specification defines the interface that a user of the package has to the facilities provided by the package. The body defines the implementation of the package; the details appearing in the body cannot be used by the user of the package. The package feature supports modularity, data abstraction, and information hiding.

Source List*

Information on Tandem Ada products:

Contact your local Tandem office or
Tandem Computers
Marketing Services
19191 Vallco Parkway, LOC 4-31
Cupertino, CA 95014
(408) 725-6000

Information on Ada requirements and standards:

Contact:
Ada Information Clearinghouse
3D139(1211fern, C-107)
The Pentagon
Washington, D.C. 20301-3081
(703) 685-1477

Information about programming in Ada:

The following list is a sample of the types of books available on the subject of Ada.*

Comparing & Assessing Programming Languages ADA C PASCAL
Feuer & Gehani
Prentice Hall Inc.
Engelwood Cliffs, NJ 07632

Software Engineering with Ada
(Second Edition)
Grady Booch
Benjamin/Cummings Publishing Company
2727 Sand Hill Road
Menlo Park, CA 94025

*Tandem makes no recommendation for, or endorsement of, any of these books, nor does Tandem intend to imply anything by the absence of any book from this list.

Figure 1

```
-- Ada comments begin with a "--" and end at the end of the line.
-- This is the specification of a package defining a stack data
-- structure. The stack contains integer values. A user of this
-- package can only make use of entities declared in the
-- specification. In this case, the only things a user has access to
-- are three stack operations. Note that in this simple example, the
-- stack contains integers and there is only one stack. Ada has a
-- feature called "generic units" which could be used to allow the
-- user to customize the type of the stack elements. There are also
-- ways to define packages which would allow multiple instances of
-- stacks.

package Stack is
   procedure Push (Pushval : Integer);      -- Push a value onto the stack.
   function Pop return Integer;             -- Return and pop off the top
                                            -- of the stack.
   function Tos return Integer;             -- Return the top of the stack,
                                            -- without popping it off.
end Stack;

-- This is the body of the stack package. Implementation details in
-- the body do not directly affect a user of the package. In this
-- body the stack is implemented with an array. However, a new body
-- implementing the stack with a linked list could replace this array
-- version (to allow a dynamically sized stack rather than a fixed
-- sized stack). The user of this package would not be affected and
-- need not recompile or modify any code since the specification
-- remains unchanged. The implementation of the stack is hidden from
-- the user. Note that in this simple example, error conditions
-- (e.g., stack overflow) are not handled. Ada has a built-in feature
-- called "exceptions" which makes such error processing convenient.

package body Stack is

   -- A constant defining the size of the stack.
   Stack_Size : constant Integer := 4;

   -- The stack pointer, the index of the top element of the stack in the
   -- array. The value is initialized to zero, which denotes an empty
   -- stack. Note the value must remain between zero and the stack size.
   Sp: Integer range 0..Stack_Size := 0;

   -- The array holding the stack elements.
   Stack_Object : array (1..Stack_Size) of Integer;

   procedure Push (Pushval : Integer) is
   begin
      Sp := Sp + 1;
      Stack_Object (Sp) := Pushval;
   end Push;

   function Pop return Integer is
      Popval : Integer;
   begin
      Popval := Stack_Object (Sp);
      Sp := Sp - 1;
      return Popval;
   end Pop;

   function Tos return Integer is
   begin
      return Stack_Object (Sp);
   end Tos;

end Stack;
```

Figure 1.
Example of a package implementation of a stack data structure.

The separation of the specification and body is especially useful in large software projects. If the various package specifications are defined early in the project, programmers can write the package bodies independently. Since a user's interface to a package is completely defined by the specification, changes to the body do not affect any users. Figure 1 is an example of a simple stack package that might be used by Ada programmers.

Tasks

Tasks are Ada entities that execute logically in parallel. Tasks can be used for processing interrupts, routing messages, and controlling resources. On a Tandem system supporting Ada, tasking is a way of specifying threads of concurrency within the language for a single (GUARDIAN 90) process.

Generic Units

A generic unit defines a template for a subprogram or package. By "instantiating" a generic unit, a specific instance is created. By "parameterizing" the generic unit, the created instances can be customized. For example, suppose the Ada application needs two linked lists, one of integers and another of floating point numbers. Rather than writing two linked list packages, only one generic linked list is needed. This generic unit would be parameterized so that the type of elements of the list is given at the time of instantiation. Having written this one generic unit, it can be instantiated twice, once for integers and again for floating point numbers. The generic unit feature is particularly useful for writing sharable, reusable code.

Tandem's Ada Products

Tandem has worked with Ada since 1980. Development efforts for Tandem's production quality Ada compiler for NonStop systems began in late 1984. At about the same time, the DoD, probably the largest single computer market, began emphasizing Ada as a language requirement for use in new hardware appropriations. By late 1986, a company without Ada on its system could not get a DoD contract without a special waiver.

The current Tandem development team consists of nine people with more than 35 years of Ada experience. The Tandem Ada Compiler, Ada Library Manager, Ada Binder, and a large part of the Ada-specific portions of the symbolic debugger are all written in Ada. Quality assurance for the Ada product was begun in 1985, so that it could be performed synchronously with development. This reduces regressive problems with the product and expedites the quality assurance cycle.

Tandem's Ada compiler conforms to both the ANSI/MIL-STD-1815A specification and OSI/8652-1987 Programming Languages-Ada. It requires the floating-point microcode option and is capable of supporting programs of up to 4 Mbytes of code space and up to 128 Mbytes of data space. It is available with the C00 software release on NonStop systems.

In addition to the packages defined and described in MIL-STD-1815A (e.g., TEXT_IO, SEQUENTIAL_IO), Tandem provides an additional predefined package with its Ada product. The COMMAND_INTERPRETER_INTERFACE permits access to start-up message information such as the in-file and out-file, parameters, and ASSIGNs and PARAMs. It is via this interface that Ada programmers access the input from the command interpreter. Tandem Ada can also call TAL routines to perform other system functions the programmer desires.

Libraries

Ada shifts responsibility for many "bookkeeping" functions from the programmer to the compiler and the Ada binder. Ada compilation units are compiled into a "library" that records information about compilations. While a mechanism such as this is required to support the separate compilation of multiple compilation units, it requires processing overhead to update all of the necessary information.

The ADALMGR command is used to create a library. The ADA command is used to compile source files into libraries. The user specifies the name of the "primary" library into which the files are to be compiled and the names of other libraries containing units that might be needed by the compilation.

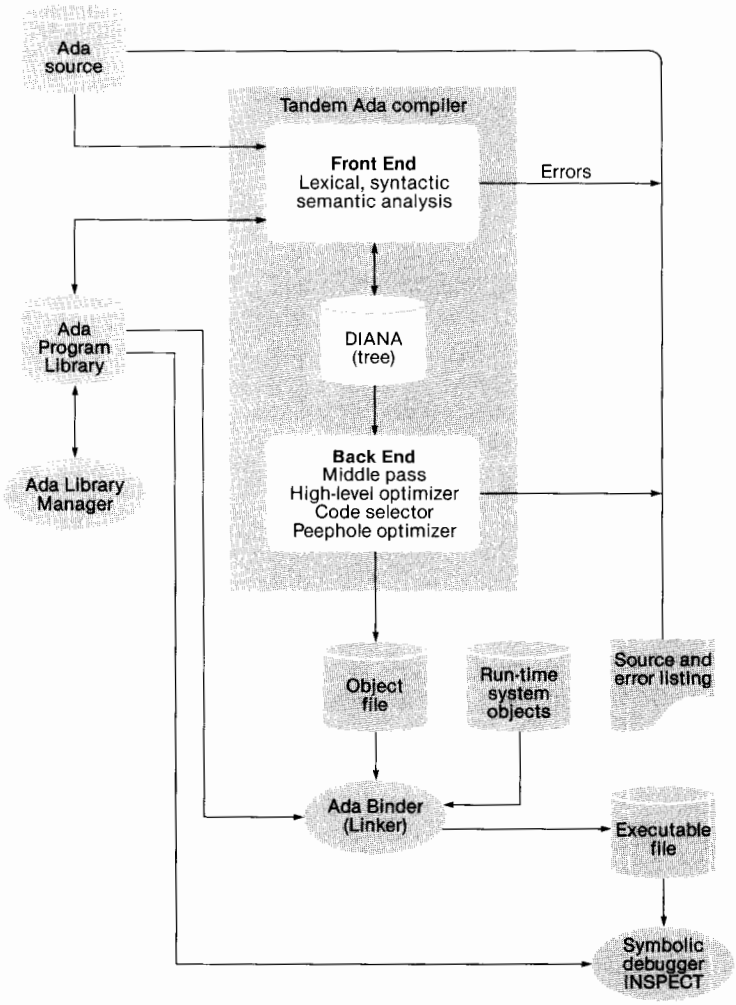
After units have been compiled into libraries, the ADALMGR command can be used to find out information about these units or to delete them from the libraries. The ADABIND command is used to create an executable object program after all the compilation units that it needs have been compiled. The user tells ADABIND the name of the main program and the library in which it resides.

DIANA

Most Ada implementations use an intermediate language to represent the source code along with additional information gathered during semantic processing. Tandem's compiler uses a de facto standard language called Descriptive Intermediate Attributed Notation for Ada, or DIANA. DIANA files are produced by the front end of the compiler and contain all the essential syntactic and semantic information for a given program unit.

Since DIANA is well defined, various tools can use a DIANA representation of a program as input. For example, the symbolic debugger provided with Tandem Ada obtains information about symbol names from the DIANA intermediate form.

Figure 2



In this case, the command line
invokes the Tandem Ada compiler signified by
>Ada /in source_filename, out listing_filename/library_filename

Figure 2.

Tandem Ada programming environment. The Ada development environment is complex. Because of the number of files accessed and the activities taking place, Ada requires more resources and more time to compile code than most other language compilers.

An example of this implementation on a Tandem system is shown in Figure 2. Note that the Ada compiler and programming environment are more complex and resource-intensive than those of other languages. This is because the Ada language is more complex and requires more processing, the Ada compiler is large and requires more main memory to run efficiently, and Ada requires additional disk space for the information and files maintained by the library.

Conclusion

Ada is one of the most rigorously defined ANSI-standard languages. By requirement, it has a support environment that enables easy maintenance of the software produced by the compiler. It is ideal for applications which will have long life cycles. With Ada, most of the application life cycle costs tend to occur early and decline as the application ages.

Ada is a very young language. As more and better formal Ada education (university level) grows, more validated Ada compilers of adequate performance become available, and the user community gains a better understanding of the Ada language and environment, Ada will surely become more widely used.

References

ANSI Reference Manual for the Ada Programming Language. ANSI/MIL STD-1815A-1983. U.S. Government—Ada Joint Program Office. (This manual is available from Tandem Computers Incorporated, part no. 84063.)

NonStop Systems Ada User's Guide. Tandem Computers Incorporated. Part no. 82523.

Acknowledgments

In addition to the Ada product developers and manual writers, the author would like to thank Dick Thomas for his advice on structure and content, and Jim Meyerson and Katie Pepper for their technical input. Special thanks to Paul Delvigna and David M. Ng for all of the above.

Richard Vnuk joined Tandem in 1980 and is currently the product manager for Languages and Tools. He has also held such positions as regional education specialist, division MIS manager, and senior systems analyst in the Performance Support Group. Prior to joining Tandem he was a programmer and systems analyst for end users, developing and designing large databases for on-line transaction processing. He also worked for a mainframe vendor supporting marketing, installing, converting, and designing large databases for OLTP functions.

Controllers built by Tandem interface purchased disk and tape drives to the Tandem I/O channel. Interfaces provided by drive manufacturers are becoming more complex as recently designed drives incorporate functions previously performed by the controller. This article discusses several types of disk and tape interface and explains why one type might be preferred.

Input/output Subsystems

Disk and tape drives move a magnetically coated medium past electromagnetic read and write heads. The write head applies to the coating a pattern of magnetization that can be detected as a sequence of electric pulses in the read head. Some combination of hardware and software must convert computer data bytes to head pulses and back again for a disk or tape drive to be used for data storage.

Drive operation must also be controlled in response to requests from the computer's operating system. Disk heads must seek the proper track for file access, and tape drives must rewind or otherwise move the tape to locate user data.

Most large computer systems use input/output subsystems to perform these data transfer and drive control functions. Figure 1 shows an I/O subsystem for disk or tape. The subsystem is shown as a series of layers between the drive and the I/O channel. The layers correspond to portions of the data transfer and device control functions. The subsystem may be divided at any layer boundary. The layers on the I/O channel side of the division constitute the *I/O controller*. The remaining layers, together with the mechanical disk or tape drive, comprise the *I/O device*.

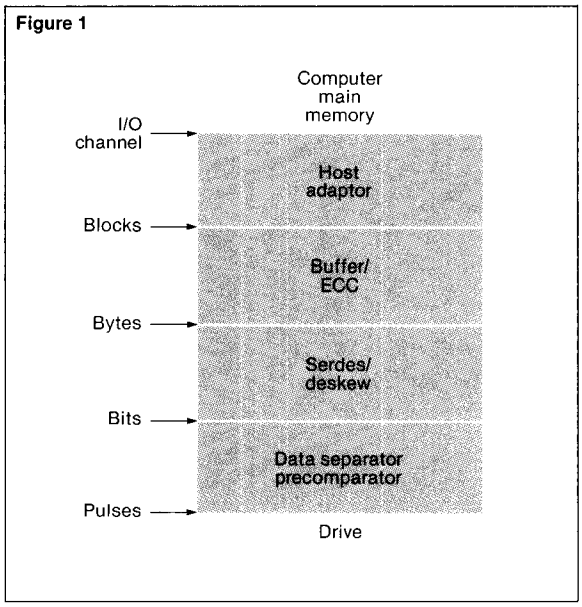


Figure 1.
Model of I/O subsystem with computer and drive. The names of the interfaces between the layers of the subsystem correspond to the form in which information is transferred.

Figure 2.
Example of division of I/O subsystem between a Tandem 3108 disk controller and device.

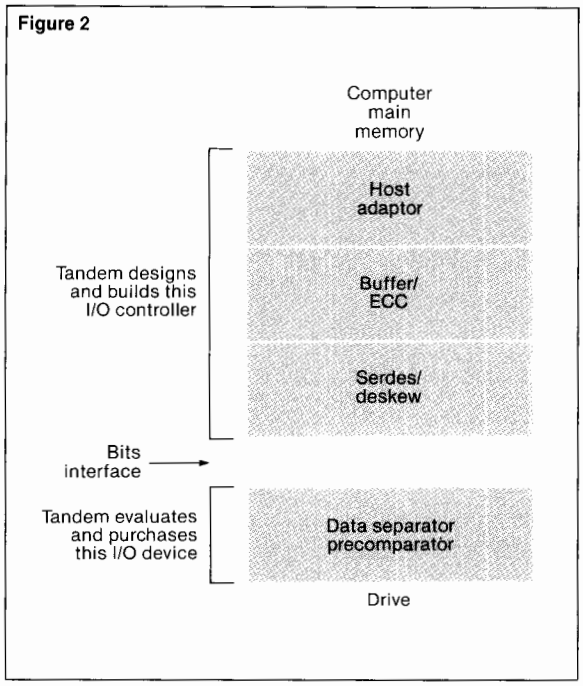
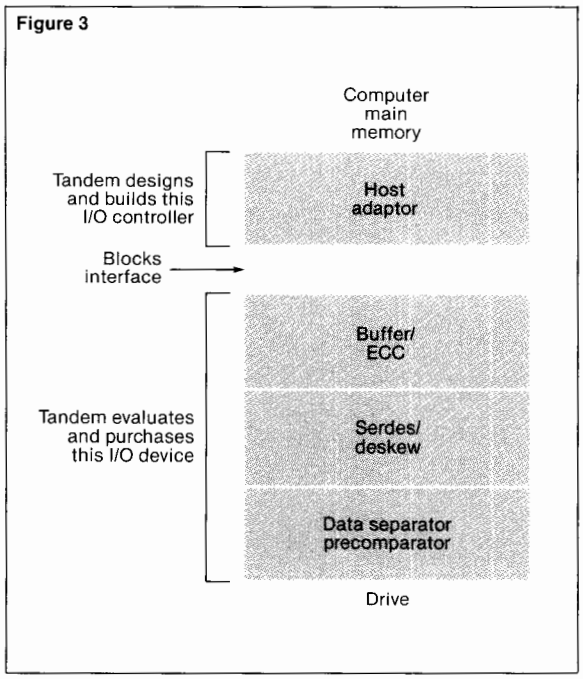


Figure 3.
Example of division of I/O subsystem between a Tandem 3209 cartridge tape controller and device.



A computer system manufacturer could purchase a device with a *bits* interface and build a controller providing the host adaptor, buffer/ECC, and serdes/deskew functions.[1] The Tandem 3108 disk controller shown in Figure 2 is an example of this configuration. Alternatively, a manufacturer can purchase a device with a *blocks* interface and build a controller which provides only the host adaptor function; e.g., a Tandem 3209 cartridge tape controller. (See Figure 3.) The choice of subsystem layers to build into an I/O controller and what type of peripheral device interface to purchase affects I/O system performance, maintenance policies, and the time required to integrate a peripheral into a computer system.

This article identifies and describes each of the layers shown in Figure 1. It also describes the interfaces between these layers and discusses the effect of building the layers into an I/O controller as opposed to buying them from a device manufacturer.

Function of the Layers of an I/O Subsystem

To understand I/O subsystem interfaces it is first necessary to understand the function of the drive and of each layer in the subsystem.

Magnetic Transitions for Information Storage

The magnetic coating on computer disks and tape may be magnetized in one of two directions. The direction of magnetization is not detected with a read head in high-density digital recording. Instead, a *reversal* in the direction of magnetization is detected; it induces an electrical pulse when passing the read head. (See Figure 4a.)

Usually a flip-flop circuit drives the write head. Each pulse input to the flip-flop reverses the direction of current flow in the write head's coil. When this occurs, the direction of magnetization in the magnetic coating is also reversed. (See Figure 4b.) Thus, when reading and writing the medium, an electric pulse corresponds to a reversal in the direction of magnetization; the absence of a pulse corresponds to no reversal.

Magnetization of disk media was discussed more fully in a previous issue of the *Tandem Systems Review* (Ng, 1986).

[1] A disk serializer and deserializer together are often called a "serdes." Error correcting code (ECC) and deskew are discussed later in this article.

Write Compensation

Write compensation improves the magnetization patterns written, increasing the precision with which magnetization reversals (and therefore the corresponding pulses) can be located on the medium. This also increases the density at which data may be recorded. Forms of write compensation are used on both disk and tape.

There are two common write-compensation techniques: reduced write current precompensation is performed in the drive; timing precompensation is handled in the data encoding section of the I/O subsystem.

Reduced Write Current. Usually it is possible to write the same amount of information on a track near the center of a disk as on a track near the outer edge. The inside track is shorter than the outside track, which means that the magnetic transitions must be physically closer together. Adjacent pulses will interfere with each other unless the write current in the head is reduced, applying less magnetization to the inside tracks. Small Winchester disks have only two values of write current: standard and reduced. Larger disks may have many write current values.

Some high-density tape drives write a test pattern on the leader of each tape and use it to adjust the value of the write current, permitting the drive to optimize the magnetization of each tape. This is necessary because tape media are produced by several manufacturers and have different magnetic properties.

Timing Precompensation. Timing precompensation is a fine tuning of magnetic transitions to accommodate the "peak-shift" phenomenon (Ng, 1986). If the series of pulses shown in Figure 5a is written to a disk or tape drive and then read back, peak-shift will move the fourth pulse to the right. (See Figure 5b.) Timing precompensation moves pulses in the opposite direction of anticipated peak-shift before the pulses are written to the drive. (See Figure 5c.) When the precompensated pulses are read, peak-shift will move the fourth pulse to its proper position. (See Figure 5d.)

Figure 4

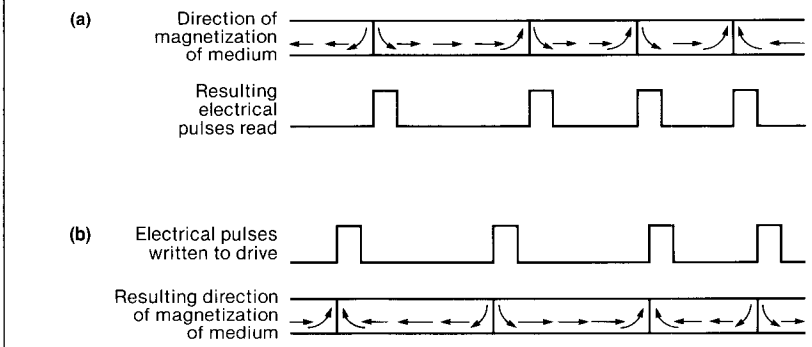


Figure 5

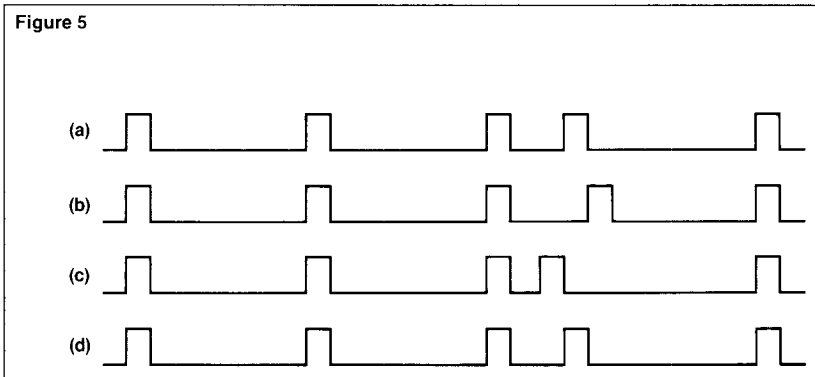


Figure 4. *(a) Pulses corresponding to magnetic reversals when read. (b) Magnetic reversals corresponding to pulses when written.*

Figure 5. *(a) Uncompensated series of pulses. (b) Series of pulses showing effect of peak-shift. (c) Pulses after precompensation. (d) Precompensated pulses showing effect of peak-shift.*

Figure 6

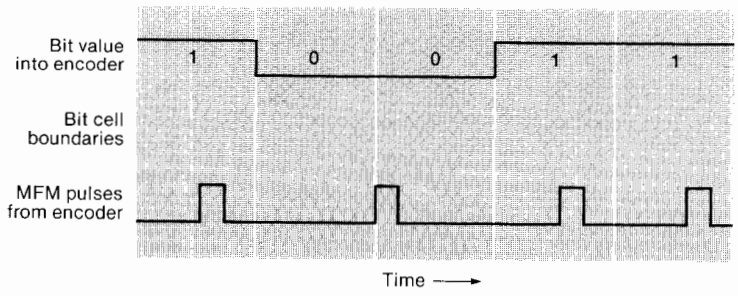


Figure 7

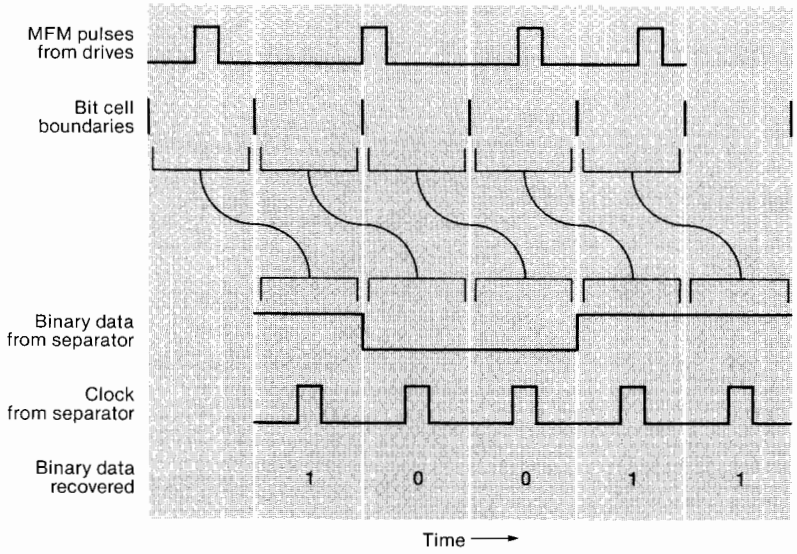


Figure 6.
Example of MFM data encoding.

Figure 7.
Example of MFM data separation.

Data Encoding and Separation

Each bit of digital information is contained in a small piece of the magnetic medium. Because the magnetic medium moves at a constant speed over the head, these fixed-length segments of the medium correspond to fixed amounts of time. The space taken up by a bit on the medium and the time it takes to write it are called a "bit cell."

Figure 6 illustrates one system of data encoding called Modified Frequency Modulation (MFM). The rules for turning binary data into pulses in bit cells are simple:

- A binary zero is translated into a pulse at the beginning of the bit cell.
- A binary one is translated into a pulse in the middle of the bit cell.
- Any time a binary zero follows a binary one, the pulse for that binary zero is omitted.

Data separation is the inverse of data encoding. A string of pulses is transformed into two separate signals, one representing data and the other a data clock. The data separator uses a phase-locked oscillator to determine where the bit-cell boundaries are relative to the pulses.

Figure 7 shows an example of MFM data separation. In this case, the rules are as follows:

- Any bit cell with a pulse in the middle corresponds to a binary one.
- Any bit cell with a pulse at the beginning corresponds to a binary zero.
- Any bit cell with no pulse, immediately following a bit cell containing a binary one, corresponds to a binary zero.

There are other systems of data encoding, such as RLL 2-7 for disk, and PE and GCR for tape.[2] Details about data encoding and separation for disk drives were described in a previous issue of the *Tandem Systems Review* (Ng, 1986).

[2]RLL 2-7 = Run Length Limited 2-7; PE = Phase Encoded; GCR = Group Code Recording.

Serialization and Deserialization (Serdes)
Usually, a computer system transfers data internally as bytes or words. Manipulation of streams of bits to the data encoder and from the data separator is performed by the I/O subsystem.

In a disk subsystem, a byte from the computer is converted by a serializer into a string of bits before being written. (See Figure 8.) In a tape subsystem, the bytes are divided into eight parallel bit streams, which are written with a ninth parity track simultaneously on the tape.

When a string of bits is read from a disk data separator, a deserializer converts groups of bits into bytes. (See Figure 9.) A nine-track tape subsystem has a functional unit analogous to a deserializer called a deskew buffer, which synchronizes the nine bit trains from the nine data separators and assembles them into bytes. The buffer is needed because tape head misalignment interferes with the pulses from each track differently: the nine bit cells for each byte are not read simultaneously as the tape moves over the head.

Error Detection and Correction
Error detection and correction are performed in the I/O subsystem by adding error correcting code (ECC) information to the data when it is written on the medium, and checking the data and ECC information for validity when the medium is read.

The subsystem uses error detection to determine whether or not information read from the medium has been damaged. The subsystem then deduces the correct data values from damaged data and ECC information.

Disk subsystems usually try to prevent errors rather than correct them routinely. When a disk is formatted, it is also tested for errors. Any sectors or tracks which cannot be written to and read from without error are marked as unusable. Tape subsystems also avoid using bad portions of the media. Tapes are read and checked for errors as they are written. Records which show errors are erased and rewritten farther down the tape.

Figure 8

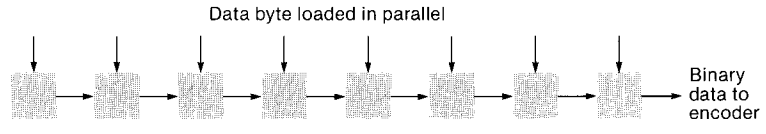
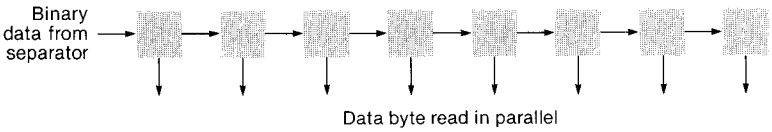


Figure 9



A disk subsystem detecting a read error may attempt to reread the data, or may attempt to use the ECC to correct the erroneous data. Frequently, disk errors do not recur if the read operation is retried. Disk subsystems may retry before attempting to correct with the ECC, or they may use the ECC first and reread only if this fails to correct the error. Tape subsystems also use a combination of read retries and ECC. For example, when the Tandem 5120 cartridge tape device finds a read error, it backspaces and attempts to reread up to 15 times, attempting correction with ECC each time.

Figure 8.
Disk serializer.

Figure 9.
Disk deserializer.

There are many types of error detecting and error correcting codes, with certain basic differences:

- *Detection capability*, the number of erroneous bits in a disk sector or tape record that can be detected.
- *Correction capability*, the number of erroneous bits in a disk sector or tape record that can be corrected.
- *Code length*, the number of bits of ECC information added to the recorded data.
- *Generation time*, the length of time required to generate the correction information added to the data when recorded.
- *Correction time*, the length of time required to correct an error or determine that an error exceeds the correction capability of the code.
- *Probability of miscorrection*, the probability of miscorrecting an error exceeding the detection capability of the code and falsely reporting successful correction.

Data Buffering

The data buffer is used to match the data rate of the disk or tape drive to that of the computer's I/O channel, and to provide a place to process the data and ECC information if error correction is necessary. Some high-performance disk subsystems also use the data buffer as a disk data cache (Grossman, 1985), and streaming tape subsystems use a data buffer to facilitate streaming mode operation. (See the accompanying article, "Streaming Tape Drives.")

Host Interface

The host interface is the I/O subsystem's connection to the computer's I/O channel or system bus. Tandem's host interface is unique in that it connects to two channels for fault tolerance.

Peripheral Device Interfaces

Four types of interface between the layers in the I/O subsystem model are used as device-level interfaces. These are compared in Table 1; their relationship is shown in Figure 1.

Table 1.
A brief comparison of device interfaces.

Interface	Advantages	Disadvantages
Pulses	Low device cost	Short cables
		Controller must have data separator
Bits	No data separator in controller	Short cables
Bytes	Long cables	Fixed byte length
Blocks	Long cables	High cost
	Ease of integration	Loss of control to device manufacturer

Pulse Interface

The pulse interface is also called the "bare drive interface." Compared to other interfaces, it provides the least functionality in the disk or tape device.

The data interface is a stream of digital pulses. The controller must generate encoded pulses and perform timing precompensation when writing. The controller may also be responsible for signaling the drive when to use reduced write current. A data encoder and data separator must be built into the controller.

The device control interface is very simple. Typically, there are individual signals which are pulsed by the controller to perform functions such as track seek on a disk drive or rewind tape on a tape drive.

This is the least expensive interface and is frequently chosen for high-volume, cost-sensitive applications (e.g., floppy disk drives in personal computers). Data separators are difficult to design, and they limit the device data rate. Higher data rate devices cannot be used with the controller unless the data separator is redesigned. Therefore, the trend among drive manufacturers and computer system manufacturers is to use a bits interface rather than a pulses interface except for the most cost-sensitive applications.

The ST-506 interface used on personal computer Winchester disk drives and the device interface in the Tandem 3207/5104 tape subsystem when in 1600 bpi mode are two examples of a pulse interface.

Bits Interface

This type of interface puts the data encoding and separation function into the drive and uses data bits rather than pulses. When recording the data on the magnetic media, the device implements some type of encoding system (such as MFM or RLL 2-7 for disk and PE, GCR, or MFM for tape). The interface provides the controller with a binary data signal and a clock signal during read operations. It provides a write clock signal and requires binary data during write operations.

The device manufacturer selects the pattern of magnetization corresponding to the binary values of the bits, but does not control the grouping of data bits into bytes or words. The controller performs the disk serdes or tape deskew function. Precompensation is handled entirely by the disk or tape device.

The bits device control interface may be different from that of the pulse device. Most floppy disk devices with a pulse interface require a separate command to move the head from a track to the next adjacent track. A controller must remember the head's position and generate an appropriate number of track seek commands when the computer requests a seek to a particular track. Most small Winchester disk devices with bits interfaces accept commands for multitrack head repositioning. The controller needn't remember the head's position.

Examples of this type of interface are the enhanced small device interface (ESDI), which is used between the Tandem 5120 tape drive and formatter, and the storage module drive (SMD) interface used in the Tandem 3107/4120 disk subsystem.

Bytes Interface

A disk or tape device with a bytes interface transfers data to and from the controller as bytes. Disk devices have a serdes, and tape devices have a deskew buffer. The control interface may be as simple as a pulse control interface, with commands sent over individual signal lines, or the commands may be encoded as a binary number on several signal lines.

A bytes interface has several advantages over a bits interface:

- It uses lower frequency signals to transmit data. For example, an 8-Mbit-per-second serial data stream becomes eight 1-Mbit-per-second parallel data streams.
- Lower data rate signals permit lower cost cables and transceiver circuits (e.g., twisted-pair rather than coaxial cable).
- Frequently the same signal lines carry control information and data. In this case, the bytes interface requires fewer signal lines than a bits interface.
- Some bytes interfaces permit considerably longer cables between controller and device than many bits interfaces.

A bytes interface allows less flexibility as to how a disk is formatted. Typically, the manufacturer defines a byte length. Fields in disk sectors must be byte-aligned, which may cause difficulties for computer systems with a different byte length. (Most disks and tapes with byte interfaces have an 8-bit byte length, while some computer systems have a 6-bit byte length).

When operated in 800 bpi mode, Tandem's 3207/5104 tape subsystem uses a bytes interface with an individual command line control interface. A more complex control interface using five signal lines to encode up to 32 commands is available on several manufacturers' streaming tape drives.

Blocks Interface

A disk or tape device with a blocks interface performs many of the functions traditionally provided by the controller. It presents a view of "perfect" blocks of data; this means that error detection, error correction, media defect management, and data buffering are all handled by the device. The device control interface is typically time-multiplexed on the same signal lines as the data path. Commands and data are distinguished by interface tag signals, or by interface sequence definitions. Command packets and responses are defined. Devices may perform sophisticated operations (such as device to device copy) under their own control.

The advantage for a system manufacturer is that controllers are easier to design; a controller designed for a disk drive may also work with a tape drive.

The disadvantage is that the drive manufacturer has already made many system design choices. For example, many of the software copy protection techniques used on floppy disks are not possible in a device with a blocks interface. Also, the device contains a microcomputer programmed by the device manufacturer rather than the system manufacturer. Errors or undocumented features in this program must be discovered and accommodated by the system manufacturer who is unable to change the program.

Performance optimization may also be difficult since the device incorporates some of the functions that the system manufacturer may wish to handle in the operating system (e.g., disk-seek scheduling).

Finally, field diagnostic techniques are largely determined by the drive manufacturer. If sufficient diagnostic and self-test functions are not provided by the drive, field diagnosis may be difficult.

The small computer system interface (SCSI) used in the Tandem 3209/5120 cartridge tape system is one example of a blocks interface.

Conclusion

The main considerations in choosing the interface between a peripheral device and an I/O controller are:

- Quality and reliability of the resulting I/O subsystem.
- Cost and availability of peripheral devices with the various types of interface.
- Ease of integration into the computer system.

Pulse interfaces will continue to be used in low-cost, low-performance, high-volume products such as 3.5-inch floppy disks on personal computers. The trend in the mid-range and high-performance I/O subsystems is toward devices with bytes or blocks interfaces.

References

Ng, D.S. 1986. Plated Media Technology Used in the XL8 Storage Facility. *Tandem Systems Review*. Vol. 2, No. 2. Tandem Computers Incorporated. Part no. 83937.

Ng, D.S. 1986. Data-encoding Technology Used in the XL8 Storage Facility. *Tandem Systems Review*. Vol. 2, No. 2. Tandem Computers Incorporated. Part no. 83937.

Grossman, C.P. 1985. Cache-DASD Storage Design for Improving System Performance. *IBM Systems Journal*. Vol. 24, Nos. 3/4.

John Blakkan wrote this article, as well as "Streaming Tape Drives."

Tandem Systems Review Index

August 1987

The *Tandem Journal* became the *Tandem Systems Review* in February 1985. Four issues of the *Tandem Journal* were published:

Volume 1, number 1	Fall 1983	Part no. 83930
Volume 2, number 1	Winter 1984	Part no. 83931
Volume 2, number 2	Spring 1984	Part no. 83932
Volume 2, number 3	Summer 1984	Part no. 83933

As of August 1987, seven issues of the *Tandem Systems Review* have been published:[1]

Volume 1, number 1	February 1985	Part no. 83934*
Volume 1, number 2	June 1985	Part no. 83935*
Volume 2, number 1	February 1986	Part no. 83936*
Volume 2, number 2	June 1986	Part no. 83937**
Volume 2, number 3	December 1986	Part no. 83938
Volume 3, number 1	March 1987	Part no. 83939
Volume 3, number 2	August 1987	Part no. 83940

The articles published in all eleven issues are arranged by subject below. (*Tandem Journal* is abbreviated as TJ and *Tandem Systems Review* as TSR.) A second index, arranged by product, is also provided.

Index by Subject

Article title	Author(s)	Publication	Volume, Issue	Season or Month and Year	Part Number
Application Development and Languages					
Ada: Tandem's Newest Compiler and Programming Environment	R. Vnuk	TSR	3,2	Aug. 1987	83940
An Introduction to Tandem EXTENDED BASIC	J. Meyerson	TJ	2,2	Spring 1984	83932
State-of-the-art C Compiler**	E. Kit	TSR	2,2	June 1986	83937
Tandem's New COBOL85*	D. Nelson	TSR	2,1	Feb. 1986	83936
The ENABLE Program Generator for Multifile Applications*	B. Chapman, J. Zimmerman	TSR	1,1	Feb. 1985	83934
PATHFINDER—An Aid for Application Development	S. Benett	TJ	1,1	Fall 1983	83930
A New Design for the PATHWAY TCP	R. Wong	TJ	2,2	Spring 1984	83932
PATHWAY IDS: A Message-level Interface to Devices and Processes**	M. Anderton, M. Noonan	TSR	2,2	June 1986	83937
TACL, Tandem's New Extensible Command Language*	J. Campbell, R. Glascock	TSR	2,1	Feb. 1986	83936
New TAL Features**	C. Lu, J. Murayama	TSR	2,2	June 1986	83837
TMF and the Multi-Threaded Requester	T. Lemberger	TJ	1,1	Fall 1983	83930
Writing a Command Interpreter*	D. Wong	TSR	1,2	June 1985	83935
Customer Support					
Customer Information Service	J. Massucco	TSR	3,1	March 1987	83939
Remote Support Strategy	J. Eddy	TSR	3,1	March 1987	83939
Tandem's Software Support Plan	R. Baker, D. McEvoy	TSR	3,1	March 1987	83939

[1]Articles and issues indicated by an asterisk (*) are no longer in stock. Preliminary versions of these articles are available on the CIS (Customer Information Service) system. Issues indicated by a double asterisk (**) are available in limited quantities.

Article title	Author(s)	Publication	Volume, Issue	Season or Month and Year	Part Number
Data Communications					
Changes in FOX*	N. Donde	TSR	1,2	June 1985	83935
A SNAX Passthrough Tutorial	D. Kirk	TJ	2,2	Spring 1984	83932
SNAX/APC: Tandem's New SNA Software for Distributed Processing	B. Grantham	TSR	3,1	March 1987	83939
SNAX/HLS: An Overview*	S. Saltwick	TSR	1,2	June 1985	83935
Data Management					
A Comparison of the B00 DP1 and DP2 Disc Processes*	T. Schachter	TSR	1,2	June 1985	83935
DP1-DP2 File Conversion: An Overview*	J. Tate	TSR	2,1	Feb. 1986	83936
DP2's Efficient Use of Cache*	T. Schachter	TSR	1,2	June 1985	83935
DP2 Highlights*	K. Carlyle, L. McGowan	TSR	1,2	June 1985	83935
DP2 Key-sequenced Files*	T. Schachter	TSR	1,2	June 1985	83935
Determining FCP Conversion Time*	J. Tate	TSR	2,1	Feb. 1986	83936
The Relational Data Base Management Solution	G. Ow	TJ	2,1	Winter 1984	83931
Improvements in TMF*	T. Lemberger	TSR	1,2	June 1985	83935
TMF Autorollback: A New Recovery Feature*	M. Pong	TSR	1,1	Feb. 1985	83934
The TRANSFER Delivery System for Distributed Applications	S. Van Pelt	TJ	2,2	Spring 1984	83932
Manuals/Courses					
B00 Software Manuals*	S. Olds	TSR	1,2	June 1985	83935
New Software Courses*	M. Janow	TSR	1,2	June 1985	83935
Subscription Policy for Software Manuals*	T. McSweeney	TSR	2,1	Feb. 1986	83936
Tandem's New Products*	C. Robinson	TSR	2,1	Feb. 1986	83936
Tandem's New Products**	C. Robinson	TSR	2,2	June 1986	83937
Operating Systems					
The GUARDIAN Message System and How to Design for It*	M. Chandra	TSR	1,1	Feb. 1985	83935
Highlights of the B00 Software Release*	K. Coughlin, R. Montevaldo	TSR	1,2	June 1985	83935
Increased Code Space*	A. Jordan	TSR	1,2	June 1985	83935
Managing System Time Under GUARDIAN 90*	E. Nellen	TSR	2,1	Feb. 1986	83936
New GUARDIAN 90 Timekeeping Facilities*	E. Nellen	TSR	1,2	June 1985	83935
New Process-timing Features*	S. Sharma	TSR	1,2	June 1985	83935
NonStop II Memory Organization and Extended Addressing	D. Thomas	TJ	1,1	Fall 1983	83930
Robustness to Crash in a Distributed Data Base: A Nonshared-memory Approach*	A. Borr	TSR	1,2	June 1985	83935
The Tandem Global Update Protocol*	R. Carr	TSR	1,2	June 1985	83935
Performance and Capacity Planning					
Capacity Planning Concepts	R. Evans	TSR	2,3	Dec. 1986	83938
A Performance Retrospective	P. Oleinick, P. Shah	TSR	2,3	Dec. 1986	83938
The Performance Characteristics of Tandem NonStop Systems	J. Day	TJ	1,1	Fall 1983	83930
Performance Considerations for Application Processes	R. Glasstone	TSR	2,3	Dec. 1986	83938
Buffering for Better Application Performance*	R. Mattran	TSR	2,1	Feb. 1986	83936
Optimizing Sequential Processng on the Tandem System	R. Welsh	TJ	2,3	Summer 1984	83933
Predicting Response Time in On-line Transaction Processing Systems**	A. Khatri	TSR	2,2	June 1986	83937
The 6600 and TCC6820 Communications Controllers: A Performance Comparison	P. Beadles	TSR	2,3	Dec. 1986	83938
Improved Performance for BACKUP2 and RESTORE2*	A. Khatri, M. McCline	TSR	1,2	June 1985	83935
Credit-authorization Benchmark for Performance and Linear Growth*	T. Chmiel, T. Houy	TSR	2,1	Feb. 1986	83936
DP2 Performance*	J. Enright	TSR	1,2	June 1985	83935
The ENCORE Stress Test Generator for On-line Transaction Processing Applications	S. Kosinski	TJ	2,1	Winter 1984	83931
FASTSORT: An External Sort Using Parallel Processing	J. Gray, M. Stewart, A. Tsukerman, S. Uren, M. Vaughn	TSR	2,3	Dec. 1986	83938

Article title	Author(s)	Publication	Volume, Issue	Season or Month and Year	Part Number
Performance and Capacity Planning (Continued)					
Performance Measurements of an ATM Network Application	N. Cabell, D. Mackie	TSR	2,3	Dec. 1986	83938
How to Set Up a Performance Data Base with MEASURE and ENFORM	M. King	TSR	2,3	Dec. 1986	83938
MEASURE: Tandem's New Performance Measurement Tool	D. Dennison	TSR	2,3	Dec. 1986	83938
Message System Performance Enhancements	D. Kinkade	TSR	2,3	Dec. 1986	83938
Message System Performance Tests	S. Uren	TSR	2,3	Dec. 1986	83938
The PATHWAY TCP: Performance and Tuning*	J. Vatz	TSR	1,1	Feb. 1985	83934
Understanding PATHWAY Statistics	M. Pong	TJ	2,2	Spring 1984	83932
Sizing Cache for Applications that Use B-series DP1 and TMF**	P. Shah	TSR	2,2	June 1986	83937
Getting Optimum Performance from Tandem Tape Systems	A. Khatri	TSR	2,3	Dec. 1986	83938
NonStop VLX Performance	J. Enright	TSR	2,3	Dec. 1986	83938
Peripherals					
The 5120 Tape Subsystem Recording Technology	W. Phillips	TSR	3,2	Aug. 1987	83940
The 6100 Communications Subsystem: A New Architecture	R. Smith	TJ	2,1	Winter 1984	83931
The 6600 and TCC6820 Communications Controllers: A Performance Comparison	P. Beadles	TSR	2,3	Dec. 1986	83938
Data-encoding Technology Used in the XL8 Storage Facility**	D.S. Ng	TSR	2,2	June 1986	83937
Data-window Phase-margin Analysis**	A. Painter, H. Pham, H. Thomas	TSR	2,2	June 1986	83937
The DYNAMITE Workstation: An Overview*	G. Smith	TSR	1,2	June 1985	83935
An Introduction to DYNAMITE Workstation Host Integration*	S. Kosinski	TSR	1,2	June 1985	83935
Introducing the 3207 Tape Controller*	S. Chandran	TSR	1,2	June 1985	83935
The Model 6VI Voice Input Option: Its Design and Implementation	B. Huggett	TJ	2,3	Summer 1984	83933
Peripheral Subsystems and Interfaces	J. Blakkan	TSR	3,2	Aug. 1987	83940
Streaming Tape Drives	J. Blakkan	TSR	3,2	Aug. 1987	83940
Plated Media Technology used in the XL8 Storage Facility**	D.S. Ng	TSR	2,2	June 1986	83937
The V8 Disc Storage Facility: Setting a New Standard for On-line Disc Storage*	M. Whiteman	TSR	1,2	June 1985	83935
Processors					
The High-Performance NonStop TXP Processor	W. Bartlett, T. Houy, D. Meyer	TJ	2,1	Winter 1984	83931
The NonStop TXP Processor: A Powerful Design for On-line Transaction Processing	P. Oleinick	TJ	2,3	Summer 1984	83933
NonStop VLX Hardware Design	M. Brown	TSR	2,3	Dec. 1986	83938
The VLX: A Design for Serviceability	J. Allen, R. Boyle	TSR	3,1	March 1987	83939
Security					
Distributed Protection with SAFEGUARD**	T. Chou	TSR	2,2	June 1986	83937
System Management					
Configuring Tandem Disk Subsystems	S. Sitler	TSR	2,3	Dec. 1986	83938
Using FOX to Move a Fault-tolerant Application*	C. Breighner	TSR	1,1	Feb. 1985	83934
Introducing TMDS, Tandem's New On-line Diagnostic System*	J. Troisi	TSR	1,2	June 1985	83935
Enhancements to TMDS	L. White	TSR	3,2	Aug. 1987	83940
VIEWSYS: An On-line System-resource Monitor*	D. Montgomery	TSR	1,2	June 1985	83935
Utilities					
Enhancements to PS MAIL	R. Funk	TSR	3,1	March 1987	83939

Index by Product

Article title	Author(s)	Publication	Volume, Issue	Season or Month and Year	Part Number
3207 Tape Controller					
Introducing the 3207 Tape Controller*	S. Chandran	TSR	1,2	June 1985	83935
5120 Tape Subsystem					
The 5120 Tape Subsystem Recording Technology	W. Phillips	TSR	3,2	Aug. 1987	83940
6100 Communications Subsystem					
The 6100 Communications Subsystem: A New Architecture	R. Smith	TJ	2,1	Winter 1984	83931
6530 Terminal					
The Model 6VI Voice Input Option: Its Design and Implementation	B. Huggett	TJ	2,3	Summer 1984	83933
6600 and TCC6820 Communications Controllers					
The 6600 and TCC6820 Communications Controllers: A Performance Comparison	P. Beadles	TSR	2,3	Dec. 1986	83938
ADA					
Ada: Tandem's Newest Compiler and Programming Environment	R. Vnuk	TSR	3,2	Aug. 1987	83940
BASIC					
An Introduction to Tandem EXTENDED BASIC	J. Meyerson	TJ	2,2	Spring 1984	83932
C					
State-of-the-art C Compiler**	E. Kit	TSR	2,2	June 1986	83937
COMINT (CI)					
Writing a Command Interpreter*	D. Wong	TSR	1,2	June 1985	83935
CIS					
Customer Information Service	J. Massucco	TSR	3,1	March 1987	83939
COBOL85					
Tandem's New COBOL85*	D. Nelson	TSR	2,1	Feb. 1986	83936
DP1 and DP2					
Sizing Cache for Applications that Use B-series DP1 and TMF**	P. Shah	TSR	2,2	June 1986	83937
A Comparison of the B00 DP1 and DP2 Disc Processes*	T. Schachter	TSR	1,2	June 1985	83935
DP1-DP2 File Conversion: An Overview*	J. Tate	TSR	2,1	Feb. 1986	83936
DP2's Efficient Use of Cache*	T. Schachter	TSR	1,2	June 1985	83935
DP2 Highlights*	K. Carlyle, L. McGowan	TSR	1,2	June 1985	83935
DP2 Key-sequenced Files*	T. Schachter	TSR	1,2	June 1985	83935
DP2 Performance*	J. Enright	TSR	1,2	June 1985	83935
Determining FCP Conversion Time*	J. Tate	TSR	2,1	Feb. 1986	83936
DYNAMITE					
The DYNAMITE Workstation: An Overview*	G. Smith	TSR	1,2	June 1985	83935
An Introduction to DYNAMITE Workstation Host Integration*	S. Kosinski	TSR	1,2	June 1985	83935
ENABLE					
The ENABLE Program Generator for Multifile Applications*	B. Chapman, J. Zimmerman	TSR	1,1	Feb. 1985	83934
ENCOMPASS					
The Relational Data Base Management Solution	G. Ow	TJ	2,1	Winter 1984	83931
ENCORE					
The ENCORE Stress Test Generator for On-line Transaction Processing Applications	S. Kosinski	TJ	2,1	Winter 1984	83931
FASTSORT					
FASTSORT: An External Sort Using Parallel Processing	J. Gray, M. Stewart, A. Tsukerman, S. Uren, B. Vaughn	TSR	2,3	Dec. 1986	83938
FOX					
Changes in FOX*	N. Donde	TSR	1,2	June 1985	83935
Using FOX to Move a Fault-tolerant Application*	C. Breighner	TSR	1,1	Feb. 1985	83934

Article title	Author(s)	Publication	Volume, Issue	Season or Month and Year	Part Number
GUARDIAN					
The GUARDIAN Message System and How to Design for It*	M. Chandra	TSR	1,1	Feb. 1985	83935
Improved Performance for BACKUP2 and RESTORE2*	A. Khatri, M. McCline	TSR	1,2	June 1985	83935
Increased Code Space*	A. Jordan	TSR	1,2	June 1985	83935
Introducing TMDS, Tandem's New On-line Diagnostic System*	J. Troisi	TSR	1,2	June 1985	83935
Enhancements to TMDS	L. White	TSR	3,2	Aug. 1987	83940
Managing System Time Under GUARDIAN 90*	E. Nellen	TSR	2,1	Feb. 1986	83936
Message System Performance Enhancements	D. Kinkade	TSR	2,3	Dec. 1986	83938
Message System Performance Tests	S. Uren	TSR	2,3	Dec. 1986	83938
New GUARDIAN 90 Timekeeping Facilities*	E. Nellen	TSR	1,2	June 1985	83935
New Process-timing Features*	S. Sharma	TSR	1,2	June 1985	83935
NonStop II Memory Organization and Extended Addressing	D. Thomas	TJ	1,1	Fall 1983	83930
Robustness to Crash in a Distributed Data Base: A Nonshared-memory Multiprocessor Approach*	A. Borr	TSR	1,2	June 1985	83935
The Tandem Global Update Protocol*	R. Carr	TSR	1,2	June 1985	83935
MEASURE					
How to Set Up a Performance Data Base with MEASURE and ENFORM	M. King	TSR	2,3	Dec. 1986	83938
MEASURE: Tandem's New Performance Measurement Tool	D. Dennison	TSR	2,3	Dec. 1986	83938
PATHFINDER					
PATHFINDER—An Aid for Application Development	S. Benett	TJ	1,1	Fall 1983	83930
PATHWAY					
PATHWAY IDS: A Message-level Interface to Devices and Processes**	M. Anderton, M. Noonan	TSR	2,2	June 1986	83937
A New Design for the PATHWAY TCP	R. Wong	TJ	2,2	Spring 1984	83932
The PATHWAY TCP: Performance and Tuning*	J. Vatz	TSR	1,1	Feb. 1985	83934
Understanding PATHWAY Statistics	M. Pong	TJ	2,2	Spring 1984	83932
PS MAIL					
Enhancements to PS MAIL	R. Funk	TSR	3,1	March 1987	83939
SAFEGUARD					
Distributed Protection with SAFEGUARD**	T. Chou	TSR	2,2	June 1986	83937
SNAX					
A SNAX Passthrough Tutorial	D. Kirk	TJ	2,2	Spring 1984	83932
SNAX/APC: Tandem's New SNA Software for Distributed Processing	B. Grantham	TSR	3,1	March 1987	83939
SNAX/HLS: An Overview*	S. Saltwick	TSR	1,2	June 1985	83935
TACL					
TACL, Tandem's New Extensible Command Language*	J. Campbell, R. Glascock	TSR	2,1	Feb. 1986	83936
TAL					
New TAL Features**	C. Lu, J. Murayama	TSR	2,2	June 1986	83837
TMF					
Improvements in TMF*	T. Lemberger	TSR	1,2	June 1985	83935
TMF and the Multi-Threaded Requester	T. Lemberger	TJ	1,1	Fall 1983	83930
TMF Autorollback: A New Recovery Feature*	M. Pong	TSR	1,1	Feb. 1985	83934
TRANSFER					
The TRANSFER Delivery System for Distributed Applications	S. Van Pelt	TJ	2,2	Spring 1984	83932
TXP					
The High-Performance NonStop TXP Processor	W. Bartlett, T. Houy, D. Meyer	TJ	2,1	Winter 1984	83931
The NonStop TXP Processor: A Powerful Design for On-line Transaction Processing	P. Oleinick	TJ	2,3	Summer 1984	83933

Article title	Author(s)	Publication	Volume, Issue	Season or Month and Year	Part Number
V8					
The V8 Disc Storage Facility: Setting a New Standard for On-line Disc Storage*	M. Whiteman	TSR	1,2	June 1985	83935
VIEWSYS					
VIEWSYS: An On-line System-resource Monitor*	D. Montgomery	TSR	1,2	June 1985	83935
VLX					
NonStop VLX Hardware Design	M. Brown	TSR	2,3	Dec. 1986	83938
NonStop VLX Performance	J. Enright	TSR	2,3	Dec. 1986	83938
The VLX: A Design for Serviceability	J. Allen, R. Boyle	TSR	3,1	March 1987	83939
XL8					
Data-encoding Technology Used in the XL8 Storage Facility**	D.S. Ng	TSR	2,2	June 1986	83937
Plated Media Technology used in the XL8 Storage Facility**	D.S. Ng	TSR	2,2	June 1986	83937
Miscellaneous[2]					
Performance Measurements of an ATM Network Application	N. Cabell, D. Mackie	TSR	2,3	Dec. 1986	83938
Buffering for Better Application Performance*	R. Mattran	TSR	2,1	Feb. 1986	83936
Capacity Planning Concepts	R. Evans	TSR	2,3	Dec. 1986	83938
Credit-authorization Benchmark for High Performance and Linear Growth*	T. Chmiel, T. Houy	TSR	2,1	Feb. 1986	83936
Data-window Phase-margin Analysis**	A. Painter, H. Pham, H. Thomas	TSR	2,2	June 1986	83937
Configuring Tandem Disk Subsystems	S. Stiler	TSR	2,3	Dec. 1986	83938
Peripheral Subsystems and Interfaces	J. Blakkan	TSR	3,2	Aug. 1987	83940
A Performance Retrospective	P. Oleinick	TSR	2,3	Dec. 1986	83938
Performance Considerations for Application Processes	R. Glasstone	TSR	2,3	Dec. 1986	83938
The Performance Characteristics of Tandem NonStop Systems	J. Day	TJ	1,1	Fall 1983	83930
Predicting Response Time in On-line Transaction Processing Systems**	A. Khatri	TSR	2,2	June 1986	83937
Remote Support Strategy	J. Eddy	TSR	3,1	March 1987	83939
Optimizing Sequential Processing on the Tandem System	R. Welsh	TJ	2,3	Summer 1984	83933
Tandem's Software Support Plan	R. Baker, D. McEvoy	TSR	3,1	March 1987	83939
Streaming Tape Drives	J. Blakkan	TSR	3,2	Aug. 1987	83940
Getting Optimum Performance from Tandem Tape Systems	A. Khatri	TSR	2,3	Dec. 1986	83938
Highlights of the B00 Software Release*	K. Coughlin, R. Montevaldo	TSR	1,2	June 1985	83935
B00 Software Manuals*	S. Olds	TSR	1,2	June 1985	83935
New Software Courses*	M. Janow	TSR	1,2	June 1985	83935
Subscription Policy for Software Manuals*	T. McSweeney	TSR	2,1	Feb. 1986	83936
Tandem's New Products*	C. Robinson	TSR	2,1	Feb. 1986	83936
Tandem's New Products**	C. Robinson	TSR	2,2	June 1986	83937

[2]This category contains *Tandem Systems Review* articles that contain product information but are not specifically product-related.

TANDEM PUBLICATIONS ORDER FORM

The *Tandem Systems Review* and the Tandem Application Monograph Series are combined in one free subscription. Use this form to subscribe, change a subscription, and order back copies.

For requests *within the U.S.*, send this form to:

Tandem Computers Incorporated
Tandem Systems Review
1309 South Mary Avenue, MS 5-04
Sunnyvale, CA 94087

For requests *outside the U.S.*, send this form to your local Tandem sales office.

Check the appropriate box(es):

- ☐ New subscription (# of copies desired _____)
- ☐ Subscription change (# of copies desired _____)
- ☐ Request for back copies. (Shipment subject to availability.)

Print your current address here:

_____
COMPANY NAME

_____
ADDRESS

_____

_____

_____

_____
ATTENTION

_____
PHONE NUMBER (U.S.)

If your address has changed, print the old one here:

_____
COMPANY NAME

_____
ADDRESS

_____

_____

_____

_____
ATTENTION

_____
PHONE NUMBER (U.S.)

To order back copies, write the number of copies next to the title(s) below.

NUMBER OF COPIES

Tandem Journal

_____ Part No. 83930, Vol. 1, No. 1, Fall 1983

_____ Part No. 83931, Vol. 2, No. 1, Winter 1984

_____ Part No. 83932, Vol. 2, No. 2, Spring 1984

_____ Part No. 83933, Vol. 2, No. 3, Summer 1984

Tandem Systems Review

_____ Part No. 83937, Vol. 2, No. 2, June 1986

_____ Part No. 83938, Vol. 2, No. 3, December 1986

_____ Part No. 83939, Vol. 3, No. 1, March 1987

_____ Part No. 83940, Vol. 3, No. 2, August 1987

Tandem Application Monograph Series

_____ Part No. 83900, *Developing TMF-Protected Application Software,* March 1983, AM-005

_____ Part No. 83901, *Designing a Tandem/Word Processor Interface,* March 1983, AM-006

_____ Part No. 83902, *Integrating Corporate Information Systems: The Intelligent-Network Strategy,* March 1983, AM-007

_____ Part No. 83903, *Application Data Base Design in a Tandem Environment,* August 1983

_____ Part No. 83904, *Capacity Planning for Tandem Computer Systems,* October 1984

_____ Part No. 83905, *Sociable Systems: A Look at the Tandem Corporate Network,* May 1985

_____ Part No. 83906, *Transaction Processing on the Tandem NonStop Computer: Requestor/Server Structures,* January 1982, SEDS-001

_____ Part No. 83907, *Designing a Network-Based Transaction-Processing System,* April 1982, SEDS-002

_____ Part No. 83908, *A Close Look at PATHWAY,* June 1982, SEDS-003

_____ Part No. 83909, *A Multi-Function Network for Business Automation,* May 1982, SEDS-004

TANDEM EMPLOYEES: PLEASE ORDER YOUR COPIES THROUGH YOUR MARKETING LITERATURE COORDINATOR.

8/87

TANDEMCOMPUTERS

Part No. 83940

400102 8/87 Printed in USA