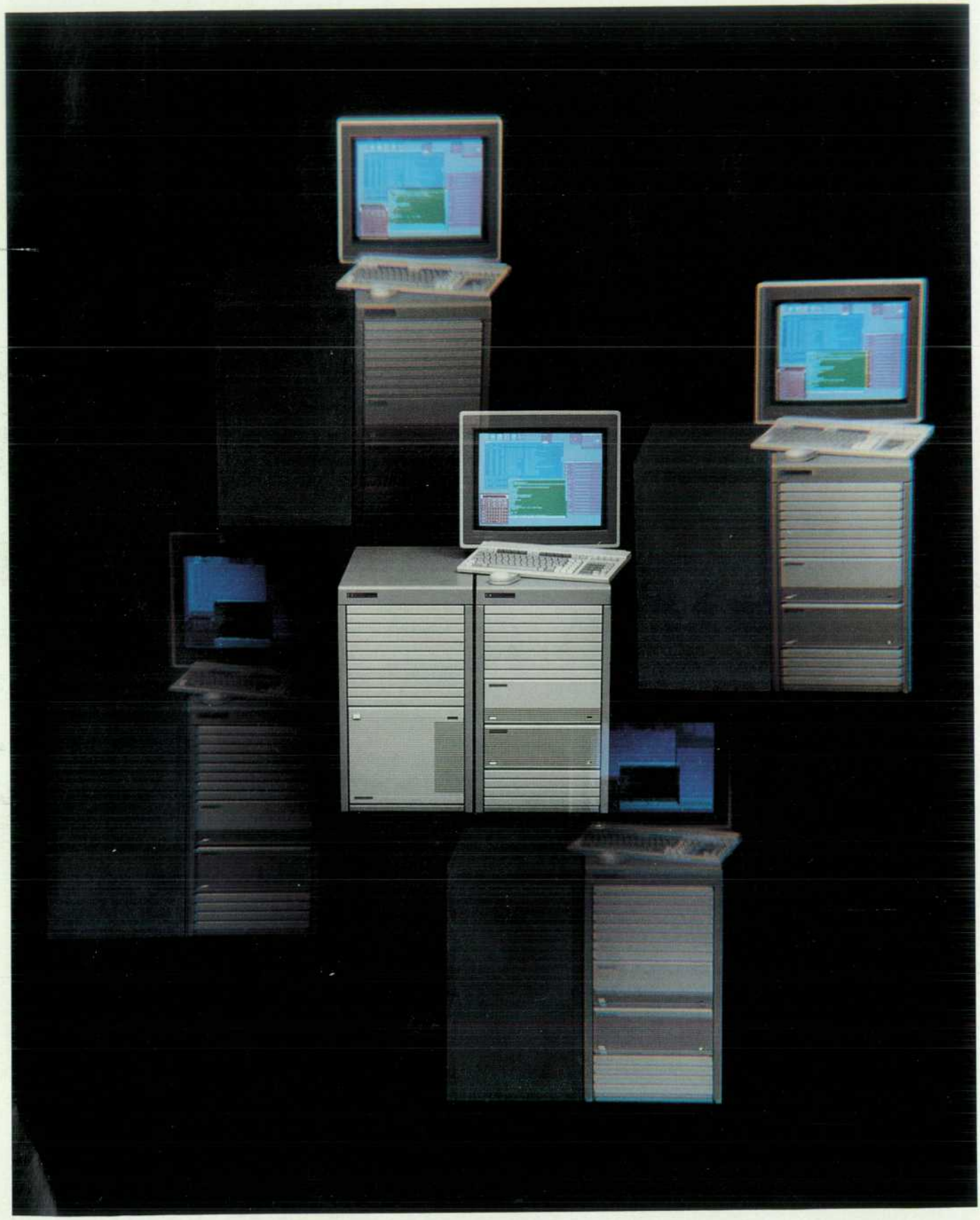
HEWLETT-PACKARD
JOURNAL

OCTOBER 1988



HEWLETT
PACKARD

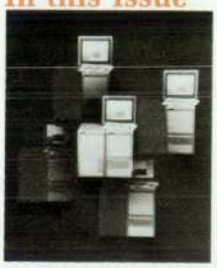
HEWLETT-PACKARD JOURNAL

October 1988 Volume 39 • Number 5

## Articles

## Departments

## In this Issue



In engineering workstations running under AT&T's UNIX® operating system or one of its many versions, such as Hewlett-Packard's HP-UX, a lot of disc space is used for system code and standard utilities that every workstation must have. When several UNIX workstations are clustered on a local area network, it's natural to think of lowering disc memory costs by storing these common programs at only one workstation and allowing the other workstations to access them over the network instead of having individual copies. To take the idea a step farther, user disc files can also be concentrated at a single workstation, so the other workstations don't need disc drives at all. This is the objective of the HP-UX 6.0 operating system for HP 9000 Series 300 Computers. With this system, tightly networked discless graphics workstations on an IEEE 802.3 local area network can share a single file system server. A simplified, proprietary networking protocol delivers discless workstation performance that comes close to stand-alone performance, while industry-standard networking services, such as ARPA/Berkeley and NFS, are provided for intervendor and intercluster communication and file sharing. HP-UX 6.0 also supports the industry-standard SCSI and VME interfaces, the X Window System, and high-performance HP 9000 graphics subsystems. Major features are the single-system view presented to users and the high degree of network transparency achieved. The system looks the same from any workstation in a cluster, and network operation is transparent to the user. While the idea behind the HP-UX 6.0 system is simple, the engineering was not. Following an introduction to the system on page 6, eight papers discuss the design challenges the development team had to deal with. These include the implementation of a discless file system (page 10), discless program execution and virtual memory management (page 15), network function and protocol design (page 20), crash detection and recovery (page 27), boot mechanism design (page 33), and system configuration (page 37). SCSI and X Window System support are described in the papers on pages 39 and 46.

The August 1988 issue featured the HP PaintJet Color Graphics Printer and its contributions to thermal inkjet printing technology, which include a second-generation printhead design, resolution of 180 dots per inch (nearly double that of the ThinkJet printer introduced in 1984), and full-color printing on paper or overhead transparency film. This issue presents the next chapter of this story, which stars the HP DeskJet printer. Delivering laser-quality printing at 300-dot-per-inch resolution on standard office papers, the DeskJet printer is priced competitively with noisier, less reliable personal printers offering much lower print quality. DeskJet features include merged text and graphics, multiple fonts, two slots for font or personality cartridges, 120-character-per-second letter-quality speed, and a built-in cut-sheet paper feeder. Beginning with management issues on page 51, eight papers in this issue tell the story of DeskJet development. The third-generation, high-resolution, thermal inkjet printhead is discussed beginning on page 55. Electrical connections to the print cartridge, and the systems that hold, move, protect, and maintain the cartridge and fire the ink drops are described in the paper on page 62. The multifunction chassis, designed using an HP CAD system, the paper handling system, an unusual transmission that lowers costs by making one motor perform three functions, and the paper drive motor and its control system are treated in the paper on page 67, while the paper on page 76 tells how a microprocessor-controlled custom integrated circuit manipulates character dot data to provide various text enhancements and graphics. Firmware design is the subject of the paper on page 81, and two manufacturing papers, one on robotic circuit board assembly and one on machine vision systems for printhead production, are on pages 87 and 91.

The shaft encoder that provides feedback for the DeskJet printer paper drive servo system is available to customers as a separate product line, the HEDS-9000 Shaft Encoder Module family. Designed for low cost, rapid assembly, and freedom from follow-up adjustments, this encoder module makes closed-loop operation feasible for low-cost products like the DeskJet printer, where it translates into higher speed and print quality. The HEDS-9000 design includes elements of integrated detector circuits, light-emitting diode technology, plastic optics, and high-volume manufacturing. The story begins on page 99.

-R.P. Dolan

## Cover

The cover photograph represents the HP-UX 6.0 discless operating system. The photographer has used a special lens to multiply the image of this HP 9000 Series 300 workstation to simulate a cluster of workstations on a local area network. All but one of the disc drive images have been faded back, indicating that in an HP-UX 6.0 discless cluster, only one workstation needs to have a disc drive.

## What's Ahead

The HP NewWave environment, a state-of-the-art user interface for personal computers, is the major subject in the December issue. There will also be articles on the HP 64700 Series host independent emulators for microprocessor-based system development, on the plain paper research that was done for DeskJet printer development, and on a technique for adjusting dual-channel data sampled by the HP 5180A Waveform Recorder. The annual index will also be presented.

The **Hewlett-Packard Journal** is published bimonthly by the Hewlett-Packard Company to recognize technical contributions made by Hewlett-Packard (HP) personnel. While the information found in this publication is believed to be accurate, the Hewlett-Packard Company makes no warranties, express or implied, as to the accuracy or reliability of such information. The Hewlett-Packard Company disclaims all warranties of merchantability and fitness for a particular purpose and all obligations and liabilities for damages, including but not limited to indirect, special, or consequential damages, attorney's and expert's fees, and court costs, arising out of or in connection with this publication.

**Subscriptions:** The Hewlett-Packard Journal is distributed free of charge to HP research, design, and manufacturing engineering personnel, as well as to qualified non-HP individuals, libraries, and educational institutions. Please address subscription or change of address requests on printed letterhead (or include a business card) to the HP address on the back cover that is closest to you. When submitting a change of address, please include your zip or postal code and a copy of your old label.

**Submissions:** Although articles in the Hewlett-Packard Journal are primarily authored by HP employees, articles from non-HP authors dealing with HP-related research or solutions to technical problems made possible by using HP equipment are also considered for publication. Please contact the Editor before submitting such articles. Also, the Hewlett-Packard Journal encourages technical discussions of the topics presenting in recent articles and may publish letters expected to be of interest to readers. Letters should be brief, and are subject to editing by HP.



Please address inquiries, submissions, and requests to: Editor, Hewlett-Packard Journal, 3200 Hillview Avenue, Palo Alto, CA 94304, U.S.A.

# Discless HP-UX Workstations

*HP-UX 6.0 provides low-cost discless workstation operation over a local area network. It also provides a single file system view, intervendor file sharing, and conformance to UNIX® System V Interface Definition (SVID) semantics.*

**by Scott W. Wang**

THE HP-UX RELEASE 6.0 SYSTEM is a major software contribution to the HP 9000 Series 300 workstation platform. This release of the HP-UX operating system provides discless workstation operation in a network and intervendor file sharing through the Network File System (NFS*).

The HP-UX 6.0 system enables tightly networked discless graphics workstations to share a single file system server transparently in an Ethernet or IEEE 802.3 local area network. Fig. 1 shows a typical HP-UX 6.0 system configuration and defines a few terms that are used here and in other articles in this issue. The terms discless cnode (cluster node) and discless workstation are used interchangeably in this article.

The standard ARPA/Berkeley networking services and NFS complement the tightly coupled workstations by offering intervendor and intercluster communication and file sharing capabilities. In addition to the discless and NFS capabilities, the HP-UX 6.0 system also offers:

- Industry standard Small Computer System Interface (SCSI) and VME support
- Enhanced graphics support for the new HP 98550A high-resolution graphics board and displays and the HP 98556A 2D integer-based graphics accelerator
- Commands and libraries from Release 1.0 of the HP 9000 Series 800 HP-UX system
- The X Window System

SCSI and the X Window System are discussed on pages 39 and 46, respectively.

*NFS is a product of Sun Microsystems, Inc.
UNIX is a registered trademark of AT&T in the U.S.A. and other countries.

## Design Goals

There are many ways to implement a discless workstation capability. However, our design choices and implementation techniques were guided by the need to achieve the highest quality goals of functionality, usability, reliability, performance, and supportability. This resulted in the following design goals for our discless workstation implementation:

- Low-cost discless workstation operation over a local area network
- A single file system view
- Conformance to AT&T's *UNIX System V Interface Definition* (SVID) semantics and backward compatibility with previous releases of HP-UX
- A design that coexists with and complements NFS, HP's Network Services (NS), and ARPA/Berkeley network facilities
- At least 80% of the throughput performance of a standalone system (workstation with a disc)
- Flexible system configuration and dynamic reconfiguration
- Thorough usability and reliability testing.

## Low-Cost Discless Workstations

Clustering discless workstations is a way to achieve lower cost per workstation, to meet certain environmental conditions (poor environment for discs), and to meet specific ergonomic requirements. To operate in a discless mode the workstation needs access to a remote file server for booting up, for gaining access to files, and for doing virtual memory swapping from the server's disc. Remote

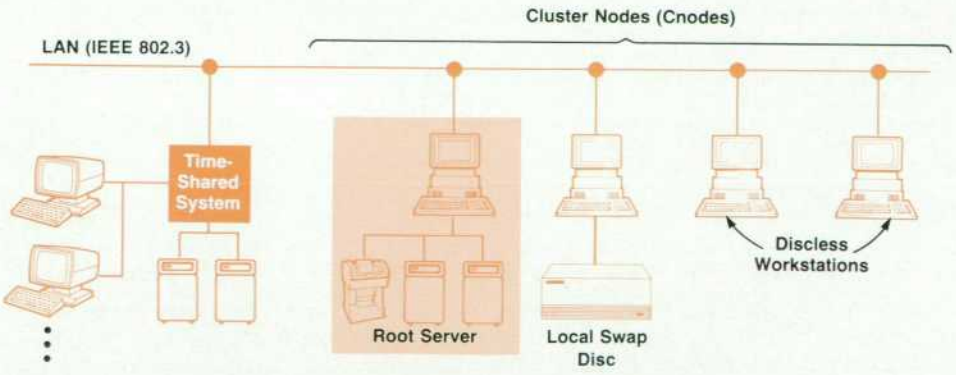



**Fig. 1.** *Major components of a cluster of discless workstations. A cluster is a group of workstations connected by a network that share a single file hierarchy. A cluster node (cnode) is one of the nodes, or workstations in a cluster. A discless cnode is a cnode that does not have a local file system; its file system resides on the root server. A root server is the cnode to which the disc containing the root file system is physically attached. There is only one root server for each cluster. In the HP-UX 6.0 system release the root server is also the file server.*

boot and virtual memory operations are described in detail in the articles "Boot Mechanism for Discless HP-UX," and "Discless Program Execution and Virtual Memory Management," on pages 33 and 15, respectively.

### Single File System View

There are two basic computing environment models: time-shared systems and distributed systems. Time-shared systems allow multiple users to communicate with each other easily, and to share a single computer's environment and resources. The disadvantages of a time-shared system are poor response time, limited configuration and scalability, limited graphics capability, and limited system availability. Distributed systems alleviate many of the disadvantages of time-shared systems by distributing the computing and other resources onto networked full graphics workstations that are smaller and less expensive. However, sharing resources and communicating between users on separate workstations is usually more complex in a distributed system. For the HP-UX 6.0 system we wanted the best of both models: a high degree of network transparency between workstations and a single-system view.

A single-system view in a workstation cluster means the user sees a single file system from any workstation and there is a single point for system administration. A user can log in to the system from any workstation in the cluster and see the same environment in the same manner as seen when logging into a time-shared system from any terminal. Single-point system administration means the system administrator can administer the cluster of workstations from any workstation in the cluster, and the work involved is no more complex than a time-shared system with the same number of users.

Most important, a single-system view in a cluster means a single global file system. Each workstation user sees and shares the same file system just as in a multiuser time-shared system. The implementation of this concept means solving many interesting technical problems. For example, file synchronization needs to be maintained between workstations in the same standard HP-UX semantic exhibited in a multiuser HP-UX system. There are subtleties and implications in performance because of file system buffer caching that involves file accesses in both synchronous and asynchronous modes.

A single-system view also means shielding the user from differences in the workstations in the cluster. In a single cluster, workstations may have different types of CPU (e.g., 68010 and 68020), different floating-point processors (e.g., 68881 versus a floating-point card), and different graphics displays. To solve this problem the concept of *context dependent files* (CDF) was defined and implemented for discless workstations. Each workstation has a context file describing that workstation. CDFs reside in a hidden directory that holds context dependent objects (text files and executables), and maintain the same file path name from any cnode in the cluster. This allows a CDF to be accessed using the same file name from any cnode, with the system automatically differentiating and selecting the proper CDF based on the workstation configuration.

A single-system view in a cluster creates the problem of process ID (PID) collisions between independently executing HP-UX environments in the workstations. Collision must be avoided since HP-UX uses PIDs as unique identifiers in many places (e.g., temporary file names). Similarly, clocks in individual workstations in a cluster must be synchronized to have a consistent time in the cluster. The single file system demands that timestamps on files be consistent no matter which workstation puts the timestamp on the file. This has interesting implications for the `make` command if the clocks are not synchronized.

Additional details on the file system can be found in the article, "A Discless HP-UX File System," on page 10.

### Compatibility

Conformance to AT&T's *UNIX System V Interface Definition* (SVID) and object code compatibility with previous releases of the HP 9000 Series 300 HP-UX systems were objectives in all design considerations for the HP-UX 6.0 system. For example, the process ID collision problem mentioned above cannot be solved by simply prepending a cnode ID number to the PIDs to make them unique. Instead, PIDs must remain five digits (1 to 32768) for compatibility. The problem is solved by a PID server process that manages and allocates PIDs in chunks to the discless cnodes while guaranteeing their uniqueness in a cluster. Other examples are file synchronization and file locking, which must be done in a way to preserve standard HP-UX semantics. See the article "A Discless HP-UX File System" for more details. Ensuring conformance to the SVID, the HP-UX 6.0 system has passed the System V Validation Suite (SVVS).

### Other Network Protocols

While the discless capability is the primary objective of the HP-UX 6.0 system, another objective was to allow access to NFS, HP's NS, and ARPA/Berkeley network services concurrently with the discless functions. Implementation of these capabilities affects the file system and the network system. For example, the key to a single-system view is the file system. This means we had to integrate all the requirements for other network file systems into the same file system used for the discless implementation.

### Discless Performance

In a discless environment, some performance loss is unavoidable because of remote file accessing and virtual memory swapping over the network. The performance goal we set for the HP-UX 6.0 system was 80% of a stand-alone workstation's throughput performance. Three areas were identified as key to achieving this performance: network protocol, virtual memory swapping, and file system caching.

A lightweight protocol was defined to handle the kernel-to-kernel communication between a discless cnode and the server. This resulted in a significant performance advantage compared to other discless implementations based on standard network protocols such as TCP/IP. The discless protocol is discussed in the article "The Design of Network Functions for Discless Clusters," on page 20. A performance analysis is also included in the article.

To address the performance bottleneck of remote swapping at the file server, we include support for local swap discs on a discless cnode. For virtual memory intensive applications running on a discless cnode, the user has the

options of adding a local swapping disc to improve performance while maintaining the single file system view, and of sharing resources with other cnodes.

Standard HP-UX file system buffer caching is maintained on the server and the client cnodes, thus maintaining the performance improvement file caching provides. This is discussed in more detail in the article "A Discless HP-UX File System."

### Flexible Configuration

For HP-UX system 6.0, all models of the Series 300 family of workstations are supported. However, the server is restricted to the Series 350 only. Every workstation, including the server, runs the same version of the HP-UX 6.0 system. The server is not a dedicated server, in that it can also be used as a workstation. In addition, the discless cnodes and the server retain their ability to support multiple terminal users if desired. The cluster size and configuration depend on the requirements of users and applications.

### Dynamic reconfiguration

Users can add cnodes to or delete cnodes from a cluster and move cnodes from one cluster to another. A cluster can dynamically grow or shrink as necessary.

A cluster can start from as little as two workstations and expand as required without unloading the file system and repartitioning the disc. Discless cnodes can join and unjoin a cluster at boot time without affecting the activity of the rest of the cluster. The new cnode is immediately recognized by other cnodes in the cluster. When a cnode leaves a cluster the rest of the cluster will automatically reconfigure and continue operation. Multiple clusters can be defined on a single LAN and each discless cnode on the LAN can easily choose to join any cluster during boot.

To maintain the single-system view, the configuration of discless cnodes must be as simple as adding a terminal to a multiuser time-shared system. Because of the single file system implementation it is not necessary to partition the server disc according to the number of discless cnodes in the cluster. The file system and swap area on the server disc are shared by all discless cnodes. This allows the system to pool a large swap area when large swap intensive application programs are executed.

Easy cluster definition and configuration are accomplished through a program called reconfig. This is described in the article "Discless System Configuration Tasks," on page 37.

Another example of flexibility and ease of configuration is sharing of peripherals on the server, and the ability to configure local devices on the discless nodes.

## Program Management

The HP-UX 6.0 system release was a large team effort spanning many organizations and functional areas. The management of this release was an excellent example of the concept called program management. The organizations involved were HP's System Software Operation (SSO), Technical Workstation Operation (TWO), Corvallis Workstation Operation (CWO), Information Systems Operation (ISO), and Colorado Networks Operation (CND). The functional areas involved included several R&D labs including operating systems, languages, graphics, commands and libraries, networking, performance, system integration, and program management. Other functional areas were product marketing from the various organizations, marketing support, documentation, quality assurance, and manufacturing. In addition, there were many applications organizations such as the Electronic Design Division (EDD) and Logic Systems Division (LSD) that needed to be kept informed of our progress. These two organizations and others were collectively called the Engineering System Group (ESG*) partners.

The program management model centered on what was called the HP-UX team, which consisted of representatives from the various organizations and functional areas. The HP-UX team met weekly for status updates, information, and issue resolution.

Program management documents included the team meeting minutes, a system PERT chart, a program data sheet, a program requirements document, a delta document, commitment lists, and a milestone checklist. Several of these documents deserve further explanation. The delta document was published early for the ESG and other partners. It contained the differences between Release 5.5 and Release 6.0 of the HP-UX system, and items that could affect the partner application and subsystem development. For example, the need for a new boot ROM affected header file changes, object code compatibility issues, and code size estimates. The commitment list was important because it provided us with a central list of all known customer commitments in terms of early release requirements, who made the commitments, when they were required, and whether they required a discless system or just NFS. The milestone checklist was used to track all major action items and all known major and minor milestones. It was reviewed and updated at each team meeting. The checklist not only enabled us to check progress and follow up on action items, but also showed progress being made. The checklist was a great supplement to the system PERT chart which was also reviewed each week.

The partners were kept up to date by means of the delta document and in some cases the team meeting minutes. We also had a monthly (later bimonthly) meeting to share status. This was called the ESG information exchange meeting. It was an effective way to exchange data and keep each other informed. I also served as the major interface to people in California through the periodic HP-UX Steering Council meetings.

The HP-UX 6.0 system program life cycle included three early bird (EB) releases that were roughly two months apart. EB1 was used to tune the processes used to build and put the system through integration and test. EB1 turned out usable enough for distribution to partners and selected customers. EB2 was a functionally complete system for entry into final QA and for partner and customer commitment distributions. EB2 was also used in the first human factor usability testing. EB3 was the final refinement of the product and the release before the final system integration and test process. In essence, the EBs were trial runs for the real release and at the same time served as useful systems.

*ESG is currently called Engineering and Measurement System Group (EMSG).

*Scott W. Wang*
R&D Lab Manager
Information Software Division

### Usability and Reliability

Features that contribute to the usability of the HP-UX 6.0 system include the single-system view, ease of configuration, and compatibility. We worked with human factors engineers to test our early releases for usability. This testing resulted in many changes to the documentation and enhancements to the reconfig program.

Reliability is achieved by extensive prototyping, design reviews, and testing. Besides the typical operating system testing done in the past, we designed and executed additional test cases specifically for the discless cluster configurations. Test clusters were set up to run a networking test scaffold at various stress levels. The HP-UX 6.0 system achieved 120 hours of continuous high-stress operation without a system crash.

The dynamic reconfiguration capability also enhances cluster reliability. When a discless cnode crashes, the rest of the discless cnodes will continue to function unaffected. This requires extensive crash detection and recovery in the operating system. However, the entire cluster will cease to operate if the server with the root file system crashes. To ensure detection of and recovery from LAN cable disconnections without affecting other cluster operations, a cable break detection mechanism has been incorporated into the system. Refer to the article "Crash Detection and Recovery in a Discless HP-UX System" on page 27 for more details.

### Acknowledgments

The technology for the discless capability started as a distributed HP-UX (called DUX[1]) project at HP Laboratories in Palo Alto. This research resulted in a prototype implementation of distributed HP-UX that was developed at the Information Software Operation in Cupertino, California and the System Software Operation in Fort Collins, Colorado. DUX incorporated a fully distributed file system and many advanced distributed operating system features.

I would like to acknowledge the many people who made the HP-UX 6.0 system a reality. It is not possible to list all the names here so the list is limited to the core operating system teams.

Xuan Bui and his kernel group: Drew Anderson, Jack Applin, Doug Baskins, Paul Stoecker, Paul Perlmutter, Pamela Marchall. Joe Cowan and his kernel group: Bruce Bigler, Dave Gutierrez, Bob Lenk, Jack McClurg, Bill McMahon, Perry Scott, Rober Quist. Ken Martin and his system integration group: Stuart Bobb, Paul Christofanelli, Jim Darling, Steve Ellcey, Bill Mullaney, Bruce Rodean, Kim Wagner. Marcel Meier and his kernel group: Debbie Bartlett, Mike Berry, Barbara Flahive, Ping-Hui Kao, Anny Randel, Fred Richart. Bonnie Stahlin and her program management and usability/test group: Rich Dunker, Lois Gerber, Dave Grindeland, Mike Steckmyer, Ron Tolley. Donn Terry and his commands and libraries group: Jer/ Eberhard, Gayle Guidry Dilley, Rob Gardner, John Marvin, Rob Robason, and Peter van der Steur.

In addition I would like to acknowledge the California contingent: Ching-Fa Hwang, Joel Tesler, Sui-Ping Chen, Chyuan-Shiun Lin, Doug Hartman, Jeff Glasson, Mike Saboff, and Ed Sesek.

I would especially like to acknowledge John Romano from Logic Systems Division for his early realization that DUX was a must requirement for his HP 64000 market, and Ching-Fa Hwang and his team at HP Labs that built the original DUX: Joel Tesler, Chyuan-Shiun Lin, John Worley, Sui-Ping Chen, Parviz Afshar, Curt Kolovson, and Ray Cheng. Their continued moral support for this project was invaluable.

Steve Boettner, Bill Eads, Gary Ho, Eric Neuhold, and Mike Kolesar provided management support. Finally, a special thanks to Sandy Chumbley, then System Software Operation manager, for sticking with us all the way.

### Reference

1. Ching-Fa Hwang, J. Tesler, and Chyuan-Shiun Lin, "Achieving a One-System View for Distributed UNIX Operating Systems," *UniForum 1987 Conference Proceedings.*

# A Discless HP-UX File System

by Debra S. Bartlett and Joel D. Tesler

THE MOST OBVIOUS REQUIREMENT of any discless system is a file service capability. All files must be stored on a file-serving node since the discless nodes normally would not have a local file system. The goal of a single-system view for an HP-UX discless cluster imposes an additional requirement—the file system should appear the same from all nodes in the cluster.

Several changes were made to the file system portion of the standard HP-UX kernel to support discless operations. These changes were made with the requirement of maintaining stand-alone HP-UX semantics and file system performance in a discless environment. Elements of the file system that were modified include: file system I/O, named FIFOs, file locking, and pathname lookup.

The discless file system operates in conjunction with the remainder of the kernel and other file systems. In particular, the discless system is designed to work together with the Sun Microsystems Network File System (NFS), which provides transparent access to files on remote machines in a heterogeneous environment. The discless file system design is such that it enhances the functionality of both file systems rather than requiring the user to choose between them.

To understand the discless file system, the reader should be familiar with the standard HP-UX file system. Fig. 1 explains several common file system terms used throughout the remainder of this article.

### System Appearance

The simplest way to implement a discless system is to partition the server's discs into multiple subdiscs. Each subdisc would be allocated to one client. The client would treat that disc as if it were local, except that all I/O would be performed over the network rather than directly to disc. While this solution does eliminate the need to attach a disc to each CPU, it fails to meet many of the other needs of a discless system. It is still necessary to provide just as much disc space as it would be if each machine had its own physical disc. Such a system would also provide no file sharing; each machine would have its own set of files. Finally, each file system would need to be independently administered.

Since the above approach has many problems and little benefit, it is rarely used. Instead, a common approach to implementing a discless file system is to provide each node with a small root file system physically located at the disc server. This root file system is private to the node owning it, and contains enough files to boot up the system. After booting, the node issues remote mount requests to mount other shared file systems from the disc server. A remote mount is similar to a normal mount in that it mounts one file system under a directory in another file system. However, the file system being mounted is remote, and is usually shared by several clients. Typically some form of remote file service is used to access the files in a transparent manner.

This approach solves several of the problems of a discless file system, but there are still some limitations that make it unsuccessful in meeting the goals of the HP-UX discless system. Each node still has its own root file system, violating the single-system view. It is possible that the various root file systems will differ from one another. In particular, the disc server's root file system is likely to differ significantly from the client file systems. Each root file system must be independently administered, eliminating the possibility of single-machine administration. Finally, each machine must independently perform the remote mounts. It is possible that different machines will perform different mounts, leading to inconsistent views. Even if the system administrator tries to keep the views the same, it is necessary to guarantee that all updates to the mount table are propagated to all machines, a task that is error-prone.

In the HP-UX discless system, we have chosen instead to have a single root file system, residing at the disc server (also referred to as the root server). All nodes in the cluster (hereafter referred to as cnodes) share the same root. Whenever a file system is mounted at the root server, all other cnodes are notified that the mount has taken place. When a new cnode joins a cluster, it inherits the complete mount table from the server. Since the same mount table is used globally, we refer to it as a global mount table. By sharing the root and the mount table, we provide a one-system view. A user can sit at any cnode and perceive the same file system. A system administrator has only a single file system to administer, and need not worry about propagating changes between cnodes.

Providing a global file system is not sufficient to provide a single-system view. It is also necessary to guarantee that the semantics of file system access throughout the cluster are identical to the semantics used when accessing the files on a stand-alone HP-UX system. Commands to manipulate files must remain the same, the system call interface to the operating system should be unchanged, and applications should not need to know whether they are running on a discless cnode or on the disc server. Furthermore, the semantics used to access files from several cnodes should be the same as if all the accessing programs were running on the same cnode. For example, if one program is writing data to a file, a program reading from that file should see the data immediately after it is written, regardless of whether the reader is on the same cnode as the writer.

### Context Dependent Files

The one-system view presented by the discless system has been stressed. Every cnode has the same view of the file system layout, and sees the same files. While this is an ideal situation, there are a few cases where this is actually not the ideal behavior. As an example, consider an application that can make good use of a floating-point co-

processor if it is present, but can run with floating-point libraries if necessary. Some cnodes may have the coprocessor and others may not. It is necessary that the application be able to run on both. While it is possible to link the program with a library that checks for the coprocessor and performs the correct operation for each cnode, this would be inefficient and would not take advantage of the compiler's built-in floating-point code generation capabilities. What is really wanted is two versions of the program: one compiled with the coprocessor code and one compiled without. The user should not need to determine which version to run, but should be able to give the same program name on either type of machine, with the operating system determining the correct program to run. Although there will not be an actual one-system view, since users on different machines will see different programs, there will appear to be a one-system view, since a single program name will attain the same functionality with only a difference in performance.

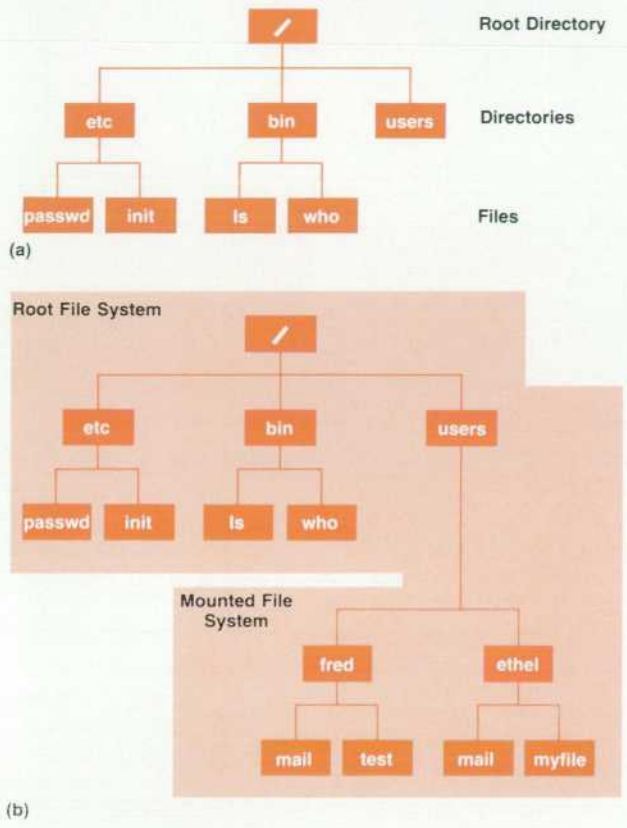



**Fig. 1.** *Standard HP-UX file system. HP-UX uses a hierarchical file system. One special type of file is a directory, which contains a list of files. These files may themselves be directories, or they may be simple files. The top directory of a file system is called the root, and is signified by /. A directory may be empty. (a) shows a miniature HP-UX file system. The root directory contains three directories,* etc, bin, *and* users. Etc *contains two files,* passwd *and* init. *When writing a file name, the components are separated by slashes, for example* /etc/init. *It is possible to attach other file systems by mounting them on a directory. (b) shows a second file system containing user files mounted under* /users. *Once the mount takes place, the second file system can be accessed as if it were part of the first, e.g.,* /users/ethel/myfile.

Another case where each machine may need a different file is when the file describes the machine configuration. For example, the file /etc/inittab describes, among other things, the terminals connected to the CPU. Each CPU may have a different set of terminals and need a different version of the file. Although it would be possible to modify the format of these files, or to rename them to include the cnode name, various programs depend on the format of the file and would need to be changed if the format or name changes. This could potentially include customer-written programs. Instead, we would like to supply a mechanism for automatically selecting the correct version of /etc/inittab based on the CPU requesting it.

To solve these problems, we have introduced a mechanism called a *context dependent file* (CDF), based in part on the hidden directory mechanism used in the Locus system developed at the University of California at Los Angeles.[1] Each cnode has a set of attributes, defined as the cnode's context. The attributes describe the type of hardware (68010 vs 68020, floating-point processor, etc.) and the cnode's name. A context dependent file consists of a specially marked directory named after the file is made context dependent. This directory is called a hidden directory, for reasons that will become obvious. Within the hidden directory are entries named after the attributes used for selecting the file. When a hidden directory is encountered during a pathname translation, the system searches the directory for an entry that matches one of the attributes of the cnode's context. If it finds one, it automatically "falls through" the hidden directory, selecting instead the matching file. An example may make this clearer.

Fig. 2 shows how /etc/inittab can be set up as a CDF. Fig. 2a shows how the file would normally appear within the /etc directory. Suppose that a cluster has three cnodes named athos, porthos, and aramis. The CDF would be set up as shown in Fig. 2b. The + after inittab indicates that the directory is specially marked as hidden. It is not actually part of the directory name. If a user on athos tries to open /etc/inittab the system will actually open the file athos within the directory. To the user on athos, the file system appears exactly as shown in Fig. 2a. The user on porthos would also see a file system that appears as in Fig. 2a, although the contents of /etc/inittab would be different. Thus, under normal circumstances, the directory is hidden.

Occasionally, the system administrator will wish to see all the contents of the hidden directory. In this case, a special escape mechanism is provided. If a + is appended to the CDF name, it will refer to the directory itself rather than falling through based on the context. Thus, a system administrator on porthos could modify the inittab belonging to aramis by editing /etc/inittab+/aramis. The pathname /etc/inittab+ refers to the hidden directory itself whereas /etc/inittab refers to the machine's own version, in this case porthos.

## File System I/O

The standard HP-UX file system buffers I/O requests to increase file system performance. The buffer cache is composed of buffer headers which contain pointers to the actual data. The buffer header data structure also contains a block number and a pointer to a vnode (a data structure describing

a particular file). The block number and vnode pointer are used to identify any block of data pertaining to the file system. When a user makes a read request to the system, the file system first checks to see if that particular block of data is already in the buffer cache. If it is in the buffer cache, then the data can be transferred to the user without incurring the overhead and time it takes to read the data from the disc drive. Likewise, if a user makes a write request, the system will buffer the data and write it to the disc at a later time. This allows the system to buffer write requests into a block size and thus minimize the number of disc writes.

This design of the buffer cache presents some problems when dealing with the discless environment. If each cnode has its own buffer cache, then there is no longer a unique buffer in the cluster's memory for a particular block on the disc. This can lead to synchronization problems. If a user on cnode A writes to a file and if a user on cnode B is reading from that same file, then the data written by cnode A may not be seen by cnode B.

This synchronization problem can be avoided by eliminating the buffer cache on the client cnodes. However, this would create performance problems. The HP-UX discless solution is to implement a compromise. Whenever possible, the discless cnode uses the local buffer cache (asynchronous I/O). When synchronization problems may arise, then the discless cnode bypasses the local cache and reads or writes directly to the server (synchronous I/O). The server always uses its buffer cache.

The determination of whether a data request should be synchronous or asynchronous is calculated on an individual file basis. Each currently referenced file in memory is represented by a data structure called an inode. Part of this data structure contains some fields called cnode maps. There is a cnode map that describes which cnodes have this file open and a reference count for each site that has it open. Likewise, there is a cnode map that describes which cnodes have this file open for write and a reference count for each cnode that has it open for write. These cnode maps are maintained on the server node only. Whenever a file is opened, the referencing cnode's identifier is added to one or both of its cnode maps depending on whether the file was opened for reading or writing. When the open condition is added to the cnode map, a file system algorithm calculates whether this file should be in synchronous or asynchronous mode. If there are no cnodes that have the file open for writing or if the file is being opened for writing and no other cnode has the file open, then the file remains in asynchronous mode. However, if opening this file in the requested mode causes more than one cnode to have the file open with at least one cnode having it open for writing, then the file is switched to synchronous mode. In switching the file to synchronous mode, the system requests that all writing cnodes flush their write buffers to the server and notifies all open cnodes that the file is now to be switched to synchronous mode. The file remains in synchronous mode until a cnode closes the file and that action causes either no more writing cnodes or there is only one cnode with the file open. A cnode using the recently closed file will be notified that it can now switch back to asynchronous mode on the next read or write request to the server.

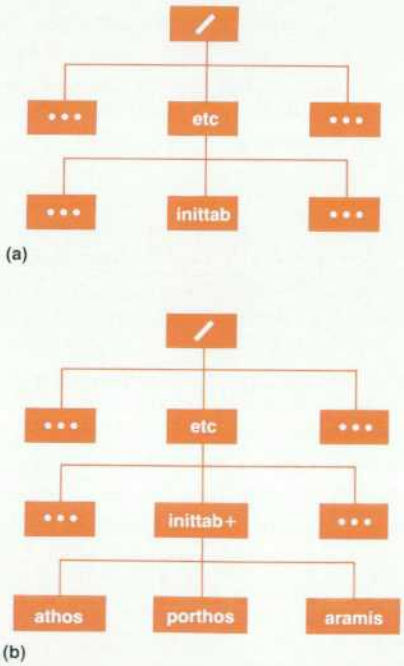



**Fig. 2.** *Example of context dependent files. (a) Directory structure for* /etc/inittab. *(b) Directory structure for* /etc/inittab *with CDFs.*

In a standard HP-UX system, the buffers associated with a file may stay in memory even after the file has been closed. Thus, if a process reopens that file and makes a read request, it can use the data that is already available in the cache. In a discless environment, this mechanism will not work. For example, suppose cnode A opens a file, reads from the file, and closes the file. Then cnode B opens the file, writes to the file, and closes the file. Now cnode A reopens the file. The buffers at site A no longer contain the correct data because cnode B has modified the data.

To take advantage of buffer caching and avoid this synchronization problem, there is now a version number associated with each file. When a file that was in asynchronous mode and has been written to is closed, the version number is changed on the server and at the cnode that closed the file. When the file is reopened and the inode is still in memory on the requesting cnode, then the old version number is checked with the current version number. If the version number is the same, then the old buffers can be used. However, if the old version number is less than the new version number, then the old buffers are invalidated.

Another consideration with buffering in a discless environment has to do with disc space allocation. In a standard HP-UX environment, when a write request is made, the system first checks to see if there is enough space on the disc for the write request. If there is not enough space left for the request, then the write fails with an error message. In a discless environment, it would help performance if each write request did not have to go to the server to ask for a disc block number. However, if it did not do this, then a user might think that a write has succeeded, but by

the time the actual asynchronous write operation goes to the server, it may fail because of no disc space. To avoid this problem, which does not occur on stand-alone systems, a nearly-full-disc algorithm has been established. The algorithm is based on knowing the number of total buffers in the cluster. Once the disc gets to the point where it does not have enough free disc space for all buffers, it notifies the discless cnodes. After this point, whenever a discless cnode makes a write request that would require space on the disc, it makes the write synchronously to the server.

### FIFO Files

In standard HP-UX, named FIFO files, also known as named pipes, are a mechanism for processes on the same machine to communicate with each other. Each process opens the same named FIFO file. Then each process uses the read and write system call to send and receive information to and from other processes. The discless implementation extends this concept so that processes on different cnodes can communicate via the same named FIFO file.

The in-memory inode for a named FIFO file contains specific fields related to that FIFO file. The specific information associated with a FIFO file consists of the read count, the write count, the current read pointer, the current write pointer, and the number of bytes in the current FIFO file. The FIFO file is maintained as a circular 8K-byte buffer. On the serving cnode, the inode contains cnode maps which specify the cnodes using the FIFO file. If only one cnode is using a particular named FIFO file, then the FIFO file specific information is maintained on the cnode that is actually using the named FIFO file. This improves performance, because the cnode does not have to communicate with the server every time it accesses the FIFO file. If another cnode opens that same FIFO file, the server recognizes that there is now more than one cnode using the FIFO file. The server then requests that the current cnode that is using the named FIFO file send all of its FIFO file specific information and data to the server and that from now on it send its read and write requests to the server. In this way, the server acts as the focal point for all communication between the cnodes.

#### Lockf

The discless implementation of file locking maintains the full standard HP-UX semantics. HP-UX provides a byte-level locking mechanism for regular files. There are advisory locks and enforced locks. Advisory locks can be used as semaphores between processes. If a file region has an enforced lock, then only the process with the lock can read from that region or write to that region.

Advisory locks are implemented with the lockf or fcntl system call. These system calls allow a user to inquire if there is a lock on the file, to test and lock a region, to lock a region, and to unlock a region. In the nondiscless version of lockf, file locks for an open file are kept in the inode structure. In a discless environment, the inode can be on more than one cnode at any given time. Thus, it must be decided where the locks will reside for a file so that everyone will know about them. One possibility is to keep all locks on the server. This is a simple implementation; however, it has the disadvantage that if a cnode has a file open with locks, then all inquiries must go to the server. The implementation that was chosen is to have each cnode keep the locks that were originated by that cnode and to have the server keep track of both the local and remote locks. Thus, if a cnode with a lock on a remote file makes a lock inquiry, the lock will be found on that cnode and it will not be necessary to send a message to the server.

If a file has enforcement mode locks on it, then each read or write system call must check to see if another process currently owns a lock in the specified read or write region. If another process does own a lock, then the requesting process must wait until the region is unlocked. When checking for other processes, it is only necessary to check on the serving cnode when a file is opened by more than one cnode and there are enforced locks on that file. The same mechanism used for keeping track of file-open requests for asynchronous and synchronous file I/O is used in this situation as well.

In the standard HP-UX version of lockf, deadlock prevention checks are done before granting a lock to avoid potential deadlocks. The basic deadlock detection algorithm is as follows. The code first looks at the status of the process that owns the lock. If the process is not waiting or is waiting for something other than a file lock, then there is no deadlock. If the owning process is waiting on a file lock, a search is initiated using the lock this process is waiting on. If the search finds the lock owned by the calling process, then a potential deadlock has been found.

In a discless environment, there are more potentials for deadlock. Therefore, the deadlock detection algorithm was enhanced to account for these situations. The differences for finding deadlocks are the result of three conditions. First, processes in the waiting chain may be distributed throughout several cnodes. Second, a process may be sleeping on a lock or may be waiting for a cluster server process on the root cnode that is itself waiting on a lock. Third, more than one process may simultaneously try to wait on a given lock as a result of concurrent deadlock searches happening on more than one cnode.

### Pathname Lookup

An important job provided by the file system portion of the kernel is the translation of a user-specified pathname into its location on the actual disc file system. For example, in the open system call, the user specifies the file name to be opened such as /dir1/dir2/dir3/file. The system then internally translates each component of /dir1/dir2/dir3/file until it has found the inode number representing /dir1/dir2/dir3/file. The system then reads this inode from the disc to determine its characteristics and the location of its data blocks. Many of the system calls pass a pathname. Examples of pathname system calls are open, creat, stat, link, and exec.

For the discless implementation, the pathname lookup code was modified. First, the code recognizes whether any component of the pathname is remote, that is, it belongs to a file system physically attached to another cnode. If the pathname is remote then the code sends the entire remaining pathname to the serving site.

To reduce the number of messages that must be communicated between the server and the requesting client, the pathname lookup code was also modified to send not

only the pathname, but also all the necessary information to complete the system call while it is still operating on the serving site. This mechanism is table driven. Associated with each pathname lookup system call there is an opcode and a structure which describe the request size, the reply size, the function on the client side that will package the required information, the function on the server side that will perform the requested operation, and the function on the client side that will unpack the request.

For example, the opcode for open is 1. Its packing function is open_pack(). Its serving function is open_serve(). Its unpacking function is open_unpack(). The function open_pack() establishes the mode to be used for opening the file and the file mode to be used if the file needs to be created. The function, open_serve() handles the requirements for the opening, such as permission checking on the file and creation of the file if necessary. The function open_unpack() allocates an inode for the file, marks it as asynchronous or synchronous, and opens the device if it is a device file.

### Interactions with NFS

In addition to the discless product, another form of remote file sharing is available with the HP-UX 6.0 system, namely NFS. NFS provides the ability to mount remote file systems. This raises a couple of questions. First, why are both NFS and the discless system needed and why can't discless be based on NFS? Second, given that both systems exist, how do they interact?

NFS is a de facto industry standard for sharing files among heterogeneous machines running different operating systems. Being general-purpose, however, it tends to impose constraints. For example, the network protocol used with NFS needs to be able to deal with routing. Also, to keep NFS simple, it does not obey full UNIX semantics. For example, it does not provide file synchronization. Finally, NFS uses a remote mount model, preventing a true single-system view. The discless system is designed for a cluster of machines with a high degree of sharing. It provides a single-system view within a cluster, but does not provide any access to machines outside the cluster. Because it has a specialized purpose, it can be optimized for that purpose. For example, because it only operates over a single LAN, it uses a very-low-overhead networking protocol with minimal need for error detection and routing. Also, the discless system maintains full HP-UX semantics including all UNIX semantics.

Since both NFS and the discless system exist within the same system, they need to coexist, preferably in a mutually beneficial manner. Indeed, each system complements the other. A cluster of workstations can replace the traditional single time-shared machine, with the workstations sharing the view of the file system, just as users at terminals on a single machine share that view. In the same manner that a user can move between terminals on a time-shared machine without noticing a difference, a user can move between workstations in a discless cluster without noticing a difference. NFS can then be used to access machines outside the cluster, just as it can be used from a time-shared machine to access other machines. To maintain the single-system view within the cluster, the NFS mounts must be global in the same manner that local mounts are: when one cnode mounts a remote NFS file system all other cnodes must see that mount also.

### Acknowledgments

Sui-Ping Chen, Barbara Flahive, Ping-Hui Kao, Curt Kolovson, and Fred Richart all contributed to the development of the discless distributed file system. Sui-Ping worked on context dependent files, Barbara worked on lockf and nonpathname related system calls, Ping-Hui and Curt worked on pathname lookup and pathname lookup related system calls, and Fred worked on mount and buffer management. Mike Berry and Fred Richart developed the distributed test tools and helped write the distributed test suites for the file system.

### Reference

1. G. Popek and B. Walker, *The Locus Distributed System Architecture*, MIT Press, 1985.

# Discless Program Execution and Virtual Memory Management

by Ching-Fa Hwang and William T. McMahon

MANY DISTRIBUTED SYSTEMS based on the UNIX® operating system offer some form of remote file access capability. However, only a few of them provide discless workstation capability, particularly in the area of paging, swapping, and execution of programs over a network. Almost all the remote file access systems assume a well-defined client/server model. Some of them have been implemented in a machine or system independent fashion and adopted as industry standards for porting to different vendors' systems. Discless workstations, on the other hand, have been offered only as proprietary systems up to this point. It is unclear at this time if any implementation will be successfully adopted as an industry standard.

The disparity between the remote file access and remote program execution implementations can be attributed to several things. Unlike the UNIX file system which has a well-defined and machine independent structure to facilitate the definition of a client/server model, the implementation of virtual memory (VM) for paging, swapping, and execution of programs over a network is not isolated from machine architecture. For example, 4.2BSD and AT&T System V are two primary bases for most vendors' UNIX implementations, but their virtual memory implementations are based on quite different machine architectures, and their performance characteristics are tuned to their native machine architectures. This creates more difficulties in defining a client/server VM model for implementing paging, swapping, and execution of programs over a network.

The key technical challenges for implementing paging, swapping, and execution of programs over a network in an HP-UX environment include: preservation of the behavior and semantics of existing program types (such as preloading versus demand paging), efficient and flexible global swap device management, and performance that is good enough to justify the cost of discless workstations. This paper describes some of the design issues and our solutions in overcoming these technical challenges.

## Overview of the HP-UX Discless Cluster

To support the one-system view, an HP-UX discless cluster has one global file system. The file system on the server node supplies all the program files that can be executed from any node (called cnode) in the cluster—as transparently as if they were executed on one system running the standard HP-UX operating system. To complete the one-system view, it should be possible to execute program files from standard HP-UX releases of the past on an HP-UX discless workstation without a recompilation. This backward compatibility applies to the various types of loading, paging, and swapping mechanisms available in the standard HP-UX environment. Loading refers to bringing a program from the file system and setting up the appropriate process control and memory mapping structure for program execution. Swapping refers to copying some of the process control structure and all the remaining pages to or from a swap disc. Paging means copying some of the referenced pages of a program to or from a swap disc.

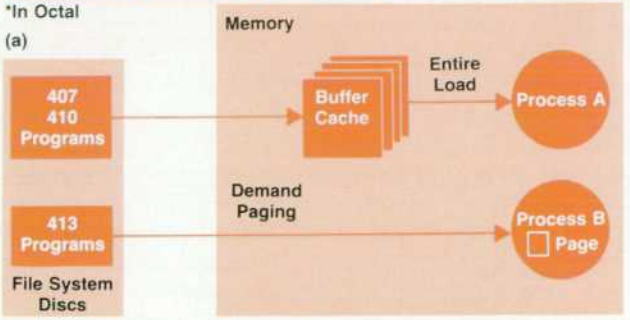
For the discussion in this paper, an HP-UX discless cluster may consist of two kinds of cnodes: (1) swap servers with local swap devices to provide swap space to their clients, and (2) swap clients without local swap discs. A swap server can also be used just like a client cnode. The swap server/client relationship is analogous to the file server/client relationship. The former describes swap space and device services and the latter file services. The common swap space pool is shared equally among all clients of a swap server (including itself as the local client). The common swap space can be expanded to multiple discs by using the HP-UX command swapon to add more swap discs. A swap client can be dynamically added and removed from the server without bringing down the entire cluster for swap space or disc reconfiguration. The server dynamically allocates swap space to its clients when needed and deallocates it when not needed. On detecting the failure of a client site, the server automatically returns the space allocated to the failed client to the common swap space pool.

The features mentioned above are considered design objectives for supporting the one-system view and achieving the high performance requirement for HP-UX discless clus-

| *Execution Type | Execution Characteristics | | | |
|---|---|---|---|---|
| | Non-sharable | Sharable | Entire Load | Demand Paging |
| 407 | X | | X | |
| 410 | | X | X | |
| 413 | | X | | X |

*In Octal

(a)




(b)

**Fig. 1.** *(a) Standard HP-UX program types and their execution characteristics. (b) Loading behavior of the different program types.*

UNIX is a registered trademark of AT&T in the U.S.A. and other countries.

ters. However, several simplifications and restrictions were placed on HP-UX 6.0. Specifically, there can be only one swap server per cluster and the swap server must be the same cnode as the root/file server. As an option, a nonroot cnode is allowed to have local swap discs for improving local swapping performance. In this instance, only the specific cnode has access to the local swap disc.

## Program Execution

To help understand the complexity of handling program execution and the interactions between the file system and virtual memory in a discless environment, we use the following scenario to illustrate the interaction in a stand-alone standard HP-UX environment. The scenario describes what happens externally in the user environment and internally in the system when an application program called foo is updated. In the rest of this paper the term "update," when used in the context of program or executable files, refers to the point at which an existing executable file is replaced with a new version of the program.

To begin the scenario, a programmer has just completed a new version of foo with the file name of foo.new, and is ready to release it while some users may still be in the midst of executing the old foo. Internally the system may have kept the program data and control information in several places, depending on the exact stage of execution. For example, the program file in the file system on disc may or may not have been fully brought into memory for execution, and part or all of the program may have been paged or swapped to a swap disc to free the memory for other process executions.

To release the new foo, the programmer types in a mv foo.new foo command. The HP-UX system will detect that the program is currently busy for execution and therefore reject the command by returning an error (ETXTBSY). Not until the last user has completed the execution of foo will the programmer be allowed to update foo. When the system detects that no other user is executing foo it will honor the mv command by copying from foo.new to foo as a normal file system operation. While doing this, the system will also invalidate all the memory pages that may still have cached data of the old foo. If a user tries to execute foo before the mv replacement operation is finished, the system will prevent execution since foo is in an inconsistent state.

In the HP-UX discless cluster, our goal was to preserve the features described in the scenario to support the one-system view and to satisfy performance requirements. This required us to consider that a program being executed may have parts paged or swapped out to the swap discs of the swap server or cached in the memory of different cnodes. In addition, the program may be requested for update; therefore, it was necessary for us to consider mutual exclusion as explained later. Many of the standard HP-UX internal mechanisms and algorithms are not adequate for handling these situations, so enhancements and new algorithms were added to handle the discless environment.

### Program Loading/Swapping/Paging

In the standard HP-UX system, when programs are compiled or loaded (through cc or ld commands) the control options 407, 410, or 413 (octal) can be included in the command string to designate the loading, swapping, and paging characteristics of the program. A 407 program requires each process invocation to have its own copy of the program in memory and not be shared by multiple processes. 410 and 413 programs can be shared among multiple processes to save memory space. 407 and 410 programs need to be loaded in their entirety from the file system before the execution can begin, while a 413 program can be loaded in by pages on demand (i.e., page fault). The loading of a program file in its entirety from the file system for execution is handled through file system I/O via the buffer cache, while the paging activities, either with the file system discs or the swap discs, use device I/O directly, bypassing the file system and buffer cache (see Fig. 1).

To maintain the identical behavior and semantics of the three program types for backward compatibility and performance, the discless file system provides a mechanism for bringing in remote program files from the file server for execution. We also implemented a remote device access mechanism that allows devices at a remote cnode, or the device server, to be accessed over the network. This remote device mechanism provides the necessary mechanism for handling paging I/O directly with either the remote file system discs or the remote swap discs. These two mechanisms provide a means for loading and/or paging the three program types.

### Mutual Exclusion with File Update

In the standard HP-UX system, executable files are usually in one of two mutually exclusive states: update and execute. A file can be brought from disc to memory for either updating or execution by one or more processes, but never for both updating and execution at the same time. However, a file can still be opened for reading while in either state. Before allowing the execution of a program, the system internally checks that no process is opening the file to write to it. Likewise, when a process is ready to open the file for writing the system also checks to see that the file is not being executed by any process before it enters the update state.

In a discless environment maintaining mutual exclusion is more complex because the processes that want to execute

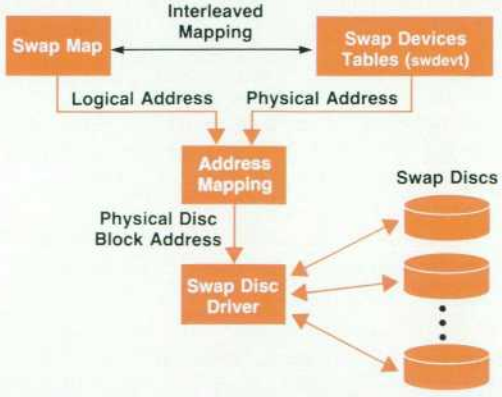



**Fig. 2.** *Mapping from logical address to physical disc blocks on the swap discs in standard HP-UX.*

or update the file may come from multiple cnodes in the cluster. Therefore, mutual exclusion must be enforced in the context of the entire cluster. To address this issue, the root server was selected as the place to enforce and coordinate mutual exclusion for the files it serves. For each file being referenced or executed, the file inode, which is an internal data structure containing a description of the file, contains entries called cnode maps. The cnode maps are used to track program execution and program file updates. The cnode map for execution keeps track of the cnodes in the cluster executing the program and keeps a reference count of the number of instances of execution of a particular program at each cnode. The execution cnode map and the write (update) cnode map together provide the root server with the necessary information to enforce mutual exclusion. For more information about the inode and cnode maps for file updates refer to the article "A Discless HP-UX File System" on page 10.

### Client Caching for Performance

In the standard HP-UX system, caching for program execution is provided to improve execution performance. When a process is terminated or when its pages are paged or swapped to a swap device, the memory pages are freed but also marked as reclaimable. This denotes that these pages can be reactivated if the data on the pages is referenced again before the pages are allocated to other programs. Like the buffer cache for minimizing file system I/Os, reclaimable pages are intended to minimize I/O overhead for paging and swapping. The effect is especially significant when a file is repeatedly executed by one or more processes.

To maintain cache consistency, when a program file is updated, the system automatically invalidates the file's reclaimable cache pages left from previous executions. This ensures that no future executions of the same file will get out-of-date data from the reclaimable cache pages. Similarly, when a file is to be executed by a process, any file data remaining in file buffers resulting from delayed or asynchronous file system writes will be flushed to disc first. This is necessary to ensure that when the program is paged in directly from discs, the file on the disc is up to date.

The standard HP-UX system keeps cached data around as long as possible. Flushing file buffers or invalidating reclaimable pages is always delayed until it is absolutely needed to maintain system consistency. This is the kind of optimization policy that we wanted to keep for HP-UX discless clusters. However, cache consistency in a discless environment is much more complicated than in the standard HP-UX system because buffers for file updating can potentially exist on multiple cnodes, and reclaimable pages for an executing program file can also exist on multiple cnodes. To maintain cache consistency in a discless environment we had to ensure that:

- For program file updates, all reclaimable pages for a particular file are invalidated throughout the entire cluster.
- Before a 413 program enters the execute state, all the file buffers associated with the program file are flushed over the network to discs at the server.

To improve performance further an extension is included in our reclaimable page invalidation mechanism. When a file is updated, instead of starting a global operation to invalidate all the reclaimable pages on all cnodes, the invalidation is individually handled and deferred for each cnode until that cnode is ready to execute the file again. Basically, we include a version number in the in-memory inodes at both the server and the clients. The version number is incremented in the inode at the server whenever the file is closed for update, but is incremented at a client only when the client is ready to access the file. When a client is about to execute a file, the version number of the file at the server is compared with that at the client. Only when the two numbers are identical will the local reclaimable pages at the client cnode be kept for possible reuse. All the other cases will cause these reclaimable pages at the client cnode to be invalidated.

## Swap Space Management

In the standard HP-UX system information about swap discs is set up at boot when the system is reconfigured, and is kept in the swap device table (swdevt). By default, swap space is first set up on the root disc and other discs can be added by the use of the swapon command. The information in the swdevt data structure is used to build the system swap map which is used to represent and keep track of the pool of available swap space (see Fig. 2). Swap space is interleaved among all the swap discs, and when swap space is allocated, the first chunk of swap space is taken from the first swap disc in the swdevt, the second chunk of swap space is taken from the second swap disc, and so on.

When a process needs swap space it grabs it from the system swap space pool. When a page is paged out to the disc the logical address in the swap space pool is mapped onto a physical disc block using the information in the swap devices table. When a process requests swap space

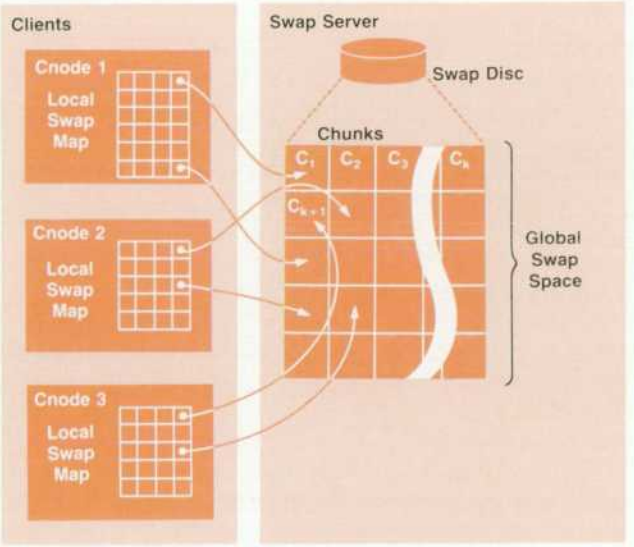



**Fig. 3.** *The swap space is physically located on the server and is allocated to the client cnodes in chunks.*

and there is no more space in the swap space pool the process is killed or an ENOMEM error is returned to the process.

### Design Considerations

One of the primary considerations in designing the remote swap mechanism for the HP-UX 6.0 system was to maintain as much of the current interface as possible. We wanted the swap device table to continue to specify the swap discs, and the use and availability of swap space to be represented by a swap map. Efficiency was important, so we had to consider methods to minimize the number of requests for swap space made from a client cnode. If the swap maps were maintained only on the server all requests for swap space would be a remote request, whereas if each cnode maintained its own swap map it would only need to make a remote request when its local swap map indicated that it was out of swap space. For these reasons two new concepts were introduced: the global swap space and the local swap space. The global swap space represents the total amount of swap space that is allocated to all clients and exists on the root server. The local swap space is the portion that is allocated to a particular cnode. Each cnode has a local swap map which is used to map from the local swap space to the global swap space (see Fig. 3). Efficiency was also considered in determining the granularity of requests for swap space. In the HP-UX 6.0 system we adopted a wholesaler/retailer allocation scheme. The swap server, functioning as a wholesaler, allocates the swap space in large chunks (in megabytes) to its clients; each client in turn allocates the space (in tens or hundreds of kilobytes) to each local process.

Another must objective was to allow dynamic reconfiguration of the cluster without bringing the entire cluster down. We did not want the cluster to be wasteful of space, and a fixed, permanent allocation of swap space to every cnode would have been very wasteful. This meant that allocation of swap space to a client had to be dynamic and that swap space had to be returned when not used or when a cnode crashed or rebooted.

We wanted to provide a way to limit the amount of swap space a cnode can consume and also to provide a way to ensure that a minimum amount of swap space was always available. Other design questions that were raised but not implemented in the HP-UX 6.0 system were whether one cnode should swap to more than one swap server and whether a cluster should support more than one swap server.

### Local and Global Space Mapping

To understand how paging and swapping works in a discless environment it is necessary to understand the division between local and global swap space, and how the local swap space maps into the global swap space. A new data structure called a *chunk map* was established to represent the global swap space. The chunk map exists on the server only and its size and initial swap space information are derived from the swdevt at the server as had been done with the swap map in the standard HP-UX system. Each entry in the chunk map represents one chunk of swap space on the disc. Mapping from chunk map address to disc block is done by using the swdevt as it is done in the standard HP-UX system when mapping from the swap map to the disc block. The information maintained in the chunk map consists of the chunk size, the number of the cnode owning the chunk, a bit to indicate if the chunk is valid, and a bit to indicate if the chunk is allocated (see Fig. 4). The sizes of all chunks are defined by a system global variable called dmmax. The number of the cnode that owns a particular chunk is kept as a sanity check and for crash recovery. The valid bit is set when a swapon is done and the chunks on the new swap disc become available for swapping. The allocated bit is set when a chunk is assigned to a cnode.

It is still necessary to complete the mapping from local swap space to global swap space. This is accomplished through the use of a data structure called a *chunk table* which exists on each client cnode. Since the swap discs are located at the server the chunk table acts as a logical swdevt for the client cnodes and provides the first step in the mapping from the local swap map to the global swap space on the server (see Fig. 5). Each entry in the chunk table represents one chunk of space in the local swap map. Conversion from a logical swap address to a chunk table entry is done by dividing by dmmax. Each entry in the chunk table contains the chunk size, a chunk map index, a valid bit, and a reference bit. The chunk size is the same as the size of the corresponding chunk map entry. The chunk map index is a pointer to the corresponding entry in the chunk map on the server. The valid bit indicates if the entry is valid (and hence represented in the local swap map), and the reference bit indicates if it is being used by any process. The chunk table is maintained at every cnode including the swap server. The chunk table allows the swap map to be more easily maintained at each cnode. The reference bit provides the information to return swap space from a cnode that is not using it.

The mapping from a client's local swap map to disc blocks on the server can be briefly summarized as follows. When a request is made from a client cnode to send some data to swap, the system converts the logical address in the local swap map to an index into the chunk table (divi-

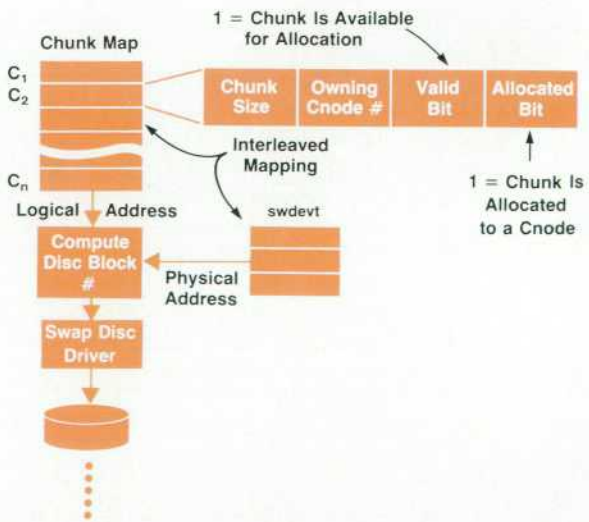



**Fig. 4.** *The chunk map entry.*

sion by dmmax) and an offset (the remainder). The entry in the chunk table yields the index of the chunk map entry at the swap server. This index and the offset are sent over to the swap server along with the data. On the swap server the system takes the chunk map index and offset and uses the rules for interleaving swap space and the information in swdevt to generate the disc block number for the request. The request is then sent to the disc driver for that device.

### Allocating Swap Space

When a process requests swap space and does not find any in its local swap map the process is not immediately killed or an error returned. Instead, the process makes a remote request to get more swap space from the global swap space pool. The swap server allocates another chunk to this cnode and the cnode adds an entry for this chunk to its local swap map and grants the process's request.

When a process makes a remote request for another chunk of swap space it goes to sleep at that point. It is then possible for another process to come along and request a piece of swap space and go to sleep. This could result in much more swap space being requested than is needed because client cnodes allocate swap space to local processes in sizes that are much less than the chunk sizes from the server (i.e., 10 or 100 kilobytes versus several megabytes). To prevent this from happening, a lock was introduced to serialize the requests. When the first remote request is made the lock is grabbed and is not released until the additional swap space is added to the local swap map. When each of the other processes acquires the lock, each one reevaluates whether there is sufficient swap space available.

### Returning Swap Space

One of the design decisions was to allow a cnode to return swap space that it is not using. However, it is not efficient for a cnode to return swap space immediately since it may just turn around and request more space. To prevent this type of thrashing, the reference bit and a daemon process are used to check for unused chunks. If an unused chunk is found, the reference bit is cleared. On the next invocation of the daemon, if the chunk is still unused, then it is returned to the swap server; otherwise the reference bit is set. The daemon is set to run once every 30 seconds, so unused swap space is returned between 30 seconds and one minute after it becomes unused.

When a cnode is removed from a cluster either by crashing or by being rebooted, it is necessary to return the swap space to the swap server. This happens when recovery is done for that cnode on the swap server. Recovery is very simple for swap space. When recovery is conducted on the swap server the system routine simply goes through the chunk map, and when it finds an entry that was allocated to the crashed cnode it marks that entry as being available again (allocate bit = 0). Crash recovery is discussed in the article "Crash Detection and Recovery in a Discless HP-UX System" on page 27.

### Controlling the Amount of Swap Space

Two configurable parameters are provided for controlling the amount of swap space allocated to a cnode. These are MINSWAPCHUNKS and MAXSWAPCHUNKS. MINSWAPCHUNKS

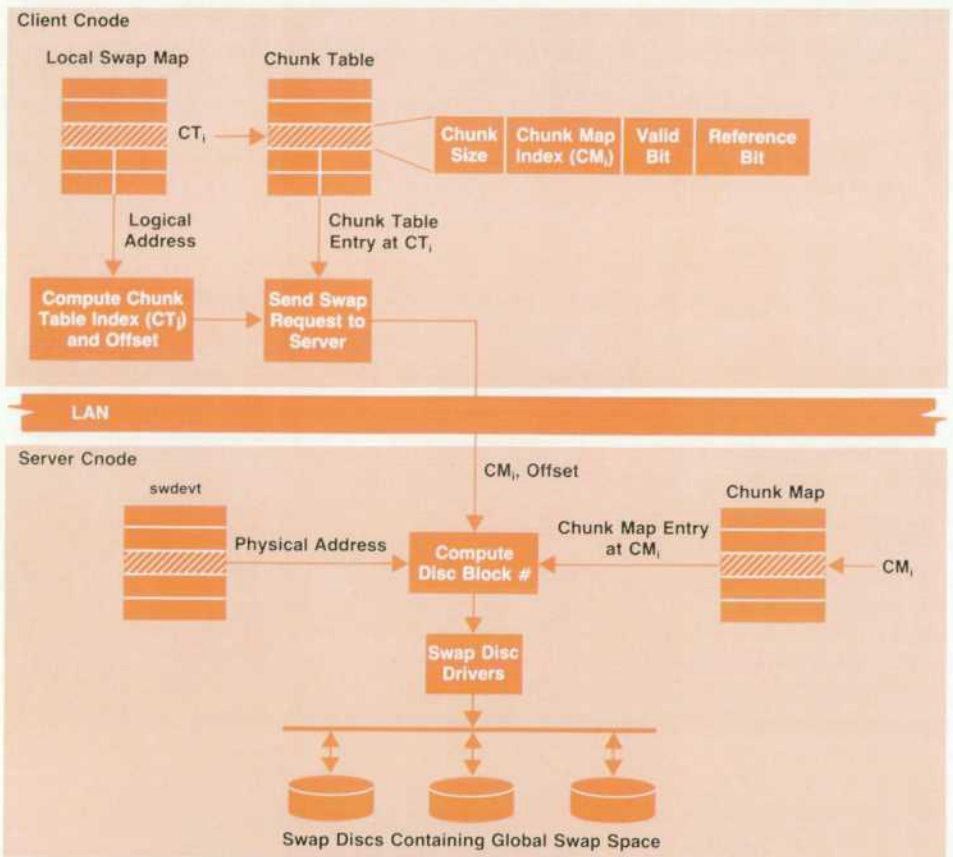



**Fig. 5.** *The relationship between the local swap map and the chunk table on a client cnode and the chunk map and swap device table on the server.*

specifies the minimum number of chunks of swap space a cluster can have even when the space is not actively used. It is the amount requested at boot and it is never returned. It ensures that a particular cnode will have at least that amount of swap space. MAXSWAPCHUNKS specifies the greatest number of chunks of swap space that a cnode can ever have.


## Summary

In summary, the HP-UX 6.0 system provides a fairly complete implementation for HP-UX discless program execution and virtual memory management. Among the features provided to this end are backwards compatibility for executable files, remote swap services, and HP-UX semantics for executable files. New mechanisms are included to minimize performance degradation over a network.


# The Design of Network Functions for Discless Clusters


**by David O. Gutierrez and Chyuan-Shiun Lin**


WITHIN AN HP-UX DISCLESS CLUSTER, the kernel of the client and server machines uses a simple, fast, and reliable network protocol to communicate through a single IEEE 802.3 10-Mbit/s local area network (LAN). The discless protocol is based on the request/reply model and its interface to the HP-UX operating system is specially tailored for efficient data transfer. To become a viable product, a discless system must provide a level of performance comparable to that of systems with local discs. Measurements on HP 9000 Model 350s show that remote file I/O throughput performance of the HP-UX 6.0 discless implementation using an HP 7958A Disc Drive is 91% of stand-alone performance in read operations and 87% of stand-alone performance in write operations when transferring large files. This performance level is achieved by a low-overhead network protocol, efficient network buffer management, cluster server processes, and carefully implemented read/write algorithms.

A cluster consists of a single file server and a number of discless client machines connected by a single LAN cable or several cables connected by LAN bridges, hubs, or repeaters. Multiple clusters may exist on the same cable. Each node of a cluster is called a cluster node (cnode) and has its own hostname and internet address. The central file server is called the root server (shortened to server in this paper) and is where all file systems and disc storage reside. The operating system software is designed to handle up to 255 client machines and each cnode is assigned a number from 1 to 255. The kernel network functions map the cnode number to the appropriate source/destination address.

The discless network protocol is designed specifically for the HP-UX discless kernel and not for general-purpose network communication. Experimentation indicates that packet loss on a single local area network is rare, and by limiting the design scope to providing intracluster network service on a single local network, we can function with a simple network protocol. The discless network functions and the kernel functions are closely tied together to minimize the path length for sending and receiving messages. General-purpose networking services are still available throughout the discless cluster to provide standard communications with the outside world.

A major source of communication overhead in normal network operations is copying data between network buffers and user buffers. It is important to minimize such copy operations. In most network systems, messages are copied from an operating system buffer into network software buffers and then into the network I/O hardware buffers. In the

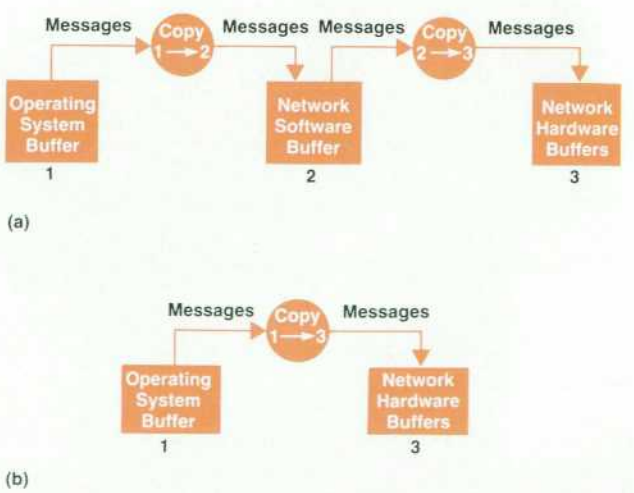



**Fig. 1.** *Reducing network communication overhead. (a) Typical message copying scheme for network communication. (b) Discless implementation.*

HP-UX discless implementation, data is copied directly between operating system buffers and hardware buffers, thus eliminating one level of copy operation. These different copying schemes are illustrated in Fig. 1.

Performance measurements on a pair of HP 9000 Model 350 client/server machines indicate that our simple discless protocol and the buffering scheme have enabled us to meet the established performance goals for discless machines. To understand the distribution of overhead among the operating system and network functions, the kernel was instrumented, and the processing time spent in the kernel and network functions on the server during read and write operations was measured. To compare the implementation of two distributed systems on the same machine the performance and overhead profile of the Network File System (NFS) functions was also measured under the same conditions. NFS provides transparent access to remote files in a heterogeneous network. The performance results are discussed on page 24.

### Overview of the Network Functions

The network functions are designed only for the discless kernel, performing intracluster kernel-to-kernel communications and linking the client cnodes with the disc facilities on the server. Users can still use the general-purpose network facilities such as the ARPA/Berkeley services, HP's NS network services, and NFS to access resources both within and outside a cluster. Since the discless cluster provides a single-file-system view, users have no need to use any of the general-purpose network functions such as ftp, rcp, or NFS to access resources within the same cluster. Intracluster remote process execution can be achieved by using the functions remsh, rlogin, or rt.

The discless network protocol coexists with other general-purpose protocols. The discless messages conform to the IEEE 802.3 link level protocol header format. Fig. 2 shows the relationship between the discless network protocol and the NFS protocol stack. The discless implementation and other general-purpose protocols share the same network hardware and device driver. Sitting above the driver level, the discless network functions are completely independent of other network functions. The network functions use the cnode number and the link level address of the LAN card for source/destination addresses. The mapping of cnode number to the link level address of each cnode's LAN card is kept in the root server's cluster configuration file. If a machine has multiple network I/O cards, only one can be used for discless communications.

In the current implementation, the client and server machines are all HP 9000 Series 300 machines. Therefore, the protocol does not need to translate the data representations from one machine's format to another. The design is extensible to accommodate heterogeneous machines within a cluster.

### Discless Message Interface Functions (DM Layer)

The discless message interface functions provide the interface between the HP-UX kernel functions and the HP-UX discless protocol. To send a message, an HP-UX kernel function (e.g., read or write) sends a request to a network function called dm_send. The parameters for dm_send include the destination cnode number, the message buffer, an optional outbound data buffer, an optional inbound data buffer, a set of control flags, and the function to be called when the reply is received back at the client. Fig. 3 shows the activities involved in processing a message at the client cnode and at the server, using the discless request/reply protocol. These activities and the discless protocol are described in the following sections.

**Message Buffers.** A network message may contain a small message buffer and, optionally, a large data buffer. The message buffer holds the commands and their associated parameters. If the request/reply message includes a large data block (e.g., a file system block), then the data buffer is used. The discless buffer management functions maintain their own pool of message and data buffers.

Before a kernel routine can send a request message it must allocate a message buffer. The discless buffer management functions provide facilities for allocating a buffer chain depending on the size of the request, and for filling the buffers with commands and parameters. Buffer management functions are discussed in detail on page 23.

**Inbound/Outbound Data Buffers.** When a file block is written to the server, the kernel write function includes the file system buffer as the outbound data buffer in the send call. For a file read the kernel read function will preallocate the file system buffer for receiving the remote file block and include the file buffer in the send call's inbound buffer parameter. This guarantees that reply messages will not be lost or delayed because of buffer shortage problems.

**Control Flags.** The control flags contain the information that enables the discless protocol functions to determine the protocols for delivering the request messages. For instance, a client may wait (go to sleep) for a reply or continue without waiting, thus enabling the discless protocol to sup-

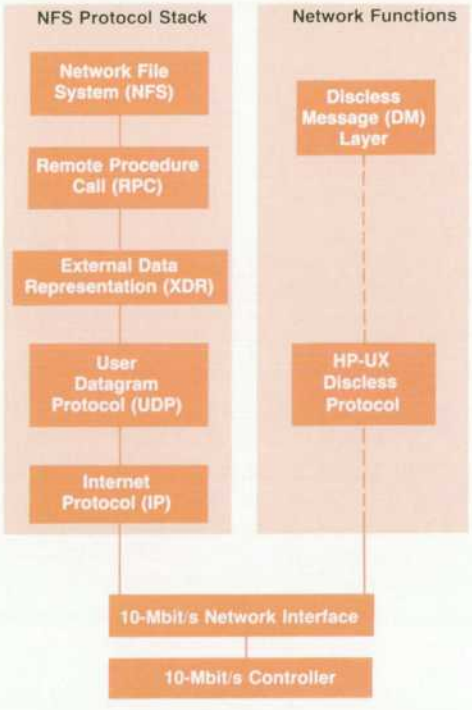



**Fig. 2.** *NFS protocol stack versus HP-UX network functions.*

port asynchronous or synchronous-mode I/O operations when accessing remote files. The client can also specify if the request is idempotent (repeatable) or nonidempotent. Idempotent messages and discless protocol are discussed in more detail in the next section.

**Arrival at the Server.** When a request message arrives at the server, the request is either processed as part of a network interrupt service function or by a server process, and then the kernel file system function specified in the command buffer is invoked and the request is executed. After the request is processed, the server calls a reply routine to send back to the client the status and, optionally, the selected file block.

**Return to the Client.** When the reply message arrives back at the client machine, a network function directly copies the reply messages from the LAN card buffer to the preallocated receive buffers. After the reply message is fully reassembled in the receive buffers, a network function wakes up a sleeping request process (if it had been placed in the wait state), delivers the reply message, and returns to the kernel function that made the send call. For asynchronous operations, the network also releases the request/reply messages automatically. The control flow diagrams in Fig. 4 show the flow of messages between client and server.

### Discless Network Protocol

The discless network protocol is based on a simple request/reply model. For each request message, the serving cnode sends back a reply message to the requesting cnode. There is no acknowledgment for receiving the request; instead, the reply is used as the acknowledgment. On an IEEE 802.3 network the maximum packet size is 1514 bytes, and it may take up to 6 packets to transmit a maximal-sized discless message (a 1K-byte message buffer plus an 8K-byte data buffer). Since messages are rarely lost in the local area network, the protocol does not need to acknowledge each individual packet. By using the reply message as the acknowledgment for the multipacket messages, it reduces a significant amount of overhead in protocol handshakes used to prevent message loss during transmission.

As mentioned above, there are two types of messages: idempotent (repeatable) and nonidempotent. These message types determine the sending/receiving request/reply protocol between the client and server cnodes. Fig. 4 shows the relationship between these two types of messages. For idempotent messages the requesting cnode continues to transmit requests until a reply is received from the server. The nonidempotent requests are processed by the server exactly once and the server repeatedly transmits replies until the client acknowledges the reply. This protocol provides excellent performance and helps handle lost messages.

For idempotent messages, when the reply is sent to the client, the server releases both the request and the reply messages. If the reply is lost, the client machines will retransmit the request and repeat the request/reply cycle again. If the same request arrives at the server just after the requested operation is finished but before the reply is returned to the client the request is considered a duplicate

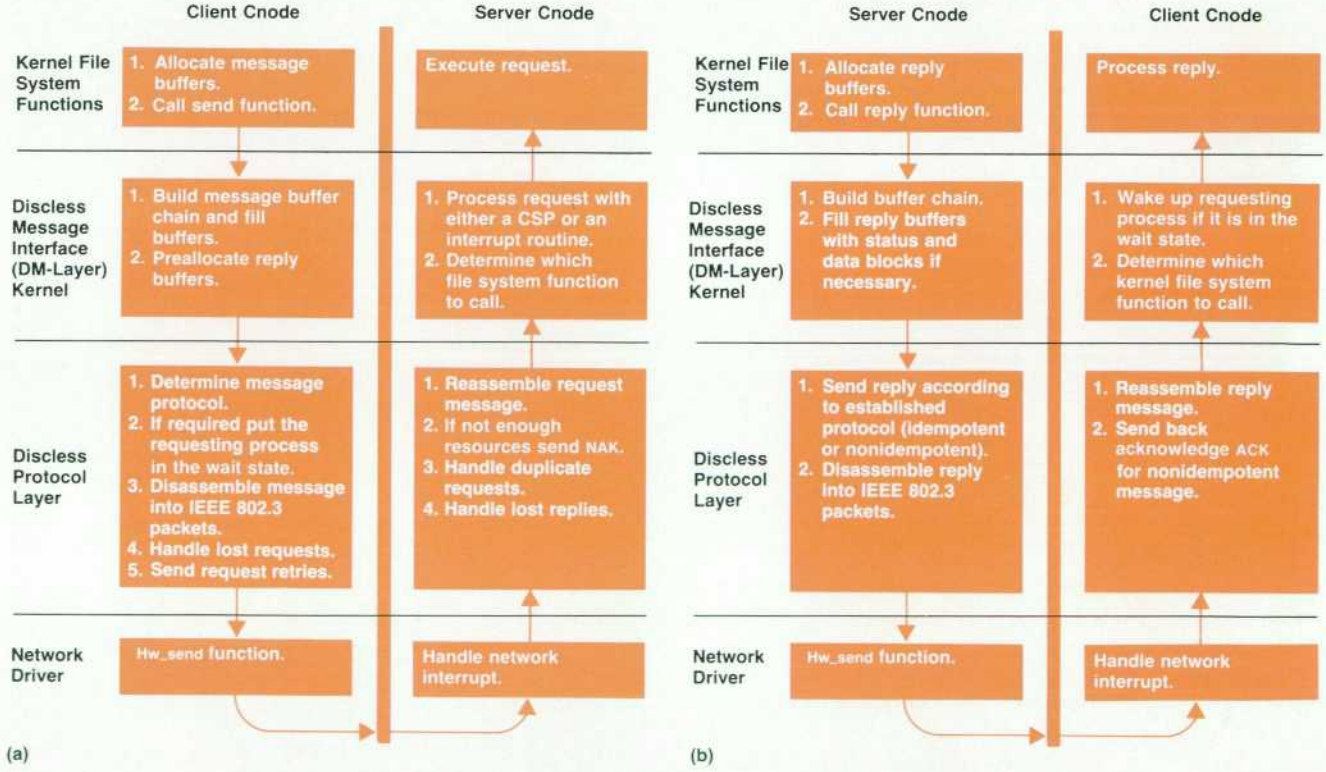



**Fig. 3.** *Network control flow. (a) Originating a request from the client and receiving a request at the server. (b) Sending a reply from the server and receiving a reply at the client. CSP = cluster server process.*

and is ignored. For the nonidempotent requests the server cannot release the request and reply messages until the client acknowledges that the reply has been received.

Some requests take an indefinitely long time to complete—for example, reading from or writing to a locked file. A request of this type (called a slow request) is indicated by the arrival of duplicate requests before the operation requested is finished. To eliminate the unnecessary retries, the serving cnode sends back a special acknowledgment to the requesting cnode whenever a slow request is detected. Upon receiving such an acknowledgment, the requesting cnode stops retrying. Since the client stops retransmitting slow requests, and the reply could be lost, we use the nonidempotent request protocol to handle the reply messages of slow requests.

To handle lost message problems, the requesting cnodes retransmit the request forever at intervals of 2, 3, 4, 5, ..., 5 seconds until the reply is received. Similarly, the server cnode retransmits the reply messages of nonidempotent requests every half second until the acknowledgment is received. The retries continue until the crash detection function detects that the destination cnode is down and aborts the requests and retries.

Messages can be lost in a receiving cnode for two reasons: the LAN card's I/O buffer has overflowed or there is a shortage of discless networking resources such as networking buffers, data buffers, or protocol table slots for keeping track of messages. Since the resources for the reply messages are preallocated, the lost message problem will not happen to the requesting cnode. To prevent excessive message loss on a heavily loaded server, the serving cnode sends back a special negative acknowledgment message (NAK) to the requesting cnode whenever it fails to allocate any network resource for a new request. The message sending functions on the requesting cnode delay sending new requests or retries to the server cnode when a NAK message is received.

The discless request/reply protocol model provides a reliable message delivery mechanism. However, some discless kernel functions, such as crash detection and distributed clock synchronization, only need quick access to the network without the request/reply protocol. These functions are not concerned with the lost message problem. To support such a requirement, the discless protocol provides a datagram service, which bypasses the request/reply protocol and directly calls the network driver to transmit a datagram over the network.

### Discless Networking Buffer Management

A design goal of the discless implementation was to achieve the highest level of distributed intracluster communication possible. Efficient networking buffer management is critical to achieving this goal. Copying network data is an expensive operation and manifests itself in various places. The protocol efforts would suffer if the overall implementation did not address and attempt to minimize data copying operations.

The first problem to overcome occurs when the server receives a write request containing a large data block. In this case a file system buffer is required, but the standard HP-UX file system buffer pool cannot be accessed by discless network functions during an interrupt. To solve this problem, a separate pool of data buffers called fsbuf was implemented for the discless system. This resource is similar to the standard HP-UX file system buffer pool except that it is available to discless network functions during an interrupt. An fsbuf is used only on a serving cnode, and the pointers for the file system and fsbuf are identical. Therefore, when the server starts processing a write request we can simply switch the fsbuf and file system buffer pointers, instead of doing a buffer-to-buffer copy.

Double-buffering and data copying operations need to be minimized for discless buffer management. Towards this end, the LAN device driver has two special, discless specific functions that provide the necessary support. These functions allow the protocol layer to copy data directly between the file system buffers and the LAN card's hardware buffers, eliminating intermediate buffering operations. The first function handles inbound messages resulting from read requests, and the second function handles outbound messages resulting from write requests. For read requests a file system buffer is preallocated before generating the request to handle the reply data. This helps to minimize delays in processing reply messages during an interrupt.

Besides fsbuf, two other data structures used for discless buffer management are mbuf and cbuf (collectively called network buffers). The number of these buffers is set at boot. These resources are the fundamental set of data structures used for all discless messages. The data structure mbuf was originally developed by the University of California at Berkeley as a general-purpose buffer management mechanism. The complete Berkeley design was deemed unnecessary for the more limited discless situation. The discless mbuf/cbuf is superficially similar in many respects to the Berkeley design but is implemented and used in a different manner. The networking buffers encapsulate the request/reply message, which contains the commands and their associated parameters. The mbuf is relatively small (128 bytes each)

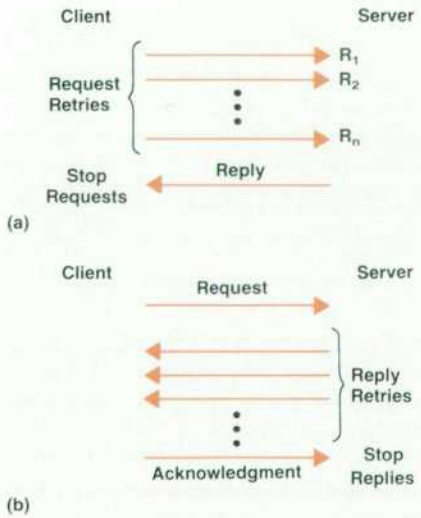



**Fig. 4.** *The relationship between idempotent and nonidempotent messages using the request/reply protocol. (a) Idempotent. (b) Nonidempotent.*

and sometimes cannot hold sufficient information. In these situations, the cbuf (1024 bytes each) is used and mbuf will contain a pointer to the allocated cbuf. For requests that involve data blocks for reading or writing, the mbuf/cbuf combination will contain an optional pointer to the fsbuf for the data block.

### Discless Cluster Server Processes (CSP)

There are many existing networking models that deal with interprocess communication in distributed environments. A common paradigm for such applications is the server/client relationship. In this model a daemon process normally listens at some well-known address for requests. Upon receiving a request for service, a daemon process forks an image of itself to handle the request. Meanwhile, the original parent server process resumes listening for additional connections. Forking the new child process and context switching from the user to the system environments are expensive operations, and would be unacceptable for handling requests in a discless environment.

The discless implementation is also based on the server/client model, but a different method of handling requests was developed. The serving cnode maintains a small dynamic pool of kernel processes called general cluster server processes (GCSPs). GCSPs avoid the overhead of forking and still satisfy requests from the client cnodes in the cluster. These processes can be quickly and efficiently created, destroyed, and context switched. When a request arrives at the server a GCSP is allocated from the pool to handle the request. When a GCSP is run it is locked into memory and cannot be swapped out. GCSPs are created from user space when the server is initially configured as the cluster server. The number of GCSPs is also determined at this time.

**Special Types of CSPs.** In general, the discless cnodes do not require GCSPs. However, certain operations must be performed by the discless cnode to maintain its membership in the cluster, such as synchronizing the mount table or converting to synchronous I/O on a file. The limited capability CSP (LCSP) solves this problem. This solitary LCSP is created when a cnode first joins a cluster.

Most client requests are handled by the GCSPs. However, in a few instances the server needs to run user-level code to service a request. For example, the server needs to read the file-system-resident cluster configuration file to determine whether to grant a client's request to join the cluster. In such cases the server dynamically creates a user-level cluster server process (UCSP), services the request, and then exits. Currently, only a single UCSP has been implemented, and it is is used to read the cluster configuration file mentioned above.

**Slow or Indefinite Operations.** Certain operations may take an indefinite amount of time to complete. For example, a read from a FIFO file could wait indefinitely. During this waiting period, a GCSP is tied up. If all the GCSPs were used in such a manner, there would be none available to service other requests and the system would come to a halt. One solution would be to dedicate certain GCSPs to slow operations while others would be available for the fast operations. In practice, it is difficult or impossible to determine in advance whether an operation will be fast or slow. For example, it is impossible to distinguish between a write to a FIFO file and one to a disc without examining the message and various file system data structures that may be implicitly referenced. Even if it can be determined that the access is to a fast device like a disc, the operation could still take a long time because of system calls like lockf() and other system interactions. For the discless implementation, a more reliable mechanism was chosen. Whenever a GCSP is invoked it sets a time-out, the duration of which is dependent on the number of free GCSPs. If the GCSP completes its assigned task before the time-out period, it clears it. Otherwise, a replacement GCSP is created and the slow GCSP is set to terminate itself upon completion. This ensures that an adequate pool of GCSPs is maintained.

### Performance Measurement Results

The environment chosen for evaluating the discless implementation for this paper was a completely isolated two-cnode discless cluster. Both the root server and the discless cnode were identically equipped, 16M-byte HP 9000 Model 350 Computers. The server used an HP 7958A Disc Drive for both the file system and the swap areas. After the discless cnode was remotely booted over the LAN, all unnecessary processes were killed on both machines. The root server had four GCSPs running and standard default-configured kernels were used. Both systems were rebooted after every benchmark to ensure that no data was cached from a previous run that might affect the current benchmark.

**Sequential Write Statistics (10M bytes transferred)**

| Block Size (K bytes) | Throughput* (K bytes/s) | % of Stand-Alone | Protocol Path |
|---|---|---|---|
| 16 | 423.57 | 100% | Stand-Alone System |
| 8 | 423.14 | 100% | |
| 4 | 420.76 | 100% | |
| 1 | 401.82 | 100% | |
| 16 | 365.13 | 86.20% | Discless Protocol Path |
| 8 | 354.80 | 83.80% | |
| 4 | 372.84 | 88.60% | |
| 1 | 359.47 | 89.50% | |
| 16 | 77.51 | 18.30% | NFS Protocol over an NFS Mount Point. |
| 8 | 78.06 | 18.45% | |
| 4 | 45.30 | 10.80% | |
| 1 | 43.63 | 10.90% | |

*All throughput numbers are the result of averaging 10 data transfers.

**Sequential Read Statistics (100M bytes transferred)**

| Block Size (K bytes) | Throughput (K bytes/s) | % of Stand-Alone | Protocol Path |
|---|---|---|---|
| 16 | 428.33 | 100% | Stand-Alone System |
| 8 | 426.10 | 100% | |
| 4 | 428.30 | 100% | |
| 1 | 428.52 | 100% | |
| 16 | 390.52 | 91.17% | Discless Protocol Path |
| 8 | 388.67 | 91.22% | |
| 4 | 388.71 | 90.76% | |
| 1 | 387.47 | 90.42% | |
| 16 | 315.43 | 73.61% | NFS + Discless Protocol over an NFS Mount Point. |
| 8 | 312.81 | 73.41% | |
| 4 | 309.06 | 72.16% | |
| 1 | 299.46 | 69.88% | |

**Fig. 5.** *HP-UX Series 300 release 6.2 discless versus NFS throughput statistics. (a) Throughput for sequential writes. (b) Throughput for sequential reads.*

Two benchmarks were run from the discless cnode. The first benchmark performs repetitive sequential reads, using various block sizes, of a 10M-byte file located on the root server's file system. The file is read ten times, resulting in a total of 100M bytes transferred. The second benchmark performs sequential writes, using various block sizes, to a file on the server's disc. The writes continue until a 10M-byte file has been written. For both benchmarks the user process allocates an incore buffer of the specified block size. This buffer is repetitively read into or written from. No intermediate files are created by the benchmarks.

For comparison, the benchmarks were run in completely discless and discless-plus-NFS environments. The NFS environment was established by mounting from the discless client the root server's test and data directories across an NFS mount point. By executing the benchmarks with the source and target files crossing the NFS mount point, NFS protocols are used and a clean separation from the otherwise discless environment is achieved. The benchmark executables were small and there was enough RAM on both systems to avoid swapping. For this scenario, four NFS block I/O daemons (biods) were run on the client and four network file server daemons (nfsd) were run on the root server.

In this way a situation was established for comparing two different distributed networking implementations. The HP-UX discless implementation uses special-purpose networking protocols, buffer management functions, and CSPs, whereas NFS is designed for a heterogenous multivendor environment and uses a different protocol stack and general-purpose HP networking and buffer management functions.

### Throughput Results

Fig. 5* represents the throughput of the read and write benchmarks for the discless and NFS protocol paths. They are compared to stand-alone results of running the same benchmarks on just the root server (i.e., no networking involved). The throughput for the discless system with this environment is encouraging. The client was able to read at a rate of approximately 389K bytes/s or 91% of the stand-alone rate for this disc drive. The write statistics are also encouraging, achieving a rate of approximately 363K bytes/s or 87% of the stand-alone numbers. The root server's file system buffer cache for this experiment was only 2.4M bytes, so few if any cache hits occurred.

The more general-purpose NFS networking path was able to achieve rates of approximately 309K bytes/s on reads, or 72% of the stand-alone rate. The NFS write statistics are less encouraging, averaging only 61K bytes/s or 15% of stand-alone. This can be directly attributed to the lack of delayed asynchronous writes in the standard implementation of NFS. Comparisons between discless and NFS writes are somewhat meaningless, since they represent different design methodologies.

Given that the benchmarks were tightly controlled, repeatable, and run on identical hardware, it can be safely stated that the overall throughputs for this experiment seem to favor the discless implementation. Performance measurements are very application dependent. These benchmarks do not address large-cluster performance and the resultant clusterwide throughputs. The data should only be considered within the context of the established experiment. Different hardware, disc drives, and numbers of discless cnodes all play a role in evaluating clusterwide performance. It is beyond the scope of this paper to address these issues.

*These results are for the latest HP-UX Series 300 release, 6.2. The NFS throughput performance in release 6.2 is much better than release 6.0. There are no significant differences in the discless read or write throughput performances between the two releases.

### Kernel Measurement Method

To improve our understanding of the throughput data for the benchmarks, the server's kernel was instrumented and statistics were gathered and examined in great detail.

**Functional Area Breakdown of Time Spent in the Server's Kernel for an NFS Read**

| Kernel Functional Area | Total Elapsed Time | Total Clock Ticks | Percentage Non-Idle Time in Kernel | 95% Confidence Interval (L ≤ P ≤ U) |
|---|---|---|---|---|
| bcopy: | 1m04.40s | 3220 | 30.21% | 29.3 – 31.1% |
| General Kernel: | 0m38.06s | 1903 | 17.85% | 17.1 – 18.6% |
| Disc I/O — DMA: | 0m23.32s | 1166 | 10.94% | 10.3 – 11.5% |
| LAN Driver: | 0m30.20s | 1510 | 14.17% | 13.5 – 14.8% |
| Network Buffer Management: | 0m12.52s | 626 | 5.87% | 5.4 – 6.3% |
| File System: | 0m10.44s | 522 | 4.90% | 4.9 – 5.3% |
| NFS protocols: (UDP, XDR, RPC, NFS, and IP.) | 0.32.62s | 1631 | 15.29% | 14.6 – 16.0% |
| Discless Overhead: | 0m01.02s | 31 | 0.29% | .2 – .4% |
| Totals: | 3m33.20s | 10,660 | 100.00% | |
| *Kernel Idle: | 7m08.08s | 21,400 | | |

(a)

**Functional Area Breakdown of Time Spent in the Server's Kernel for a Discless Read**

| Kernel Functional Area | Total Elapsed Time | Total Clock Ticks | Percentage Non-Idle Time in Kernel | 95% Confidence Interval (L ≤ P ≤ U) |
|---|---|---|---|---|
| bcopy: | 1m13.70s | 3685 | 42.65% | 41.6 – 43.7% |
| General Kernel: | 0m33.44s | 1672 | 19.35% | 18.5 – 20.2% |
| Disc I/O — DMA: | 0m22.40s | 1170 | 13.54% | 12.8 – 14.3% |
| LAN Driver: | 0m21.02s | 1051 | 12.16% | 11.5 – 12.9% |
| File System: | 0m09.30s | 465 | 5.38% | 4.9 – 5.9% |
| Discless Protocol: | 0m08.56s | 428 | 4.95% | 4.5 – 5.4% |
| Buffer Mgmnt.: | 0m02.40s | 120 | 1.39% | 1.1 – 1.6% |
| GCSP: | 0m01.64s | 82 | 0.95% | 0.7 – 1.2% |
| DM Layer: | 0m00.76s | 38 | 0.44% | 0.3 – 0.6% |
| Totals: | 2m53.22s | 8641 | 100.00% | |
| *Kernel Idle: | 2m12.06s | 6603 | | |

(b)

*This is time spent in the kernel idle loop waiting for such things as disc I/O.

**Fig. 6.** *Kernel server measurements by functional area. (a) NFS read of 100M bytes using 8K-byte blocks. (b) Discless read of 100M bytes using 8K-byte blocks.*

At every clock tick (every 20 ms on an HP 9000 Series 300), the system sampled the processor's program counter (PC), the active process identifier (PID), the type of process (CSP versus regular), and other system parameters. The samples were written to an incore kernel buffer, extracted from the buffer by a user-level process, and then written to disc. The data was postprocessed by associating the sampled program counters with specific kernel procedures. The incremental cost imposed on the system by the monitoring actions was minimal, approximately one fifth of one percent of all the time spent in the kernel. The data was postprocessed on another machine after the entire benchmark was completed. Each benchmark was repeated ten times, and the results were completely consistent. The postprocessed data was segmented into separate kernel functional areas: byte copy (bcopy), disc I/O, LAN device driver, file system, networking protocol stack(s), and buffer management.

As a result of using a 20-ms sample rate, some level of confidence with the derived numbers was required. A 95% statistical confidence level was chosen and upper and lower statistical confidence intervals were calculated.[1] The confidence intervals were derived on the basis of the number of observed clock ticks in each kernel functional area as a function of the total number of samples taken. All numbers were rounded up to three decimal places. The 95% confidence intervals were derived as follows:

P = (functional_area_ticks) / (total_sample_ticks_N)
Lower Bound L = P − 1.96 * sqrt ((P * (1 − P))/N)
Upper Bound U = P + 1.96 * sqrt ((P * (1 − P))/N)

### Discless Server Kernel Functional Area Profiles

Fig. 6 shows the profiles for the functional areas measured in the server's kernel. The profiles show a read benchmark (executed from the client) using both the discless and NFS protocol paths to transfer 100M bytes of data in 8K-byte blocks.

The NFS write strategy does not make allowances for delayed asynchronous writes, while the HP discless implementation does. Since the HP discless and NFS client write strategies are so different, it seemed unnecessary to present data that could not be realistically compared.

An enlightening set of observations can be extracted from the server's kernel profiles compared with the client throughput results. Of all the time spent in the server's kernel routines, byte-copying data (bcopy) is by far the largest consumer of CPU resources. This is predominately the CPU cost of copying data to or from the LAN card's hardware buffers and the target networking buffers.

The next interesting set of numbers shows the amount of kernel time spent in performing discless protocol and buffer management functions: 4.95% and 1.39% respectively, with a combined elapsed time of 10.96 seconds. Comparing these numbers with the more general-purpose NFS path, we get 15.29% for the NFS protocol stack and 5.87% for buffer management with a combined elapsed time of 45.54 seconds. These comparisons must also be weighted by acknowledging that the entire 100M-byte read took 247 seconds for the discless system and 441 seconds for the NFS path.

The combined total of the discless specific components—protocol, buffer management, CSPs, and DM layer—accounts for 10.73% of the server's total kernel time, or an elapsed time of 13.36 seconds. This is only 41% of the time spent in just the NFS protocol stack to accomplish an equivalent transfer of data.

### Conclusions

High-level algorithms play a key role in the performance of distributed systems. Special-purpose networking protocols, server processes, and network buffer management routines must all play together in the design of such a system. Good performance requires not only a system view of the goals, but also an efficient implementation of the design.

Special-purpose designs like the HP-UX discless implementation have their advantages and disadvantages. The advantages are considerable in the context of a closely knit work group where a single-system view, high-speed intracluster communication, and transparent sharing of files and access of data are extremely important. As long as the special-purpose design allows a peaceful coexistence and complete interconnectivity with the outside world via standard and evolving networking services (ARPA/Berkeley, NFS, etc.), the user is provided with a powerful combination of capabilities. It is only in the more limited context of wide-area connectivity for discless cnodes that the special-purpose design shows disadvantages. Specifically, the inability to operate across a gateway limits the range of interconnectivity possible. It is this type of situation that places undesirable limits on the design of discless systems and tends to hinder their performance.

### Acknowledgments

Special acknowledgments are extended to the following individuals for their considerable contributions: Joel Tesler for his efforts in CSPs, DM, and protocol slow requests, Ching-Fa Hwang for slow request and initial project management, Bob Lenk for CSPs, Bruce Bigler for DM, Joe Cowan for project management, Doug Baskins for kernel instrumentation, and Ray Cheng for early protocol prototype.

### References

1. P.Z. Peebles, Jr., *Probability, Random Variables and Random Signal Principles*, McGraw-Hill, 1980.

# Crash Detection and Recovery in a Discless HP-UX System

by Annette Randel

HP-UX DISCLESS CLUSTERS depend on close, kernel-to-kernel communication across a local area network to maintain high performance and transparent file system access. This kernel-to-kernel communication relies on state information maintained on all nodes of the cluster. When a cnode is removed from the cluster because of an expected or an unexpected failure, this kernel state information must be cleaned up to reflect the new cluster configuration. Because this state information is at a very low level in the HP-UX implementation, failure to clean up state information after a crash can cause other cnodes in the cluster to hang indefinitely, waiting for the crashed cnode to complete a transaction. This is unacceptable, and therefore, prompt and reliable detection and cleanup of crashed cnodes are required at the kernel level.

For the purpose of this article, a crash or failure can be defined as the removal of a cnode from an HP-UX cluster. Two types of crashes or failures can occur on a cnode in a cluster: an expected failure or an unexpected failure. An unexpected failure may be caused by a hardware failure, a loss of power, or a software failure. When an unexpected failure occurs, the failing cnode may be unable to notify other members of the cluster of its demise. An expected failure occurs when a system is intentionally and properly shut down by the operator. During an expected failure, the cnode being shut down should notify all other cnodes that it is leaving the cluster. Reliable detection of both expected and unexpected failures provides the HP-UX system with resiliency in the face of an unexpected failure as well as dynamic reconfiguration around an expected failure.

There are four general requirements for crash detection and recovery in HP-UX clusters. First, it is required that the operating system maintain HP-UX semantics in the face of a failure. This requires that file system consistency and reliability be maintained, even though a cnode is removed from the cluster. Second, it is required that a consistent view of the cluster membership be maintained from all cnodes. A third requirement is that the detection of a crashed cnode and the recovery of that cnode's resources be transparent to users of other cnodes in the cluster. This means that no user action is required and the performance impact on the user is minimized. Finally, it is required that the rest of the cluster be resilient in the face of a client cnode failure. This means that the failure does not cause a chain reaction of failures in other cnodes and that no data loss occurs on nonfailing cnodes. The exception to this requirement is the root server cnode. Because client cnodes cannot recover from the failure of a root server cnode, a root server failure will cause all nodes in the cluster to fail.

## Crash Detection

Because of the state dependent nature of communication in HP-UX clusters, cnode failures must be detected quickly to prevent long system delays on functioning cnodes needing resources tied up by the failed cnode. In addition, failures must be recognized before allowing the failed cnode to rejoin the cluster. If the failed cnode were allowed to rejoin the cluster before crash recovery had taken place, valid state information could be improperly cleaned up, or invalid state information could be acted upon by the newly clustered node (e.g., it could respond to the retry of a request sent before the failure occurred). Neither is acceptable.

The crash detection mechanism must also ensure a consistent view of cluster membership for all cnodes in the cluster. All cnodes maintain their own cluster status table, and, while most cluster communications occur between the root server cnode and client cnodes, there are some kernel messages that are passed from one client cnode to all other cnodes. State information for these messages must be cleaned up on the client cnodes as well as on the root cnode after a failure occurs.

One final requirement of the crash detection mechanism is that it must never incorrectly declare a nonfailed cnode to have crashed, even in the face of a LAN failure. This requirement conflicts somewhat with the requirement to detect unexpected crashes quickly, and multiple detection mechanisms are employed to fulfill all the requirements.

**Detection Mechanisms.** Five different mechanisms are employed to detect both unexpected and expected cnode failures reliably in a minimal period of time:

- The failing cnode broadcasts a datagram indicating its expected failure.
- The server and each client cnode exchange status messages.
- When the server detects a cnode failure, it informs the cluster by broadcasting a reliable message.
- If a failed cnode attempts to rejoin the cluster before its failure has been detected, the clustering operation will be postponed until the crash recovery has been completed.
- LAN cable failures are detected, and the crash detection mechanism is disabled until the LAN is correctly configured. This prevents cnodes from being incorrectly declared failed because of LAN cable failures.

When all of these mechanisms are used together, they provide reliable detection of both expected and unexpected failures within a reasonable period of time. A more detailed description of each mechanism and their interactions is presented below.

**Broadcast Failure Datagrams.** When an expected failure or an orderly shutdown occurs, the failing cnode cluster-casts (broadcasts to the entire cluster) a failure datagram. This tells the other cnodes that it is about to shut down and that no further communication from this cnode should be expected. Since this message is a datagram, we cannot depend on cnodes receiving this message. The failure datagram is an attempt to get the word about the failed cnode out quickly, rather than a reliable information mechanism.

**Server/Client Status Messages.** To detect both expected and unexpected crashes reliably, the server and client exchange status messages. This message exchange occurs only when there has been no communication with a cnode for a given period of time. A state transition model (see Fig. 1) is used to track communication with other cnodes and to determine when a given cnode can no longer be contacted. This state transition model is based on an internal data structure, the cluster status table, which maintains a given cnode's view of the status of the entire cluster. A cnode entry in the cluster status table may be in one of the following states:

- ACTIVE: A message has recently been received from this cnode.
- ALIVE: No messages have been received from this cnode in several seconds.
- RETRY: A message has not been received from this cnode in several seconds, and we are currently requesting status information from this cnode.
- FAILED: This cnode has failed. If executing on the root server, all cnodes have not yet been informed of the failure.
- CLEANUP: This cnode has failed, and its resources are being recovered.
- INACTIVE: This cnode has never joined this cluster, or it has failed and recovery is complete.

Every time a message of any type is received from an ACTIVE, ALIVE, or RETRYing cnode, that cnode's status is updated to ACTIVE in the cluster status table. Every few seconds a kernel-level status checking process is executed which downgrades the status of each cnode according to its current state (see Fig. 2). On the root server cnode, this process downgrades the status of all cnodes. On discless nodes, however, this process only affects the state of the root server cnode. States are affected by this process as follows:

- ACTIVE: Downgraded to ALIVE.
- ALIVE: Downgraded to RETRY.
- RETRY, FAILED, CLEANUP, and INACTIVE: No change.

A status of ALIVE or RETRY may be upgraded at any time to a status of ACTIVE by the receipt of a message from that cnode.

When the status checking process downgrades a cnode's status from ALIVE to RETRY, it also sends a status request message to the cnode whose status is being downgraded. The status request messages are based on the datagram service. Datagrams are fast, but unreliable, so they must be retried manually. To do this, another process, the retry status request process is executed every second (See Fig. 3). This process determines, via a global variable if any cnodes are currently in the RETRY state. If there are none, this process simply exits. If, however, one or more cnodes are in the RETRY state, then the retry status request process searches the cluster status table for the RETRY cnode entries. When a cnode in the RETRY state is found, a retry counter is incremented for that cnode, and another message requesting a status update reply is sent. When a cnode whose retry counter has exceeded its maximum is found, that cnode is considered to have failed. However, before declaring the cnode as failed, the LAN failure detection mechanism, described below, will be invoked.

When a cnode receives a status request message, a status

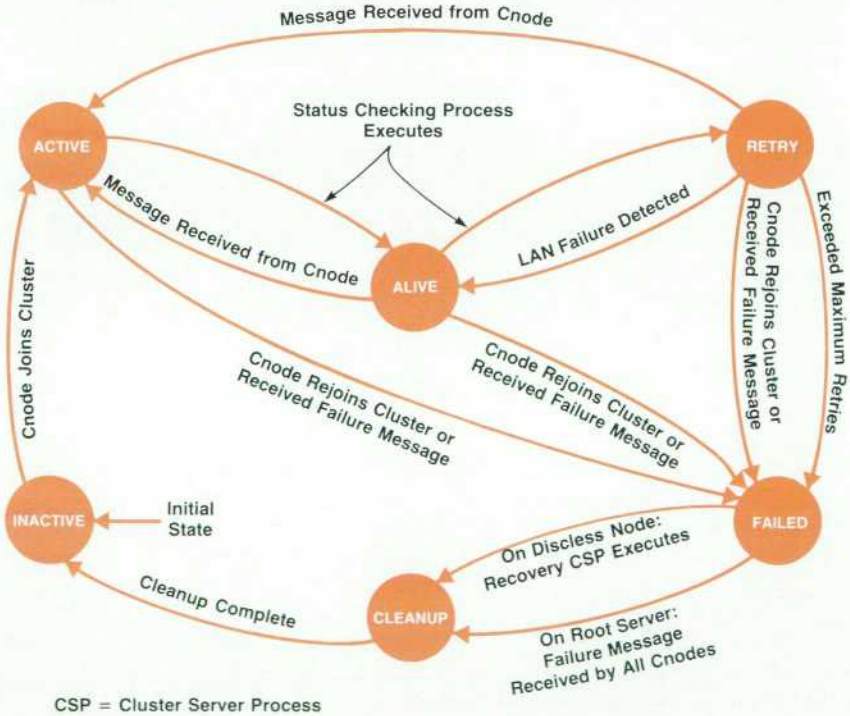



**Fig. 1.** *States of crash detection and recovery.*

reply message, also based on the datagram service, is immediately sent in reply. This message is also unreliable (i.e., not retried by the networking service), and the crash detection mechanism relies on retries from the requester to cover lost messages.

**Server Broadcast Failure Mechanism.** The broadcast failure datagram mechanism is unreliable, and because the server/client status message exchange does not inform the entire cluster of a failure, another mechanism is needed to ensure that the entire cluster is aware of a cnode failure. The mechanism used for HP-UX is a reliable failure message which is clustercast (broadcast to the entire cluster) by the root server.

The root server broadcasts the failure message from the crash recovery mechanism. In the recovery process on the root server cnode, if a cnode has a status of FAILED, then the root server broadcasts a message to all other cnodes informing them of the failure. The recovery mechanism uses a single process to clean up the resources of all failed cnodes. This process is serial, and because multiple cnode failures may occur simultaneously or close together, the recovery process cannot sleep waiting for the cnodes to reply. If it were to do so and if one of the cnodes that had not yet replied were to fail, then the root server would be deadlocked, because the recovery process would be waiting for a failed cnode that would never reply and whose resources would never be recovered. The crash recovery mechanism is discussed in more detail on page 30.

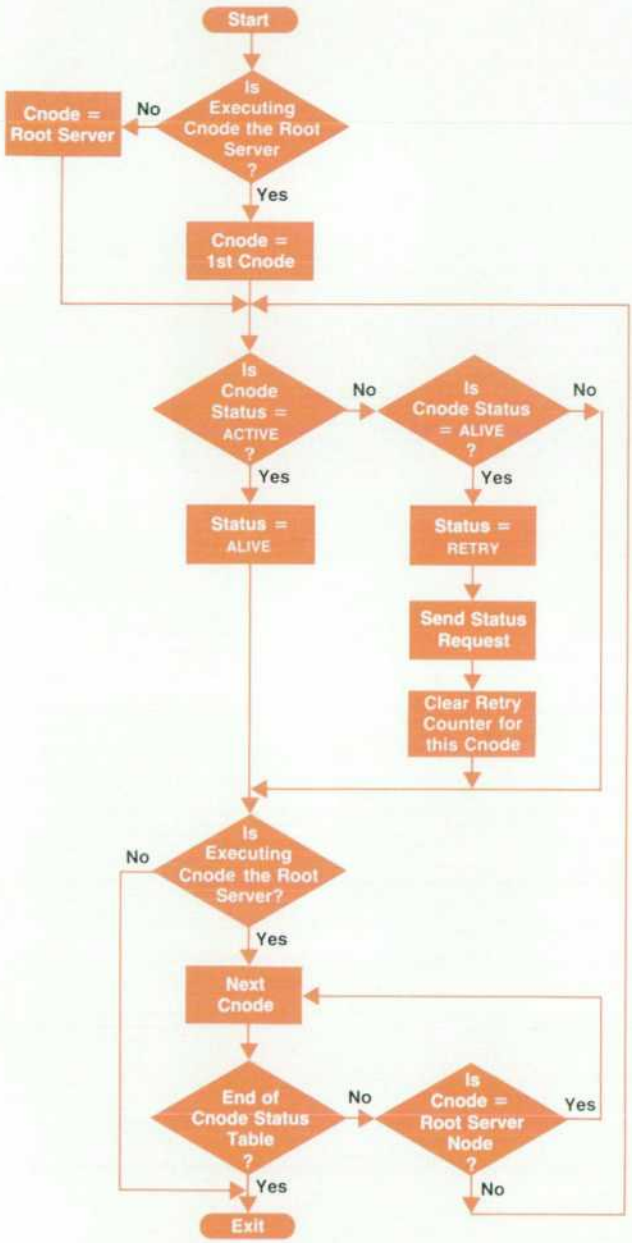



**Fig. 2.** *Crash detection: status checking process.*

**Failure Detection at Boot.** As mentioned, a failure must be detected and recovered before the failed cnode can be allowed to cluster (rejoin the discless cluster). Since all cluster requests are directed to the root server, the root server can block the cluster request until all recovery is complete. When a cluster request is received from a cnode whose status in the cluster status table is currently ACTIVE, ALIVE, or RETRY, the server process handling the cluster request sets the status of the requesting cnode to FAILED and invokes the recovery process.

Race conditions between the clustering process and the recovery process are prevented by not allowing more than one cnode to join the cluster at one time and by not allowing the joining process to continue until the recovery process has been completed.

**LAN Failure Detection.** Because detection of unexpected cnode failures is based on the exchange of status messages, an undetected LAN failure could be misinterpreted as a cnode failure if messages could no longer be received from a given cnode.

If a LAN failure were misinterpreted as a cnode failure, it would cause all cnodes on the failed LAN to panic (experience a system crash) because of loss of contact with the root server. The root server must detect the failure of a client cnode to ensure that resources held by the failed cnode are released, but it is less obvious why a client cnode must detect the loss of contact with the root server, especially when the only result is for the client cnode to panic. The client cnode must detect loss of contact with the root server because it is possible that, because of a hardware failure of some sort, the client cnode has become deaf (incapable of receiving messages) to the network. If a deaf cnode were allowed to remain in a cluster, it would continue to send requests to the root server, which would continue to mark the cnode as ACTIVE, even though the cnode could not participate in the cluster. This means that the deaf cnode could tie up cluster resources indefinitely, hanging the cluster. Therefore, client cnodes must panic on loss of contact with the root server to ensure that they are not tying up cluster resources.

A LAN failure is detected by running the LAN card hardware diagnostics from a LAN card driver level. These diagnostics are invoked by the retry status request process previously described, just before setting a cnode's status to FAILED. If the LAN card diagnostics indicate a LAN failure, a warning message will be displayed on the system console and the cnode's status will be upgraded to ALIVE. The process of downgrading all cnodes' status to RETRY,

running the LAN diagnostics, and upgrading all cnodes to ALIVE will be repeated until the LAN failure is repaired. All file system access from discless nodes will hang until the LAN is corrected, and it is possible that the root server cnode will hang waiting for a resource held by one of the isolated cnodes. Once the LAN failure is corrected, the system will continue normally.

The LAN card hardware diagnostics cannot detect a LAN failure that occurs on the other side of a LAN bridge. If a cluster is spread across such a bridge, a LAN failure (such as a break in the cable) on one side of the bridge will cause all discless cnodes on the other side of the bridge to lose contact with the root server and panic.

### Crash Recovery

Once a cnode failure has been detected, crash recovery is invoked. Recovery from the crash of a cnode requires that certain cluster resources being used by that cnode be released and cleaned up so they can be used by other nodes in the cluster. There are four basic resources that need to be cleaned up in the discless HP-UX system:

- File system data structures (inode)
- Swap space
- Process ID blocks
- Discless networking data structures.

In the current release, the root server maintains most of the cluster's resources, so most of the recovery function occurs on the root server. Discless networking resources are necessary on all cnodes, however, so crash recovery is invoked clusterwide with the root server performing most of the work.

A separate recovery function is called for each basic resource to be cleaned up after the failure of a cnode. The general tasks performed by each function are described below.

**File System Recovery.** When a failure occurs, the crashed cnode may have file system resources locked up, or those resources may be in an inconsistent state. Each currently referenced file (including named FIFO files, directories, and regular files) in memory is represented by a data structure called an inode. There are two situations where file system cleanup must be performed after a cnode fails, both of which must be performed on the inode:

- The file is locked by the failed cnode, either via a kernel inode lock or by a lockf(2) or fcntl(2) call. An inode lock is indicated by two fields in the inode, one that indicates that the inode is locked and another that indicates what cnode is responsible for the lock. If the file is locked by the failed cnode, then the recovery routine unlocks it. A lock created by lockf(2) or fcntl(2) is indicated by a lock list in the inode structure. If the lock list indicates that the file is locked by the failed cnode, then the recovery routine frees the lock.
- A cnode map field in the inode indicates that the file:
  - is being referenced by the failed cnode
  - is opened by the failed cnode
  - is opened for write by the failed cnode
  - is a FIFO file opened for read by the failed cnode
  - is a FIFO file being executed by the failed cnode.

A cnode map field in an inode is a table that maintains a count for each cnode in the cluster. Cnode maps only

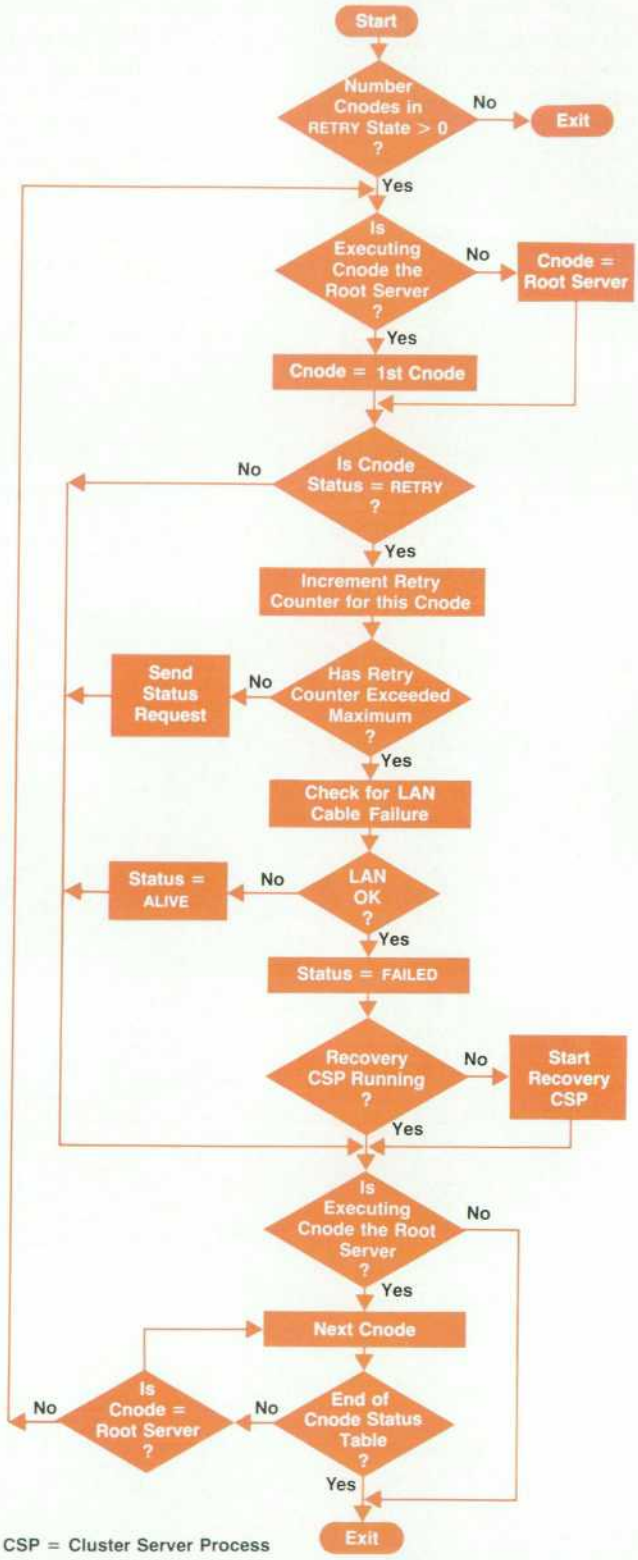



**Fig. 3.** *Crash detection: retry status request process.*

appear in inodes on the root server. Five different types of cnode maps may be present in an inode, one for each of the cases listed above. The recovery function checks each applicable cnode map (there is no point in checking the FIFO read count if the file is not a FIFO file) and, if the cnode map entry for the failed cnode is nonzero, then the appropriate action (e.g., closing the FIFO file or releasing the text segment) is taken, and the cnode map entry is cleared.

The file system recovery function looks through the memory-resident inode table and cleans up each inode that was being used or referenced by the failed cnode.

**Swap Space Recovery.** The swap space recovery function looks through the table of allocated swap space on the root server cnode and releases any swap space that was allocated to the failed cnode.

**Process ID Recovery.** In the discless HP-UX system, the root server cnode acts as a process ID (PID) allocator to guarantee unique PIDs throughout the cluster. The process ID recovery function goes through the PID allocation table on the root server cnode and marks any PIDs allocated to the failed site as available for use.

**Discless Networking Recovery.** Three basic discless networking resources must be cleaned up when a cnode crashes: cluster server processes (CSPs), networking state information on outstanding requests to other cnodes, and outstanding requests from other cnodes.

CSPs acting on behalf of the failed cnode are killed by a routine that scans the table of active CSPs for those with a cnode ID field matching the failed cnode. All such CSPs are sent a signal indicating that they should abort the current process. Since it may take some time for all the CSPs to abort, this routine is reinvoked until it does not find any CSPs acting on behalf of the failed cnode. The CSP cleanup routine also removes requests for CSP service by the failed cnode from the CSP service queues.

Networking resources for requests to the failed cnode and for remote requests being serviced on behalf of the failed cnode must be recovered. The list of outstanding networking requests is scanned, and all requests destined for the failed cnode are marked undeliverable. Retries on these requests are stopped, associated resources are freed, and a local reply is generated. Requests being serviced on behalf of the failed cnode are cleared, and replies to these requests are stopped and all associated resources are freed.

## The Recovery Mechanism

When a failure is detected, a cluster server process (CSP) is invoked in the kernel to execute the recovery functions. This process will be referred to as the recovery CSP. When more than one failure occurs simultaneously, or when a failure occurs while the recovery CSP is still cleaning up a previous failure, the same CSP is used to clean up the resources of all the failed cnodes. The cleanup of each failed cnode's resources is done serially. This means that we cannot allow any recovery function to sleep waiting for a resource that may be held by another cnode that may have failed, or we risk a possible deadlock situation.

To prevent this potential deadlock situation, each recovery function can return an error code that indicates that it was blocked from completing cleanup of a cnode because

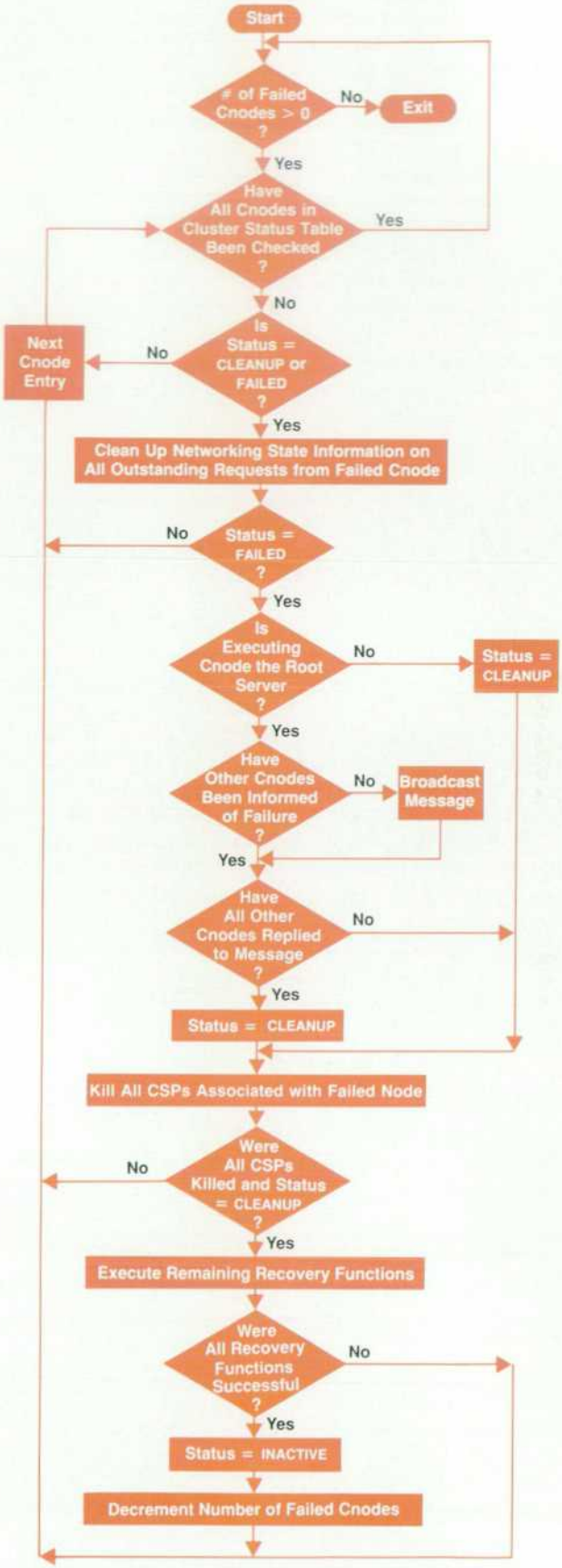



**Fig. 4.** *General crash recovery mechanism (recovery CSP).*

of a locked resource. When such an error code is returned, the failed cnode is not marked as cleaned up and the recovery functions will be rescheduled for that cnode after the recovery CSP has had an opportunity to execute the recovery functions on other crashed nodes in the cluster.

In addition, to prevent another deadlock situation, we must always guarantee that a CSP will be available to execute the recovery function. This is done by allowing the recovery function to be executed by a special CSP called the limited CSP. This CSP is limited because it only executes processes that are necessary to maintain membership in the cluster. None of the processes that are allowed to execute on the limited CSP should cause the limited CSP to sleep indefinitely waiting on a failed cnode. Therefore, we can always assume it will eventually become available for use in crash recovery. The recovery functions can also be executed on a general CSP if one is available.

The mechanism for crash recovery (see Fig. 4) is closely tied to the crash detection mechanism. Both use the state transition model (see Fig. 1) based on the cluster status table. To execute the recovery mechanism, a CSP is invoked by the crash detection function unless one is already running. This CSP checks a global variable, set up by the crash detection function, to see if there are any failed cnodes. If there are failed cnodes (there is always at least one on the first pass), then the process looks through the cluster status table searching for cnodes with a status of FAILED or CLEANUP (for the first pass for a given cnode, its status is always FAILED). When such a cnode is found, all outstanding kernel networking requests from the failed cnode are cleaned up. This must be done before the failed cnode's CSPs can be successfully aborted. The failed cnode's CSPs must be aborted before any other cleanup can be done to ensure that these CSPs do not make changes after the resource recovery has been executed.

If the failed cnode has a status of FAILED, then a discless cnode will update the failed cnode's status to CLEANUP. On the root server cnode, however, the root server will notify all discless cnodes of the failure and wait for all clustered cnodes to respond to this notification before it updates the failed cnode's status to CLEANUP.

The recovery CSP will then attempt to kill all the CSPs serving requests on behalf of the failed cnode. If all such CSPs have been killed, and if the failed cnode's status is now CLEANUP, then the rest of the cleanup functions will be invoked. Otherwise, the recovery CSP will return to the top of the loop and search the cluster status table for the next failed cnode. If the cleanup functions are successful, then the cnode's status will be updated to INACTIVE and the global count of failed nodes will be decremented. However, if some cleanup function was blocked from completing because of a potential deadlock situation, then the cnode's status will remain at CLEANUP and the recovery functions will be reexecuted for this cnode on the next pass through the cluster status table.

When a pass of the cluster status table is complete, the recovery CSP once again checks the global count of failed cnodes. If this count is nonzero, then the cluster status table is rescanned and the recovery algorithm above is repeated. When there are no more cnodes left in the FAILED or CLEANUP state, the recovery CSP terminates. When the recovery CSP terminates, the resources of all failed sites have been recovered and normal execution continues.

## Summary

Crash detection and recovery are an important part of making HP-UX clusters dynamic and resilient. Fast and reliable detection of failures allows users to add and remove cnodes dynamically without affecting users on other cnodes. LAN cable failure detection allows changes in the LAN configuration to be made without shutting down the cluster. Different mechanisms work together to make crash detection and recovery in HP-UX clusters both reliable and fast.


## Acknowledgments

Chyuan-Shiun Lin and Joel Tesler (then of HP Labs) did the initial design and implementation of crash detection and recovery on which the final implementation is based. Dave Gutierrez did the design and implementation of the LAN failure detection mechanism and was very helpful in the refinement of the networking resource recovery. Bob Lenk, Debbie Bartlett, Barb Flahive, and Fred Richart were all a great help in the refinement of various areas of resource recovery. Mike Berry and Fred Richart developed distributed, coordinated failure simulations, which were invaluable in the testing of this functionality.

# Boot Mechanism for Discless HP-UX

by Perry E. Scott, John S. Marvin, and Robert D. Quist

THE IMPLEMENTATION OF A DISCLESS WORKSTATION requires three distinct services: a remote file system, a remote swapping capability, and the ability to load and initialize the operating system from a remote source. All of these services are implemented for the HP-UX 6.0 system with the goal of maintaining a single-system view. For the boot mechanism this means that although the operating system and its loader are on a remote system (i.e., the root server), a user can power up any workstation in a cluster and get the same boot sequence that is experienced with a stand-alone system. A stand-alone system is a workstation that uses a local disc for booting and file system operations. This article describes how the standard HP-UX boot mechanism works, and the modifications made for the HP-UX 6.0 discless implementation.

## Overview

The major modules and interfaces involved in the HP-UX system boot mechanism are shown in Fig. 1. Fig. 1a shows the boot components for a conventional stand-alone HP-UX system and Fig. 1b shows the components for a discless configuration. The following sequence outlines what happens when a discless workstation is powered on and booted. A more detailed description of these steps and the components shown in Fig. 1 is given later.

- After power-up, the boot ROM searches for and assigns an input device (keyboard) and an output device (display) to use as a console.
- The boot ROM checks for and tests interface cards, RAM, and other internal peripherals. It then displays the information shown in the left side of Fig. 2. This is called self-test.
- The boot ROM loader polls all supported mass storage devices and LANs connected to the computer for an operating system, and the message SEARCHING FOR A SYSTEM (RETURN To Pause) appears on the display (see Fig. 2).
- If the user strikes the keyboard during self-test the boot ROM assumes the user wants to control the selection of the operating system to boot. This is called the attended mode. When this is done a list of available operating systems appears on the right side of the display (see Fig. 2). The user selects a system by entering one of the two character codes (e.g., 1H). If a key is not struck the boot ROM loader automatically selects the first bootable system it finds. This is called the unattended mode.

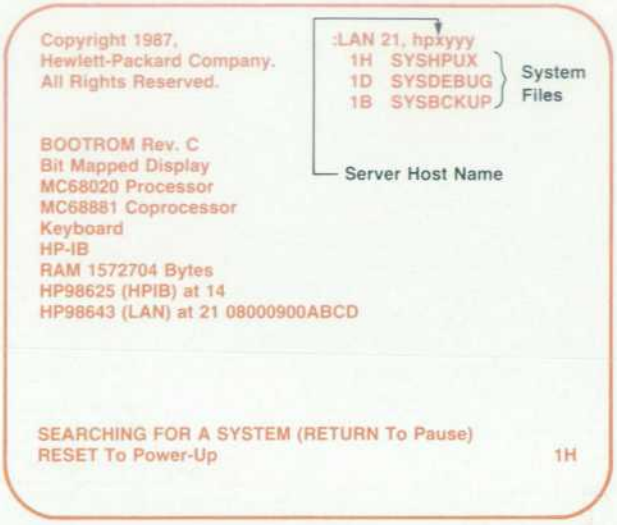



**Fig. 2.** *A typical screen the user sees during the boot process.*

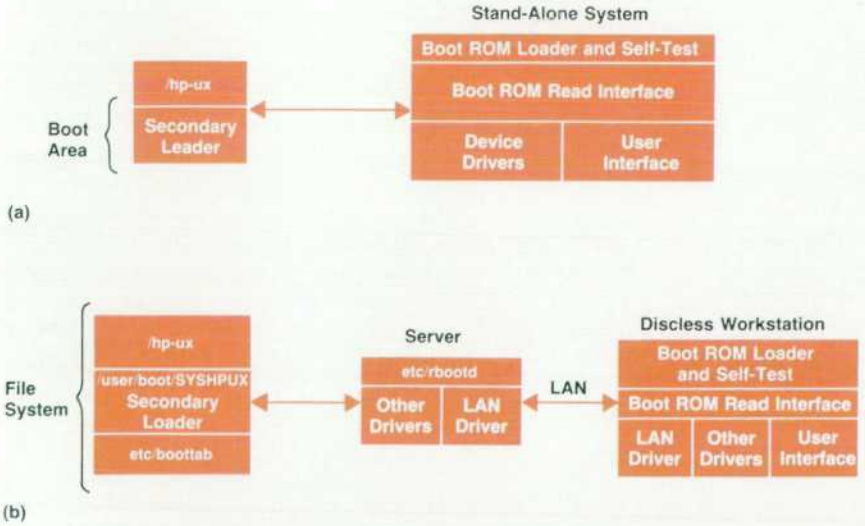



**Fig. 1.** *(a) The major components involved in the boot process for a stand-alone HP-UX system. (b) The major components involved in the boot process in a discless environment.*

- Once the operating system is chosen (assume 1H) the boot ROM retrieves the secondary loader from the server and loads it into RAM on the discless cnode. Control is then transferred to the secondary loader.
- The secondary loader retrieves the operating system (e.g., /hp-ux) from the server, loads it and transfers control to the operating system.
- The operating system initializes the discless kernel.

The first five steps in this sequence are called the boot ROM phase, and the last two steps are called the secondary loader phase and the HP-UX initialization phase, respectively.

Except for searching the LAN connection and loading the secondary loader from the server, these same actions also take place when a stand-alone HP-UX system is booted. The difference is that the stand-alone system accesses files directly from its local disc instead of going over the LAN. From the user's perspective, the boot process looks the same.

There may be more than one cluster of workstations connected to a LAN cable, and therefore more than one server may exist on the LAN. One of the main features of the discless boot mechanism is that when a booting cnode is polling the LAN connection for an operating system it is able to select the correct server. The mechanism for doing this is explained later.

### Discless Workstation Boot Modules

**Boot ROM Loader.** The HP 9000 Series 300 boot ROM loader is one of the boot ROM modules located in EPROM on the CPU board. After self-test the boot ROM loader initiates communication with the server to retrieve the bootable system files. During the boot sequence, when the boot ROM loader finds a LAN interface it broadcasts a server identify request packet. Typically a cnode belongs to one server; however, there is the possibility for a cnode to be configured with more than one server. Each server has a process called /etc/rbootd listening to the LAN. Based on the information in the server's configuration file (/etc/clusterconf), etc/rbootd decides whether to respond with the server's host name. The host name is then displayed on the cnode's system console. The process /etc/rbootd, which is discussed later, is a server daemon that handles communication with discless cnodes during boot.

For each server responding, the boot ROM loader sends a file list request packet containing a file number. The file number is incremented for each file list request sent to a particular server. As the file names are sent to the requesting cnode they are displayed on its system console (see Fig. 2). This is done until the file number exceeds the number of boot file names the server has available to send. At this point the server responds with a reply packet that indicates there are no more file names to send. When a bootable file is selected (e.g., 1H) the boot ROM sends a request to open the file. This file (e.g., SYSHPUX) is the secondary loader and resides on the server as /usr/boot/SYSHPUX.

In addition to opening the boot file, the boot ROM records several global variables in RAM that are used by the secondary loader and the HP-UX kernel. These values include:

- MSUS (mass storage unit specifier). Information about the boot device, such as the directory format, device type, and select code.
- SYSNAME. The name of the selected operating system (e.g., SYSHPUX).
- SYSFLAG2. The name of the processor type on the cnode (e.g., 68020).
- LOWRAM, HIGHRAM. The low and high limits of system memory.
- F_AREA. A driver scratch area where the LAN link level address of the server is stored. The link level address is retrieved from the IEEE 802.3 packet containing the server's host name.

After the boot file is opened, the boot ROM loader issues a read request packet to the server to read the secondary loader into the discless cnode's memory. When the secondary loader has been loaded, a boot complete packet is sent to close the boot file and terminate the session. The boot ROM then passes control to the secondary loader.

**Boot ROM User Interface.** The displays produced during boot and the handling of user input are the responsibilities of the boot ROM user interface modules. When a key is struck during self-test (attended mode) the interface module is responsible for assigning the two-character codes (e.g., 1H, 2B) to each bootable operating system that is found. All prompts and error messages go through the user interface routines.

**Boot ROM Read Interface.** The read interface provides file open, read, and close facilities to the boot ROM loader and the secondary loader, and it functions as an interface to the driver modules. The boot ROM loader uses the read interface to load the secondary loader, and the secondary loader uses it to load the HP-UX system.

The read interface operates in either an absolute mode or a file mode. In file mode, file relative addressing is used to access files on the server. The booting cnode relies on the server to resolve the logical address into physical disc blocks. In absolute mode, device relative addressing is used and the calling routine is responsible for performing the logical-to-physical disc block mappings.

For the discless implementation one of the design goals was to make the read interface to the LAN driver look like other devices so that existing secondary loaders would not have to change. The original HP-UX loader was built on the assumption that it was always booting from a local disc; therefore, it uses the absolute mode. The absolute mode proved impractical for the LAN driver. The HP-UX secondary loader was modified to recognize nondisc devices and use the file mode. We already had secondary loaders for our BASIC and Pascal workstations which use the file mode for boot over the Shared Resource Manager (SRM). The SRM has characteristics similar to the LAN.

### Root Server Boot Modules

**/etc/rbootd (remote boot daemon).** /etc/rbootd is a process that runs on the root server and handles all of the boot protocol requests between the server and the discless workstations. Rbootd uses two files to determine how it should respond to requests from the discless cnodes: a configuration file /etc/clusterconf and a boot table /etc/boottab. The configuration file contains the names and link level addresses of the cnodes associated with the server. /etc/boottab contains a list of boot files available to each cnode in the cluster. Rbootd

detects when changes are made to either of these files and reconfigures itself using the new information.

To allow context dependent boot files (files tailored to the capabilities of the workstation), rbootd emulates the pathname lookup code used by the HP-UX 6.0 kernel to handle context dependent files.The emulation is not perfect since rbootd cannot determine some of the hardware-specific context (e.g., whether the discless cnode has an MC68881 floating-point coprocessor installed). Therefore, hardware-specific context elements are not supported for boot files. Context dependent files (CDF) are discussed in detail in the article "A Discless HP-UX File System," on page 10.

Rbootd supports four levels of error and information logging, ranging from logging only fatal errors to recording the beginning and end of every boot session. The logging level is set with a command line option.

The communication protocol used by rbootd is based on a simple request/reply model. When a packet arrives, rbootd wakes up and processes the packet, usually by sending a reply, and then goes back to sleep. Requests are queued by the link level access driver in the kernel. Because queue space is limited, rbootd uses HP's real-time priority feature to ensure that boot (especially unattended boot) does not fail because of dropped packets.

Several boot protocols were investigated for our discless implementation. The Trivial File Transfer Protocol (TFTP) was considered, but could not be used. First, the boot ROM read interface is random-access and TFTP is sequential-only access. Second, TFTP is built on top of IP, which would require more code in the boot ROM. Finally, the boot ROM must obtain a list of file names, which is not provided by TFTP. We could have worked around many of these limitations; however, we decided to use a version of the Remote Maintenance Protocol (RMP) boot capability. This protocol was already in use within HP and the only capability missing was the ability to obtain a list of files from the server. Investigation showed that special interpretation of certain fields in the boot request packet would allow this feature to be implementated.

Rbootd services five types of requests: server identify, boot file list, boot request, read request, and boot complete. The boot request, read request, and boot complete packet types are standard RMP requests. The server identify and boot file list packet types are extensions to the RMP boot request packet.

- Server Identify Request. In the boot ROM phase the discless cnode uses the server identify request to get a server's hostname. At the same time the server's link level network address is obtained from the IEEE 802.3 packet header sent by the server's LAN driver.
- Boot File List Request. The boot file list request is sent by the boot ROM to obtain the names of the files listed in /etc/boottab. The request packet contains an index number that is used by rbootd to respond with the name of the file. If the number is greater than the number of files available, rbootd responds with a packet indicating that there are no more boot files.
- Boot Request. A boot request opens the requested boot file and allocates a session number. This session number is used by the discless cnode for the read request and boot complete request. Session numbers are used to support concurrent boot requests.
- Read Request. A read request is used to read a boot file. The request packet contains an offset and the number of bytes to be read from the file. This enables the discless cnode to access data randomly from the boot file. Rbootd responds with a packet containing the number of bytes actually read.
- Boot Complete Request. Boot complete causes rbootd to close the boot file and deallocate the session number.

**Secondary Loader.** In a stand-alone system the secondary

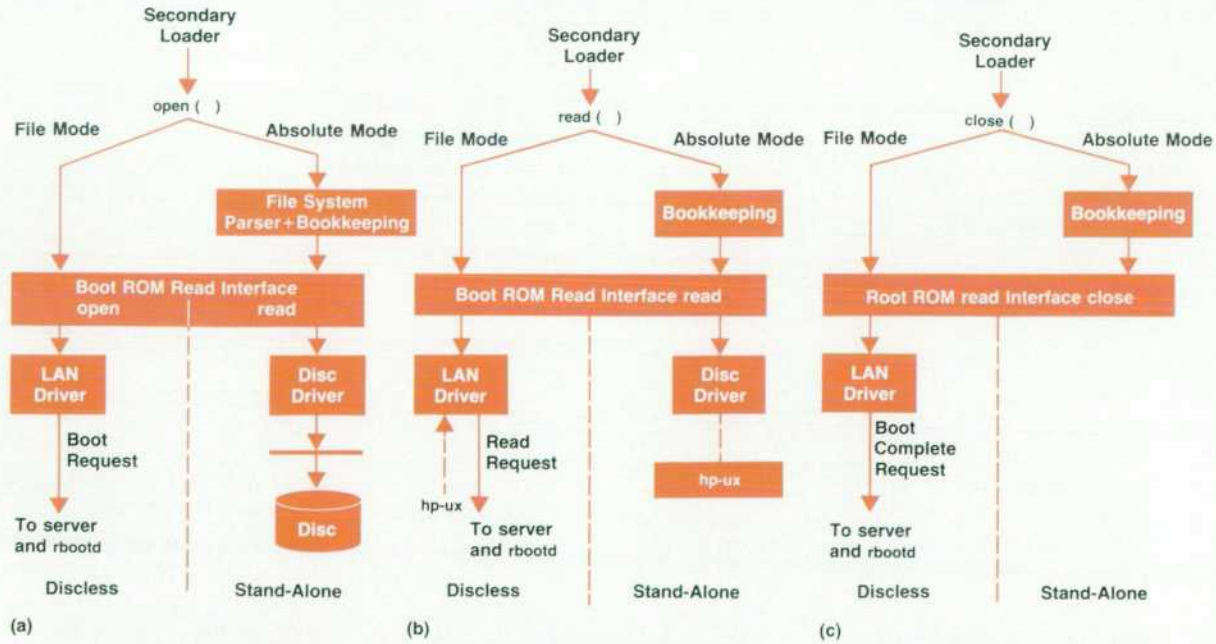



**Fig. 3.** *Secondary loader control flow on discless and stand-alone system during a) a file open, b) a file read, and c) a file close.*

loader resides in Logical Interchange Format (LIF) in the first 8K bytes of the boot disc. It is transferred to memory by the boot ROM interface routines at the end of the boot ROM phase.The purpose of the secondary loader is to load the /hp-ux a.out file (i.e., the HP-UX operating system) into low memory and execute it. Fig. 3 shows the secondary loader's flow of control and the processes involved for discless and stand-alone loading situations. The open( ), read( ), and close( ) routines emulate the behavior of the HP-UX system routines by the same name, and provide the secondary loader with an interface to the boot ROM read interface open, read, and close routines. The file system parser is a routine that understands the HP-UX file system structure and is responsible for resolving HP-UX pathnames during a boot file open in the absolute mode. Bookkeeping functions include the activities performed to keep track of data being transferred from disc (for instance, keeping a count of the number of blocks and current file offset and size, or processing partial or multiblock data transfers).

The secondary loader starts the loading process by examining the LOWRAM variable to determine the load point for the HP-UX kernel, and then uses the variable MSUS to determine the boot device. The name of the boot file is retrieved from the variable SYSNAME and the boot file name is translated to an HP-UX pathname and the open( ) routine is called. For instance, the boot file SYSHPUX is translated to /hp-ux.

The open( ) routine selects either the absolute or the file mode of the open operation depending on the type of boot device. For local boot the file system parser resolves the HP-UX pathname by using the boot ROM read interface read routine to perform pathname lookup. For a remote boot, as in the discless situation, the LAN driver is invoked through the boot ROM read interface open routine and a boot request is sent to the server where it is processed by rbootd.

The read( ) routine makes the same selection as the open( ) regarding absolute or file mode and uses the boot ROM read interface read routine to access the drivers. For absolute mode the loader uses the bookkeeping function to keep track of character counts, number of blocks read, and block addressing. For the discless situation a read request is sent to the server to be processed by rbootd. The read( ) operation results in transferring the selected operating system (/hp-ux) to the discless cnode's memory. The loading sequence for the operating system proceeds as follows: first the /hp-ux a.out header, which contains the sizes of the text, data, and uninitialized data areas, is read into a temporary area, and then the file /hp-ux is read into memory in two read calls, one for text and one for data.

When the operating system is loaded the close( ) routine is called. For the discless situation this results in a boot complete request being sent to rbootd. For the stand-alone situation the loader does some internal bookkeeping without calling the boot ROM. When the close operation is complete the secondary loader transfers control to the HP-UX kernel.

### Kernel Debugger Considerations

The above process changes slightly if SYSDEBUG is chosen instead of SYSHPUX. The kernel debugger is loaded just like the HP-UX kernel. When the debugger is started, it opens the a.out file /SYSDEBUG to find its relocation information, then moves itself into high RAM, adjusting all of its jump points. It then adjusts the HIGHRAM boot ROM variable, effectively protecting itself from being overwritten.

The debugger uses the secondary loader open( ), read( ), and close( ) routines, which are left in high RAM. After the user selects the kernel to boot, the debugger loads the HP-UX kernel like the secondary loader loads the HP-UX kernel.

### HP-UX Discless Kernel Initialization

The HP-UX discless kernel finds its server's LAN card address in the boot ROM F_AREA. This value is used to initialize several discless kernel pointers, which effectively turns on the discless message interface. The discless message interface provides the protocol for communication between a discless workstation and the server. The discless message interface is described in detail in the article "The Design of Network Functions for Discless Clusters" on page 20. Once the discless message layer is operational the discless cnode sends a cluster request message to the server. The cluster message contains the discless cnode's LAN address, which is used for security purposes, and its kernel release number, which is used to prevent server or client kernel mismatch.

The server validates the discless cnode's request by comparing the cnode's LAN address against the list kept in /etc/clusterconf. If it is not there the request is rejected. Likewise, the request is rejected if the kernel release numbers do not match. Otherwise, the server broadcasts a message to the rest of the cluster and the discless cnode is admitted. The server then sends a message to the cnode that contains the current system time, a description of the rest of the discless cnodes in the cluster, and the ID of the cnode's root and swap servers. At this point, the discless cnode can use the root server's file system, and control is passed to the /etc/init program. The discless file system is used to execute programs started by /etc/init, and kernel initialization is complete.

### Acknowledgments

The authors would like to thank the following individuals who contributed to the discless boot mechanism: Anny Randel for her work on the original /etc/rbootd design and prototype, David O. Gutierrez for his patient explanation of the HP-UX LAN driver, discless messages, and kernel initialization, and Joe Cowan for project management in bringing together the resources to complete the discless boot mechanism.

# Discless System Configuration Tasks

by Kimberly S. Wagner

GOING FROM A GROUP of stand-alone machines to a clustered environment is not a particularly difficult task, but because of the large number of steps required to configure the system, an automated tool called reconfig is provided with the HP-UX discless system to simplify the process. Reconfig enables the system administrator to set up the cluster server node and add or delete cluster nodes (cnodes) as necessary.

Reconfig was originally developed for the HP 9000 Series 200 and 300 Computers' HP-UX 5.1 operating system. The tool contains a collection of monotonous and terse system administration tasks within a user-friendly menu-driven environment. Basic tasks such as setting up user access to the system and reconfiguring kernels can be easily accomplished. With the advent of discless workstations in a clustered environment, changes were made to enhance the original reconfig tool.

## Cluster Setup

For creating a cluster configuration, the minimum system includes an HP 9000 Model 350 for the root server with at least 8M bytes of RAM, at least a 130-Mbyte disc drive, and the HP-UX 6.0 operating system (or later). The setup process begins by running /etc/reconfig, and when the main menu appears selecting the option Cluster Configuration. This selection will bring up the menu shown in Fig. 1, which shows the four values required to set up a cluster server: the server node name, the link level LAN address, the internet address, and the number of cluster server processes (CSPs).

**Cluster Node Name.** The server's cnode name is the system's hostname and it is used to identify the server cnode within the cluster. All discless cnodes refer to the root server by this name.

**LAN Card's Link Level Address.** Each LAN interface card has a unique link level address. This value is set by the factory and cannot be changed. Reconfig will display the address for each LAN attached to the system. If there is only one LAN card on the system its address is used by default; otherwise, one of the available cards must be selected using the NEXT, PREVIOUS, and SELECT softkeys.

**NS-ARPA Internet Address.** The internet address enables communication with other networks and uniquely indentifies the server within a network. The internet address is not required for discless interaction, but provides a minimal NS-ARPA networking capability within the cluster to handle remote process execution for system administrative tasks. If a value is automatically displayed, that value is the internet address associated with the cnode name that already exists in the system's /etc/hosts file.

**Cluster Server Process (CSP).** The CSP is a special process that is used to handle interprocess communication in a discless envrionment. Except for one limited CSP (LCSP) which exists on each discless cnode, all the other CSPs exist on the server. The default value is 4 and the amount entered will be the minimal number of CSPs running at all times. If more CSPs are needed they will be created automatically. For an in-depth discussion of CSPs see the article "The Design of Network Functions For Discless Clusters" on page 20.

When all the entries in the menu have been entered reconfig will tell the user what it is about to do to build the system and then ask if the user wants to continue. A yes will cause reconfig to begin configuration. Reconfig performs five steps in transforming the stand-alone system to a clustered environment. The steps are done in a particular order,

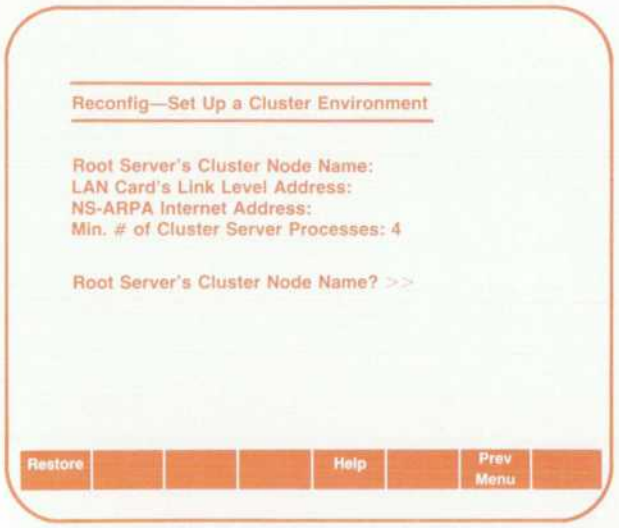



**Fig. 1.** Reconfig *menu for creating a cluster environment.*

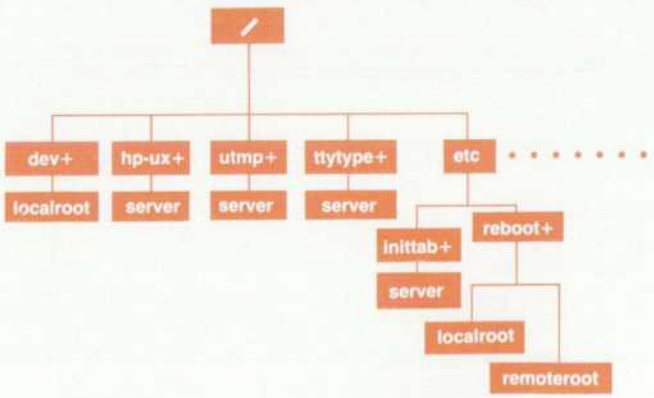



**Fig. 2.** *Partial system model for the server cnode. "Server" is the cnode name given to the root server.*

so if an error occurs during the process, the user can correct the problem and then reexecute reconfig from where it left off. Generally, each step in the process will complete without error unless certain required files and/or directories are missing. The activities that take place at each step are as follows:

- Context dependent files (designated by a + appended to the file name) are created for the cluster and root server based upon a predetermined system model (see Fig. 2). Context dependent files (CDF) are used to describe the various attributes (e.g., machine type, coprocessors) of a particular cnode. For more information on CDFs see the article "A Discless HP-UX File System" on page 10.
- A fully loaded root server kernel is built. The directory /hp-ux is turned into a CDF and the new version of the kernel resides in /hp-ux+/<cnode_name>. Cnode_name is the name entered earlier for the cluster node name.
- The NS-ARPA files for remote process execution are set up and an entry is made for the root server for the following files: /etc/hosts, /etc/hosts.equiv, and $HOME/.rhosts (root's home directory).
- The cluster configuration file /etc/clusterconf is created and the following information is entered in the file for the root server: the root server's cnode name, the server's link level LAN address, and the number of CSPs to start at boot.
- The rc file, which initiates the boot service for the discless cnodes, is modified to state which LAN device file to use if the default LAN device file is inappropriate.
- Reboot system.

### Adding and Deleting Cnodes

Once the root server of the cluster has been set up, discless cnodes can be added or deleted at will by running /etc/reconfig and selecting the Cluster Configuration option from the main menu. If the root server has already been set up (e.g., /etc/clusterconf exists), reconfig will present two menu choices for adding or deleting discless cnodes.

### Adding a Cnode

The input required for adding a cluster node is similar to that required for initial cluster setup: the cnode name, an internet address, and the link level address of the cnode's LAN card. Each discless cnode always runs exactly one limited cluster server process (LCSP) so there is no need to prompt for the number of cluster server processes. The process for adding a cnode is similar to that for setting up the clustered environment on the root server. The four steps are as follows:

- Context dependent files are created for the new discless cnode based on the system model for client cnodes (see Fig. 3).
- A minimally loaded discless cnode kernel is built. The directories /hp-ux+/<cnode_name> and /etc/conf/dfile+/<cnode_name> are created.
- NS-ARPA files for remote process execution are set up for the discless cnode. The files /etc/hosts, /etc/hosts.equiv, and $HOME/.rhosts (root's home directory) are modified to include the new cnode.
- The cluster configuration file (clusterconf) is modified to include an entry for the discless cnode. The entry includes the new cnode's name and its link level LAN address.

### Removing a Discless Cnode

Only the discless cnodes can be removed with reconfig. All that is required to remove a discless cnode with reconfig is the cnode name. The menu shown in Fig. 4 is used to select the cnode to be removed. There is an option to remove or not to remove all CDFs associated with the cnode. Unless there is a good reason for leaving the CDF elements around, the CDFs should be removed when the discless cnode is removed. The cnode removal process involves the following steps:

- Remove the ability to do remote process execution by deleting the entries for the cnode from the NS-ARPA files /etc/hosts.equiv, and $HOME/.rhosts. The cnode name and its associated internet address remain in the file /etc/hosts for later use.
- Remove the entry in the cluster configuration file (/etc/clusterconf) for the deleted cnode.

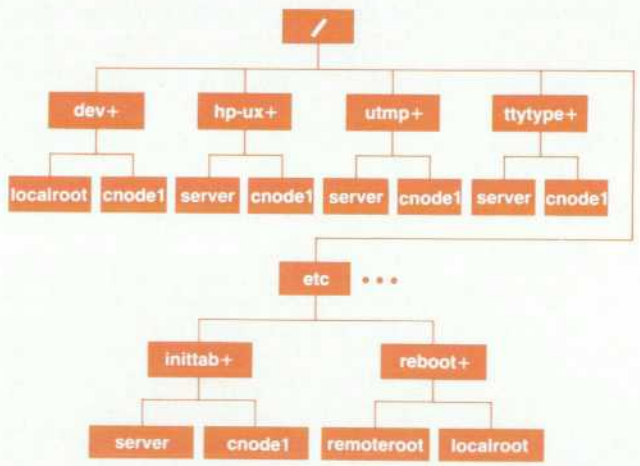



**Fig. 3.** *Partial system model for a client cnode. "Cnode1" is the cnode name given to the new discless cnode.*

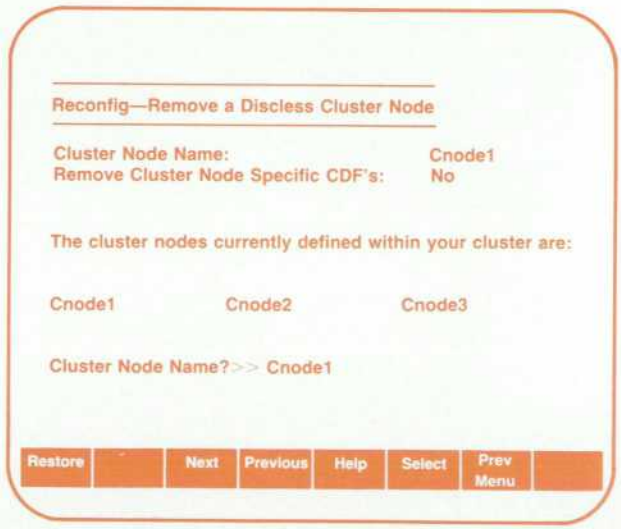



**Fig. 4.** Reconfig *menu for deleting a cnode.*

- If requested, remove all context dependent file elements of the form : <file>+/<cnode_name>.

### Conclusion

The Reconfig tool provides features that make the tasks of setting up and maintaining an HP-UX discless cluster much easier. In addition, the time required for reconfiguration is much lower with Reconfig than it would be to administer each system individually. This is one of the advantages of the HP-UX discless system.


### Acknowledgments

Special thanks go to Stuart Bobb and Dave Grindeland for their usability testing efforts, and to Paul Christofanelli and Paul Van Farowe for their invaluable NS-ARPA networking assistance.


# Small Computer System Interface

*The SCSI standard is the newest interface for the HP 9000 Series 300 family of HP-UX workstations. It offers improved performance, simplicity in design, a wide choice of controller chips, and wide acceptance in the UNIX® community*


**by Paul Q. Perlmutter**


DURING THE PAST FEW YEARS manufacturers of small computer systems and intelligent peripheral devices realized the need for an industry standard I/O interface for their systems. This interest resulted in the Small Computer System Interface (SCSI) standard. HP introduced an SCSI interface in April of 1988 for a family of high-performance disc drives. The SCSI standard is the newest interface for the HP 9000 Series 300 family of HP-UX workstations. It offers improved performance, simplicity in design, a wide choice of controller chips, and most important, wide acceptance in the UNIX community. Marketing data predicts that by mid-1989, approximately one half of all UNIX workstations will have an SCSI interface. This article provides an overview of the SCSI standard and the implementation of SCSI on the Series 300.

### What is SCSI?

The Small Computer System Interface—or SCSI—is an intelligent, general-purpose I/O bus. The entire spectrum of requirements for SCSI is specified in one document: ANSI committee standard X3.131. This standard defines the physical layer, the logical interface layer, and the device command set level for peripherals used with small computers. SCSI is very popular partly because all levels were designed and specified together, resulting in an I/O system that is integrated in a consistent and homogeneous style.

A critical design goal for SCSI was to provide the host computer with device independence within a certain class of peripheral devices. For direct-access drives (i.e., discs) this means the features that distinguish different discs are hidden from the software. The disc dependent characteristics such as the disc geometry, timing, protocol, and feature set are elements that make a disc less compatible. SCSI tries to hide these elements from the software without compromising product performance or quality. This significantly simplifies the development of the disc software driver, and enables the software to achieve a high degree of autoconfigurability. It also improves plug-and-play possibilities between different vendors' disc drives. For instance, many disc drive manufacturers have developed command sets that have many common features. SCSI extracts the common ingredients of these command sets and creates an industry standard format, command syntax, and command set. To simplify addressing, SCSI discards the older 3-vector addressing mode (sector, track, cylinder) and adopts the simpler single-vector addressing mode. In single-vector addressing, the disc is viewed as a logical single-dimensional array of blocks, and the software merely specifies the block offset from the beginning of the device.

This approach serves to bring divergent disc-like peripherals such as write-once-read-many optical discs (WORMs), CD ROMs, and flexible discs much closer together.

The Series 300 interface card (host adaptor in Fig. 1) uses the Fujitsu MB87030 controller chip to interface to the SCSI bus. This chip simplifies the software interface to the bus. It contains an 8-byte FIFO buffer, and provides DMA, asynchronous, and sychronous methods of data transfer.

Our SCSI disc driver is very flexible and highly autoconfigurable. Almost no assumptions are made about the disc. When a command first accesses the device, the driver checks to determine if the disc is alive (test_unit_ready command), then asks who it is (inquiry command), and finally, asks the disc drive for two basic geometric parameters: the logical block size in bytes, and the size of the drive in logical blocks. These two values are saved in the buffer header associated with the device, and are used for the duration of the transaction.

### The SCSI Bus

Only two devices are allowed to communicate on the SCSI bus at any time. Up to eight devices can be connected to the bus and a unique SCSI ID bit (0-7) is assigned to each device. One of the devices must be the host or initiator. When two devices communicate on the bus, one acts as the initiator and the other acts as the target. The initiator (usually a host system) originates an operation (e.g., read or write) and the target (e.g., disc controller) performs the operation. This operation is similar to the HP-IB talker/listener protocol. An SCSI device usually has a fixed role either as an initiator or a target. However, some devices can perform both roles. A typical SCSI configuration is shown in Fig.1.

An important assumption made by SCSI forces the target to drive the bus phases, and the target is allowed to disconnect from the bus when it anticipates a significant delay during data transfer. This fundamental assumption allows multiple drives to be active simultaneously, enhancing total bus bandwidth. The idea is this: since we can have only one initiator and one target active at any given time, we do not want a device to tie up the bus unless data is actively being transferred. Thus, devices are allowed to disconnect while internal-only activities (such as seeks or command parsing) are occuring. Typically, after a command has been transferred, a device will disconnect while it parses and decodes the command, seeks to the appropriate cylinder, and prepares itself for data transfer. In addition, if in the middle of a data transfer the drive anticipates dead time (such as a seek to a spared track), the device will get off the bus to allow other peripherals to access the host. As soon as the target is ready to resume data transfer, it can actively arbitrate for the bus (when the bus is free) to reattach to the host. The disconnect/reconnect option in SCSI can boost overall system performance.

### SCSI Bus Signals

The SCSI bus consists of eighteen signal lines. Nine are used for control and nine for data. The control signals are used to establish the logical bus phases (discussed in the next section) for the SCSI bus protocol, and control the transfer of data. These bus signals are shown in Table I.

**Table I**
**SCSI Bus Signals**

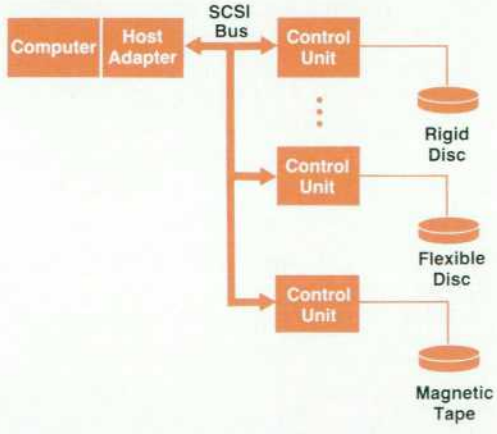
| Signal | Description | Driven by Host | Driven by Target |
|---|---|---|---|
| REQ | (Request) Data handshake line: requests data byte on bus. | | X |
| ACK | (Acknowledge) Data handshake line: acknowledge data byte on bus. | X | |
| BSY | (Busy) Indicates the bus is busy. | X | X |
| SEL | (Select) Used during selection and reselection to establish communication link. | X | X |
| I/O | (Input/Output) Indicates direction of data flow on the data bus. If I/O is true the flow is from target to initiator. | | X |
| MSG | (Message) Indicates the data on bus is a message (only valid if C/D asserted). | | X |
| C/D | (Control/Data) Determines whether control or data information is on bus. | | X |
| ATN | (Attention) Requests message out phase (initiator has message for target). | X | |
| RST | (Reset) Hard reset of all devices. | X | X |

In addition there are 8 data lines and one parity line that are driven by both devices.

### SCSI Bus Phases

The SCSI architecture defines eight distinct bus phases that define the logical characteristics of the SCSI bus:

- BUS FREE phase. Indicates when no SCSI device is actively using the bus.
- ARBITRATION phase. Allows one SCSI device to gain control of the bus.
- SELECTION phase. Allows an initiator to select a target.
- RESELECTION phase. Allows the target to reconnect to the initiator.
- COMMAND phase. Allows the target to request command information from the initiator.



- DATA phase. Provides data transfer between the initiator and the target.
- STATUS phase. Provides status information from the target to the initiator.
- MESSAGE phase. Allows the transfer of messages between the initiator and the target.

The first four phases allow devices to contend for access to the bus and establish the physical data path between the initiator and the target. The last four phases are called the information transfer phases because they are used to transfer data and control information over the data lines. The SCSI bus can never be in more than one phase at any given time. However, all devices can arbitrate for access to the bus. The following is a simple example of phase sequencing during a disc transaction:

| Phase | Comments |
|---|---|
| BUS FREE | No device on bus. |
| ARBITRATION | Initiator (host) arbitrates for bus. |
| SELECTION | Host establishes contact with target. |
| MESSAGE OUT | Host identifies itself to the target. |
| COMMAND | Host issues command (e.g., read or write). |
| MESSAGE IN | Target indicates it will disconnect and then drives the bus to BUS FREE. |
| BUS FREE | Target is detached from the bus while a seek is in progress. |
| ARBITRATION and RESELECTION | Target is read and reestablishes a link to the host. |
| MESSAGE IN | Target identifies itself to host. |
| DATA IN or OUT | Data is transferred. |
| STATUS | Target reports on transfer status. |
| MESSAGE IN | Command complete (done). |
| BUS FREE | |

### Bus Access Phases

**BUS FREE Phase.** This phase indicates that no device is using the bus and that it is available for use. BUS FREE is detected by the BSY (busy) and SEL (select) signals being false.

**ARBITRATION Phase.** Arbitration allows a device to gain control of the bus to initiate a transaction such as a data transfer, or to send a message or command. To gain control of the bus, a device (the initiator) first checks to see if the bus is free. If the bus is free the device asserts the BSY signal and sets its own device ID on the data lines. If more than one device is contending for the bus, the device with the highest priority gains access to the bus. The device that loses arbitration starts all over again and the device that wins asserts the SEL signal to end arbitration.

**SELECTION Phase.** After a device has gained control of the bus the SELECTION phase is entered by selecting the target device for the transaction. Target device selection is accomplished when the initiator sets the data bus lines to the OR of its SCSI ID bit and the target's SCSI ID bit, and asserts the ATN signal. The target will respond by asserting a MESSAGE OUT phase requesting an Identify message from the initiator. The Identify message establishes the physical data path between the initiator and the target. The initiator replies with a message indicating to the target whether it can handle target disconnection, and it determines if the target can handle synchronous data transfer. If an SCSI implementation does not support messages the target will go directly to the COMMAND phase.

**RESELECTION Phase.** When the target decides to disconnect from the bus temporarily, the RESELECTION phase is used to reestablish connection with the initiator to continue a transaction. The target device disconnects to free the bus for other devices to use when it anticipates a significant delay during the next data transfer. For instance, during a disc I/O operation the disc can disconnect from the bus while it switches heads, does a seek, or empties its controller's buffers. Before disconnecting from the bus, the target sends the messages Save Data Pointers and Disconnect to the initiator. The Save Data Pointers message tells the initiator to save the pointers to the current locations in its memory where the data is being transfered. The pointers are restored when the target reconnects to the initiator. When the target is ready to resume data transfer it must wait for BUS FREE, arbitrate for the bus, and then reselect the initiator. In implementations where there is no ARBITRATION phase the RESELECTION phase cannot be used. This also implies that the target device is not allowed to disconnect from the bus.

**Series 300 and Bus Access.** The Fujitsu chip controller used on the Series 300 interface card provides a very flexible SELECT command. For the host (initiator) to arbitrate and select a target, the host first writes the target ID bit to the TEMP register on the chip, and then issues the SELECT command to the chip. The chip handles the ARBITRATION and SELECTION phases, and will interrupt with one of three possible conditions:

- Command complete interrupt (selection completed). This indicates that the arbitration was successful and the target device responded to the SELECTION phase.
- Command complete interrupt (arbitration for the bus failed).
- Time-out interrupt (the target device did not respond, possibly because the device was powered off, or the device at the bus ID is not present).

### Information Transfer Phases

The information transfer phases are used to transfer data or control information over the data lines. These four logical phases are distinguished by three control lines: MSG, C/D, and I/O (see Table II).

**Table II**
**SCSI Information Transfer Phase Coding**

| Control Line | | | Phase | Direction of Transfer |
|---|---|---|---|---|
| MSG | C/D | I/O | | |
| 0 | 0 | 0 | Data Out | Initiator to target |
| 0 | 0 | 1 | Data In | Target to initiator |
| 0 | 1 | 0 | Command | Initiator to target |
| 0 | 1 | 1 | Status | Target to initiator |
| 1 | 1 | 0 | Message Out | Initiator to target |
| 1 | 1 | 1 | Message In | Target to initiator |

An important characteristic of SCSI is that the target drives the three control lines, and therefore controls all

changes from one information transfer phase to another. The initiator can request a MESSAGE OUT phase by asserting ATN. The initiator might use this feature to gain the attention of the target under special circumstances. For instance, suppose the host has decided that some type of catastrophic failure has occurred during the DATA phase (e.g., a parity error). The host (initiator) will assert ATN, and when the target recognizes the ATN condition, it switches the bus phase to MESSAGE OUT and allows the host to send a message. When this situation occurs for the Series 300 an Abort message is sent that tells the target to clear the current command and all status and data buffers, and allow the bus to go to BUS FREE immediately. The host may then attempt to retransmit the entire transaction or issue an error to the process that made the transaction request.

The REQ/ACK signal lines provide the handshake protocol used to control the asychronous and synchronous transfer of information during the information transfer phases. Each REQ/ACK handshake allows the transfer of one byte.

**Asynchronous Data Transfer.** The target uses the I/O signal to control the direction of transfer. If I/O is true the direction is from target to initiator (e.g., read), and if I/O is false the direction is from initiator to target (e.g., write). Fig. 2 illustrates the REQ/ACK handshake protocol which is repeated for each byte transferred. The SCSI asynchronous data rate is 1.5 Mbytes/s.

**Synchronous Data Transfer.** Asynchronous data transfer is the primary mode available in SCSI. However, synchronous data transfer is possible during DATA IN and DATA OUT if before transfer the initiator and target agree to this mode. When the synchronous mode is established the devices also agree on a minimum period between REQ and ACK signals and the maximum REQ/ACK offset. The REQ/ACK offset is used to determine the number of REQs the target will send in advance of the number of ACKs received from the initiator. During synchronous transfer the target does not wait for the ACK signal from the initiator before sending the next REQ signal to send or receive the next byte. The target will continue in this loop until the specified REQ/ACK offset. It will then compare the number of REQs with the number of ACKs to verify that all data has been transfered. The SCSI synchronous data rate is 4 Mbytes/s.

**COMMAND Phase.** When the target is ready to accept a command from the initiator it will drive the SCSI bus to COMMAND phase. The initiator will then send a command such as a read or write to the target.

**DATA Phase.** When data flows from the target to the host, we refer to this as the DATA IN phase, while DATA OUT indicates that data is going from the host to the target. This is the only information transfer phase that allows the synchronous data transfer option described above. For all other phases data must be transferred asynchronously.

**MESSAGE Phase.** When the MSG line on SCSI is asserted the data on the bus is interpreted as message bytes. Like the DATA phases, MESSAGE IN indicates that the target is sending a message to the host, while MESSAGE OUT indicates the message is going from the host to the target. Message bytes are used to help establish and coordinate the environment for data transfer. For instance, the Identify byte sent by the host during MESSAGE OUT identifies the host to the target and also indicates to the target whether it can handle disconnects and reconnects during data transfers. In a similar way, the target sends the host a Disconnect message to alert the host that it will immediately disconnect from the bus and drive the bus to BUS FREE. Messages are usually single bytes, but under certain situations are multiple bytes. For instance, extended messages are sent by the initiator to the target to determine whether synchronous transfer is feasible, and if so, to establish the synchronous data rate. During a transaction the MESSAGE OUT phase is initiated by the target in response to the ATN signal.

**STATUS phase.** The STATUS phase enables the target to inform the intitiator of the status of a transaction. After the target has completed a data transfer it sends one status byte back to the initiator. If the target sends a zero, the transaction completed normally. A nonzero status indicates that the target has additional status to send.

In the Series 300 implementation, if the status byte returned by the target is nonzero, we always request extended status. The Request_Sense command provides complete diagnostic results of the previous transaction.

In addition to bad status, other types of problems may occur. The Fujitsu chip may report parity errors or hardware errors that occurred during a transfer. Another error occurs when a time-out occurs. Whenever any hardware activity is initiated, a timer is started, and if the timer times out we assume a hardware failure has occurred.

Our error recovery philosophy is to give most transactions a second chance and no more. For instance, if at any point during a transaction a parity error occurs or a timer times out we always retry the transaction. We do not try again after a failure on the retry. The only exception to this second-chance rule is when the target reports through extended status that it could not recover from a drive error. In this situation we assume that the device's controller is smart enough to retry the transaction itself.

**Series 300 and Information Transfer.** The controller chip provides three methods of data transfer:

- Manual transfer. The host processor controls handshake lines.
- Hardware transfer with fast handshake. The chip controls the data transfer and the processor feeds the bytes to the controller's buffers.
- Hardware transfer with DMA.

Manual transfer is currently used for transferring messages and status bytes over the SCSI bus. The fast handshake option of the Fujitsu chip is used for transferring commands, while the hardware transfer with DMA is used for transferring data buffers. When a DMA channel is unavailable, the hardware fast handshake option is used to transfer data buffers.

### A Disc Transaction

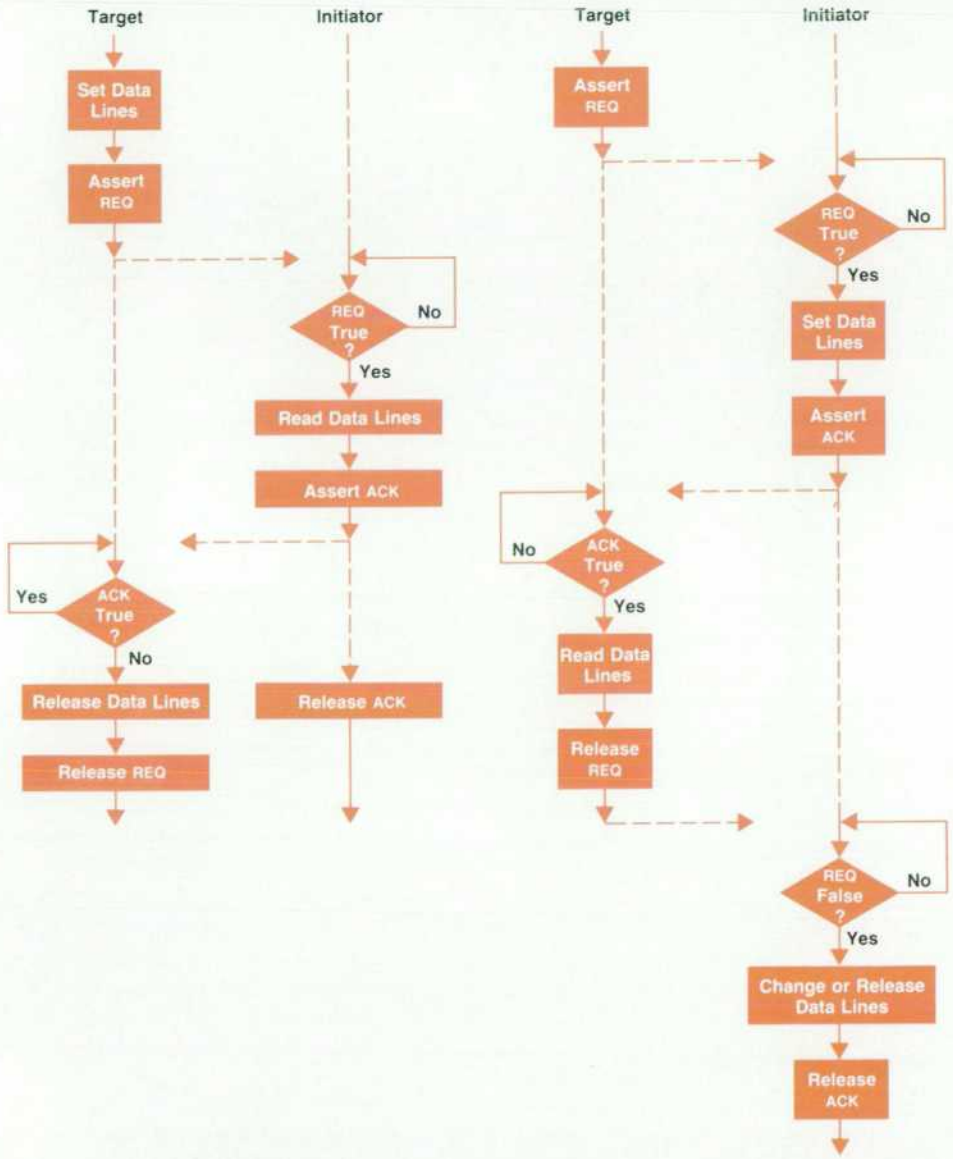
The following discussion summarizes the interactions that occur in the operating system (HP-UX), in the disc driver, and on the SCSI bus when a typical disc transaction is performed.

A disc transaction starts with a disc I/O request from the file system or other higher-level portion of the operating system. The request is passed to the driver via a buf header. This structure includes such information as the system device identifier, the data buffer address in memory, the

block offset on the disc, and the byte count of the buffer. The system device identifier encodes the major and minor numbers that are used by the UNIX system to specify special device files. The major number selects the appropriate device driver (in this case SCSI) and the minor number specifies the select code, bus ID, and unit number. The block offset on the disc is the logical offset viewing the disc as a linear array of blocks. The byte count given in the buf must be converted to a block count appropriate for that device.

When the system begins to service the I/O request and the driver gets permission to use the interface, the driver goes through the ARBITRATION and SELECTION phases to gain access to the disc. The disc (now the target) responds with a MESSAGE OUT and gets the Identify message from the host (initiator). Through the Identify message the driver tells the disc whether it can handle disconnect and reconnect during subsequent DATA phases. The driver asserts the ATN line to maintain the MESSAGE OUT phase and determines whether the disc is capable of synchronous data transfer.

Once the environment for data transfer is established, the driver issues to the disc the 6-byte or 10-byte command (COMMAND phase) that designates whether the operation is a read or write. When the disc receives the command it sends Disconnect and Save Data Pointers messages to the host, and then disconnects from the bus. The driver frees the interface for other processes to use while the disc is busy processing the command. The disc controller decodes the command to determine if it is a read or write operation, and then causes the physical mechanism to perform a seek operation. When the disc is ready to start the data transfer it reselects (RESELECTION) the host and data transfer begins. The disconnect (MESSAGE OUT), reconnect (RESELECTION) and data transfer (DATA IN, DATA OUT) phases may happen several times during the transaction. When all the data has




(a) Read (target to initiator, I/O signal = true)

(b) Write (initiator to target, I/O signal = false)

**Fig. 2.** REQ/ACK *handshake protocol.*

## SCSI and HP-IB

People are often comparing the HP-IB interface bus that is the standard bus for HP 9000 Series 300 Computers with our newest interface: SCSI. Both buses have many features in common, as well as some significant differences. The following is a short summary comparing the two interfaces.

| | HP-IB | SCSI |
|---|---|---|
| Design Goal | Instrument Bus (later adapted for high-speed interface) | Disc Interface |
| Transfer Protocol | Asynchronous only | Asynchronous and Synchronous |
| Bandwidth | ~1.2 Mbytes/s burst rates | Async. ~1.5 Mbytes/s Sync. ~4.0 Mbytes/s burst rates |
| Features | Devices may be powered off | Devices must always be powered on |
| | 8 devices per bus | 7 devices per bus |
| Parity | No | Yes |
| Bus Topology | Any style (e.g., star) except closed loop | Must be linear |
| Physical | One connector per device | Two connectors per device: one for input and one for output |
| | Connectors are sexless | Connectors have male and female ends |
| | Devices can be daisy-chained; connectors are stacked | Devices must be daisy-chained serially |
| | Termination not required (open-collector drivers eliminate need for terminators) | Bus must be terminated |
| Cable Length | Maximum cable length 8 meters total and at most 1 meter per device. | Maximum cable length 6 meters—single-ended only. The differential option offers up to 25 meters but is unavailable on the Series 300. |
| Handshake | 3 handshake lines: single talker and multiple listeners. High-speed peripherals only use single talker and single listener. | 2 handshake lines: single talker and single listener. |
| Addressing | Only hosts can select devices. Devices can respond to parallel polls. Thus, no bus arbitration is required. Primary HP-IB addressing establishes point-to-point communication. | Hosts can select targets, and targets can reselect hosts. Bus arbitration is required. |

In addition to the interface comparison, we can also compare the two disc protocols. In fact they are quite similar. Command Set 80 (CS-80), which is the HP-IB disc protocol, uses a command packet of variable size that allows a combination of transparent commands, addressing commands, and CS-80 commands. SCSI uses a fixed-format packet of either 6, 10, or 12 bytes. Message bytes are used in SCSI to complement the command by establishing the environment.

Example comparing read commands:

| | CS-80 (HB-IB) | SCSI |
|---|---|---|
| Device Addressing | Unlisten Bus<br>Talk (controller)<br>Listen (device) | Arbitrate<br>Select device<br>Identify **Message In** byte |
| byte 0 | Set unit 0 | Read Command |
| byte 1 | Set volume 0 | Address byte (set unit) |
| byte 2 | Set address | Logical block address |
| byte 3 | Data address | Logical block address |
| byte 4 | Data address | Logical block address |
| byte 5 | Data address | Logical block address |
| byte 6 | Data address | Reserved |
| byte 7 | Data address | Transfer length (upper byte) |
| byte 8 | Data address | Transfer length (lower byte) |
| byte 9 | **No-Op** | Control byte |
| byte 10 | Set length | |
| byte 11 | Data length | |
| byte 12 | Data length | |
| byte 13 | Data length | |
| byte 14 | Data length | |
| byte 15 | Read command | |

The HP-IB device addressing is almost equivalent to the **ARBITRATE** and **SELECTION** phases of SCSI. The **Message In** byte in SCSI selects the unit and also indicates to the target whether the host can accept disconnects. Although CS-80 commands appear longer, the difference in system performance is negligible. The **No-Op** in CS-80 is required because of the HP-UX C compiler. CS-80 uses a transfer length defined in bytes and SCSI uses the number of blocks.

been transferred, the status (**STATUS** phase) is sent to the host.

A requested data transfer may be broken into a series of shorter requests based on the device's requirements. The target device drives the bus phase and the host must be prepared for a phase change anywhere during a data transaction. Typically phase changes occur on logical block boundaries, but this is not guaranteed, and no assumptions can be made by the host when the phase change may occur. A typical transaction is shown in Fig. 3.

### Conclusion

Our objective in implementing SCSI on the Series 300 was to provide an industry standard interface that added flexibility, expandability, and improved performance to our product family. Our customers wanted to use peripher-

als unavailable from HP, and in some cases, SCSI enables them to do this. As an example, CD ROM support came without any change in the driver. Other customers hoping for plug-and-play compatibility wanted to buy inexpensive peripherals to lower their system cost. Here caution is necessary. SCSI does not automatically imply plug-and-play compatibility. SCSI merely sets up some basic frameworks for hardware and software designers. Options and vendor-specific commands are plentiful. Within commands there are frequently vendor-specific fields or options. In hardware, SCSI allows two different types of transfer, single-ended for short distances and differential for longer distances, which are incompatible. SCSI allows a variety of connectors and cabling. With this type of variability, any two devices may not be mutually compatible.

Perhaps the greatest advantage afforded by SCSI is its simplicity in design. This goal is admirably achieved. It simplifies the design and development of software drivers, and most important, it expedites testing and system integration.

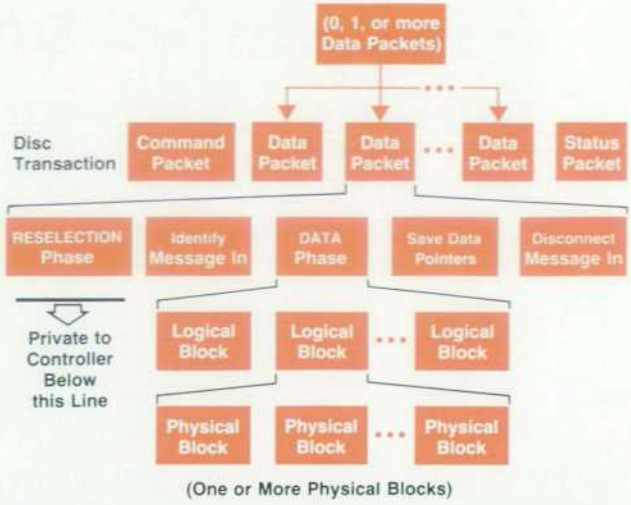



**Fig. 3.** *Data packets for a typical disc transaction on the SCSI bus. The Command packet contains the information for setting up the environment for data transfer. Information such as the* Identify *and the* Synchronous *messages are contained in this packet.*


## Acknowledgements

The SCSI product on the Series 300 was to some extent a "grass roots" effort. My thanks to John Byrnes as the key person in helping to get the project off the ground. Adding a second disc interface to the long tradition of HP-IB was not easy and John worked hard in achieving this success. Evan James came onto the SCSI project as product marketing manager and turned out to be a tremendous success in keeping the project moving and balancing the lab's requirements with those of marketing. Evan and John did an excellent job driving the team to get the job done.

Thanks to Shaw Moldauer for slipping an SCSI interface into the Model 319 almost undetected and thanks to Dave Kinsell for designing the Model 350 board.

The debugging of the software during critical moments in its life cycle was helped by Steve Wolf and Drew Anderson. The bugs that are found when hardware and software are being developed simultaneously are sometimes very difficult to contend with. When megabytes of data are flowing over the bus every second and only one or two bytes are occasionally wrong, it is a challenge beyond belief. Steve's help at some of these moments of despair was invaluable.

People who are able to straddle that magical wall that separates hardware from software are special people to HP indeed. I was fortunate to work with an excellent team.

# X: A Window System Standard for Distributed Computing Environments

*The X Window System allows applications running in different environments and on different machines to communicate high quality, graphical user interfaces over a network.*

**by Frank E. Hall and James B. Byers**

The X WINDOW SYSTEM* has emerged as the industry standard for supporting windowed user interfaces across a computer network. It was originally developed at the Massachusetts Institute of Technology (MIT) as part of Project Athena, a large research project investigating networks of computing systems from multiple vendors. MIT has facilitated the acceptance of X as a standard by placing it in the public domain, distributing the standards definition documents and the source code of sample implementations for public use for a nominal fee.

The X Window System is network transparent, which means that an application running on one vendor's computer can display a high-quality, graphical user interface to a user sitting either at that same system or at another computer across the network, perhaps made by a different vendor. The location of the application's target display is not material to the application, and is determined by a parameter when the application is run.

X is virtually independent of the underlying hardware and operating system. The X software adjusts for differences in display or computer architecture automatically as the packets of interactive graphical information are exchanged between application and display according to a well-researched, efficient protocol. This protocol forms the heart of the X Window System standard. Since applications participate in this protocol through a standard programmatic interface library, applications written to the X library are highly portable to other computer systems that support X.

These features combine to make the X Window System a significant enabling technology that allows application developers, end users, and computer hardware vendors to explore the possibilities of the distributed computer environment relatively unencumbered by proprietary barriers that have prevented such seamless integration in the past. For application developers, X promises easier porting, which can allow them to reach a wider market while spending less time on porting and more time on writing better software. For end users, X promises more and better software, and more choice in hardware.

Accordingly, support for X has gained rapid momentum among hardware vendors. HP was among the very first computer manufacturers worldwide to sell X as a product when in March 1987 it began shipping the X Window System for HP-UX, HP's version of the UNIX® operating system. X is now publicly endorsed as a standard by over 40 computer vendors in the U.S.A., Europe, and Japan, including virtually every major manufacturer of UNIX work-stations.

The increasing power of the distributed computing environment, as demonstrated by the other articles in this issue, makes X a very timely technology. It has integrating implications for the areas of user interface, graphics, and networking. It also presents new challenges for addressing the emerging distributed computing market.

In this paper, we will compare the architecture of X to conventional window systems, and describe the industry efforts to support X as a standard.

## The Basics of Window Systems

A window system is a low-level set of interactive graphics primitives that provide an application with efficient means to create, manipulate, and destroy communication regions or windows on the user's display.[1] The application uses the primitives to send simple graphics or multifont text in color or black and white to the window.

The basic unit out of which the window system builds both text and graphics is the pixel, which is the smallest directly accessible graphical element of the display, usually a very small squarish dot. On a monochrome display the pixel's value can be represented by a single bit. On a color display, the pixel contains an integer color value consisting of multiple bits, depending on the color depth of the display. For high performance, the pixels are typically accessed by memory mapped I/O techniques. Displays of this type are generally referred to as bit-mapped displays.

The window system allows an application to own many windows on the bit-mapped display at the same time, and several applications to share access to the display simultaneously. The window system provides such basic output functions as clipping, drawing, text placement, color map management, and output multiplexing to multiple windows. It provides basic input functions by collecting and routing to the appropriate applications any events received from the user's input devices, which typically consist of at least a keyboard and a pointing device such as a mouse. This allows the applications to share the input devices

*The X Window System is a trademark of the Massachusetts Institute of Technology.

without having to engage in explicit arbitration among themselves.

Since user interface style is an evolving field, it is desirable that the window system remain free of a specific user interface policy so that it can efficiently implement alternatives. For example, the methods by which the user employs the input devices to direct the placement, movement, sizing, and shuffling of windows on the display, or to designate which window shall receive keyboard input, is a question of user interface policy that can be delegated to a higher level of software called a *window manager*. Similarly, the window system need not contain specific user interface components such as menus or scroll bars. These style building blocks can be delegated to a higher level of software called a *user interface toolkit*. Finally, the window system should not unduly restrict the type or number of simultaneous terminal emulators through which the user accesses window-dumb applications that were written to talk with a serial terminal. The ideal window system is able to support a wide variety of possible window managers, toolkits, and terminal emulators.

These items often accompany a window system and occasionally become entangled with its design. We will return to these items in more detail below with regard to window system architecture.

### Convential Window Systems

Conventional window systems allow window applications to access the display device directly through calls to the operating system kernel, which is often extended to facilitate arbitration of display resource conflicts and window system communication. These applications must therefore reside on the local system.

A schematic of a conventional window system architecture is shown in Fig. 1. Here window applications A and B, linked with the window system library, share access to the display while terminal-based application C, which normally talks to a serial terminal, appears through a window provided by a terminal emulator module for backwards compatibility with the time-sharing environment. While the terminal emulator must reside locally, C may reside either locally or on a remote system that has been accessed through a network service that simulates a serial connection. In either case, C has no knowledge that it is talking to anything other than an ordinary serial terminal. User operations to shuffle and arrange the visible windows are provided by the window manager module, which is tightly coupled to the kernel and communicates with the window system library code linked with each window application.

The window manager and terminal emulator modules are often so closely integrated with the window system that alternatives cannot easily be substituted. User interface components such as scroll bars or menus, while they may be present in the window manager and terminal emulator, are often not available to the application developer.

Conventional window system architecture is more varied and complex than this simple diagram can indicate. For example, the window manager and terminal emulator may be completely implemented in the kernel, or window management may be supported by redundant code linked with the window system library into each application.

While conventional window systems were a great leap forward from the terminal-based time-sharing environment, their greatest problem in a distributed computing environment stems from their greatest strength, which is the direct display access that they provide for applications. Since this requires that window applications reside locally with the display, they cannot be accessed on a remote system. To access a remote application the user must go through a terminal emulator, thereby dropping back to the previous era's user interface paradigm.

Conventional window systems are therefore inherently limited in the distributed computing environment by their stand-alone, non-networked design.

### The X Window System

The principal feature that distinguishes X from a conventional window system is its network transparency.[2] The X Window System allows window applications, or clients, to access the display only through the display server, which is a separate process that arbitrates resource conflicts and provides display, keyboard, and mouse services to all clients accessing the display. X can support a spectrum of hardware displays ranging from small monochrome units to advanced graphics systems with up to 32 bits of color per pixel.

The client and the display server exchange information only by means of the X Window System protocol which can be implemented via any reliable byte stream. In the HP-UX implementation of X, as in most others, this byte stream is implemented as a socket, which is a logical data connection between two processes on the network. Clients may reside locally with the display server, or on a remote system across the local area network (LAN). A performance optimization bypasses physical LAN access when the client and display server are local to each other.

Because the client program and the display server are two separate entities, the target display can be specified at the time an application is run. The client program is indifferent. It sends out X protocol commands, which the network services route to the target display server, which then executes the command.

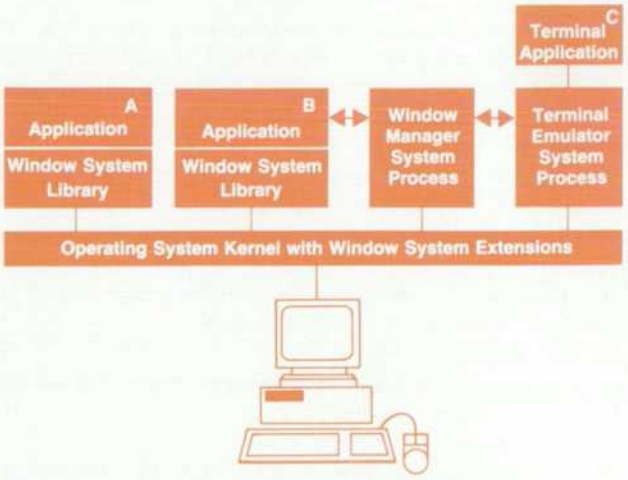



**Fig. 1.** *Conventional window system architecture.*

Note that the notion of a display server complements the notion of servers as commonly used in discussions of networks. Servers on the network provide applications with access to resources such as files, printers, or computational power. The display server rounds out that picture by making a given display on the network available to applications as a user interface resource.

In Fig. 2, window applications A and B, linked with the X Window System library, share access to the display while terminal-based application C again appears through a window provided by a terminal emulator client. User operations to shuffle and arrange the visible windows are provided by the window manager, which is an otherwise ordinary client whose function is to communicate with the display server to provide user interface policy for the user to manage the display layout. By definition the window manager does not specify how the user interacts within an application. In fact, applications can be written to execute properly with no window manager present. Sometimes this is desirable.

The window manager, user interface toolkit, and terminal emulator are cleanly separated from the X window system, so the user can substitute an alternative if desired, or even omit the item. Note that application B has chosen to use a user interface toolkit, while A has not.

Applications A, B, and C, the window manager, and the terminal emulator may be either local to the display server or on a remote system, or in any combination thereof. The only restriction is that the underlying operating system must support multitasking if the display server and a client are to reside on the same system.

For multitasking systems, it is customary for the window manager and display server to reside locally, and for the terminal emulator to reside on the system where its terminal-based application resides. A single-tasking system can execute only the display server or a single client at a time. In this case the display server can be used as a viewport onto the network. The window manager and all other clients can reside on computational servers elsewhere on the network.

The components of the X Window System standard itself are small in number. At the lowest level, it is simply a document that defines the X protocol.[3] At the programmatic level, it is a document that describes a standard programmatic interface, or window system library, by which an application participates in the protocol.[4] To facilitate ports of the X system, the MIT distribution contains source code of sample implementations of the X library and display server.

While the X library description distributed by MIT is defined for access from the C programming language, programmatic interfaces for other languages can be and have been developed. HP supplies a Fortran bindings package for the X library as part of its X Window System product.

### Window Manager

At the outset, students of window systems sometimes confuse a window manager with the window system. The following scenario illustrates the role played by the window manager working in conjunction with the capabilities of the X Window System. Fig. 3 shows how the display might look after the activities described in the scenario.

Suppose that the user has just brought up the X Window System through a script that will later start some client programs. At this point, only the X Window System is running, so the display shows only blank background, which is referred to as the root window. The system cursor, controlled by the mouse, rests in the center of the display. When the script starts the window manager for that display, the appearance of the display does not change. The script also starts two other client programs at the beginning of the session. One is a simple clock program that displays the current time in a corner of the screen. The second is a terminal emulator that opens a window on the display and waits for the user to type a command.

When the user presses the right mouse button, the window manager, which has requested the display server to notify it of all mouse events that occur when the cursor is directly over the root window, receives a notification of the event in its input queue. Let us assume that the user interface policy of this window manager honors a right button press over the root as a command to present a menu, the contents of which the user has specified in a start-up file. In actuality the meaning of this button event could be changed through the start-up file, which would allow a left-handed person, for example, to reverse the window manager's meaning for the left and right buttons.

After notification, the window manager prepares the menu contents and presents it on the display using a component from a popular user interface toolkit. Since this toolkit is also used by many applications, the user is familiar with the operation of this type of menu. The title of the menu is Launch, and it contains the names of the programs the user most often wants to start up. The user selects a program name from the menu. The window manager executes the instructions that the user specified in the start-up file as corresponding to that selection, which in this case is to start up a program directed to the local display that provides the user with a control panel that is used to

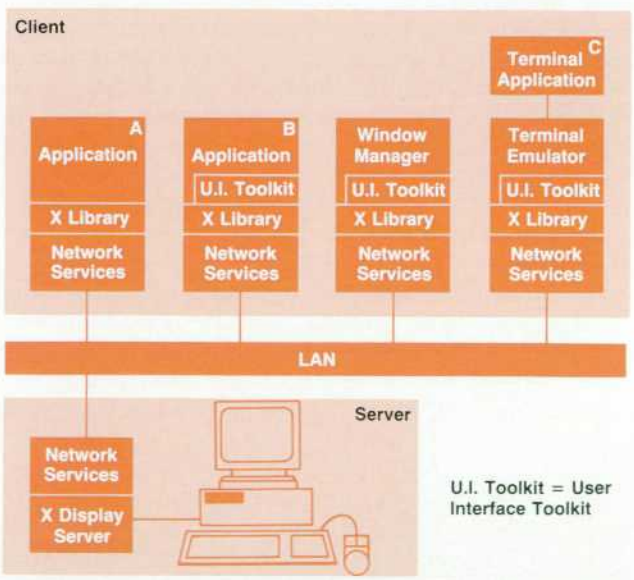



**Fig. 2.** *X Window System client-server architecture.*

monitor and direct various processes across the network.

The new program establishes contact with the display server, opens a window on the screen, and draws the control panel. When the user clicks the mouse over the appropriate buttons on the control panel, a simulation program begins on a large mainframe computer. The simulation program sends its graphical output across the network to a new window that it opens on the user's display.

While watching the simulation, the user uses the Launch menu to select a program that enables the user to browse through the contents of a remote file system, and then later the user brings up an application to read electronic mail. At this point the screen is too cluttered so the user iconifies one window by bringing up a menu specific to the window manager, and selecting the iconify option. This changes the appearance of the system cursor to indicate that the user needs to pick the window to iconify. The user does this by moving over to the terminal emulator window and clicking on it. The window manager immediately unmaps (removes without destroying) the terminal emulator window from the display, and as a placeholder puts in a convenient spot on the display a small, named, meaningful symbol of the application, called an *icon*. In this way the user also iconifies the electronic mail program. The user can restore these windows later and continue interacting with these applications.

### User Interface

User interface toolkits offer the programmer higher-level tools than the X library with which to program. As an example, in the X Window System itself there are primitives to draw lines, move rectangles of pixels, and so forth. Toolkits, on the other hand, provide the programmer with easy ways to create and manipulate useful interactive gadgets such as menus, buttons, scroll bars, text entry fields, and so on. These tools greatly reduce the effort that the programmer must put into creating the user interface portion of a program.

To promote acceptance of the X Window System as a standard, HP developed a user interface toolkit based on X, called Xrlib, and contributed it to the MIT public distribution of X Version 10.4 in December of 1986. HP subsequently enhanced this toolkit and ported it to the next revision, X Version 11. It provides a useful set of 13 interactive components called field editors, including pop-up walking menus, panels, scroll bars, title bars, a variety of buttons, and fields for entering, editing, and displaying text or graphical data.

Several user interface toolkits have now been contributed to the MIT public X distribution, offering a wide range of capability. Perhaps the most significant is Xt which is a set of low-level toolkit procedures called *intrinsics*. The Xt intrinsics were developed in a collaborative effort by Digital Equipment Corporation, Hewlett-Packard, Massachusetts Institute of Technology, and others. The intrinsics provide a flexible, powerful foundation upon which to construct interactive components, such as buttons, scroll bars, menus, and other items which are collectively called *widgets*. The X Consortium has voted to accept Xt as part of the X standard. HP is developing a useful set of widgets, based partly on the Xrlib functionality, that has been contributed to the MIT public distribution to promote acceptance of the Xt intrinsics.

### Terminal Emulator

The HP X Window System product includes hpterm, a terminal emulator that approximates an HP terminal, complete with softkeys. This allows the HP workstation user to access a broad range of terminal applications that are compatible with this generation of HP terminals.

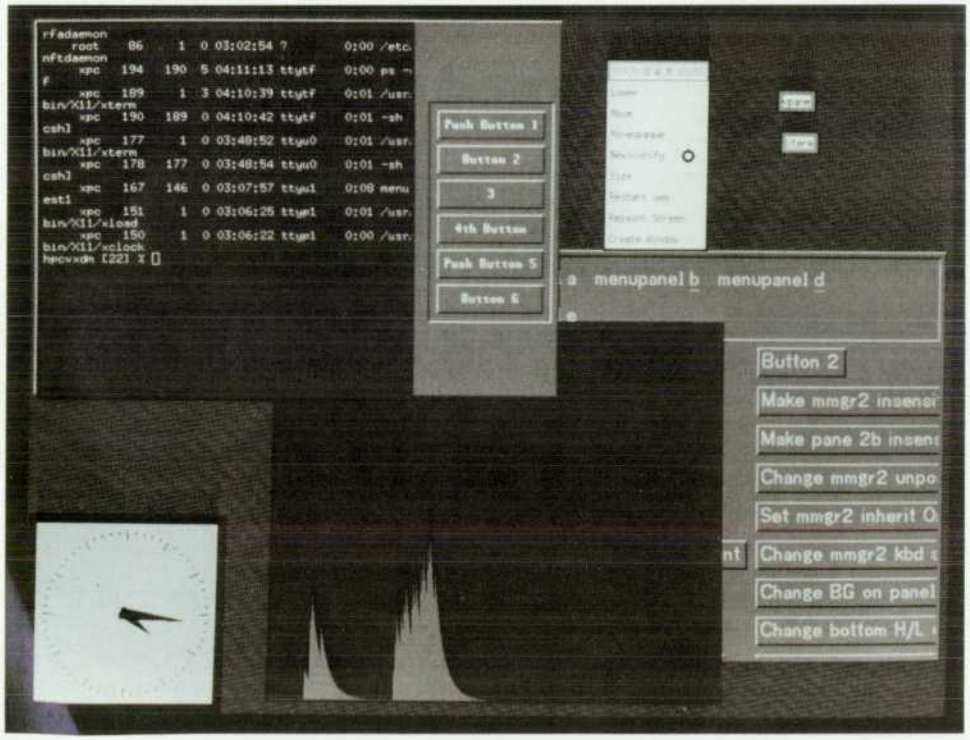



**Fig. 3.** *Final screen for the scenario presented in the text. The* Launch *menu and the menu containing the* iconify *command are pop-up menus; they are called up by clicking the proper mouse button and go away after a selection is made. The two icons for the terminal emulator and electronic mail applications are in the upper right corner.*

The MIT X distribution contains source code for a Digital Equipment Corporation VT100 and a Tek 4010 terminal emulator called xterm, which is also included in HP's X Window System product. It allows applications written for these older Digital Equipment Corporation and Tektronix Inc. terminals to be accessed directly from the HP workstation display.

The HP workstation user can activate any number of these terminal emulators in any combination, choosing the right one for the application to be accessed.

### Support

The 1986 release of X, Version 10.4, was the first version with multivendor support. While this has allowed solution creators to begin development of X Window System solutions, developers soon recognized that to make X a solid standard, there was a need for increased capability and backwards-compatible extensibility of the X protocol to incorporate new functionality that might arise. This protocol extensibility would allow X to improve without breaking applications written earlier. Such enhancements to the design of the protocol, X library, and display server resulted in Version 11 of X, the sample implementation of which MIT formally released in March, 1988.

While the technical innovations in X are quite impressive, what really sets X apart from other window systems is its openness as a standard and its broad base of support.[5,6,7,8] in coordination with MIT, a consortium of companies has been formed to define enhancements and to divide the engineering effort needed to develop standard descriptions and sample implementations of those enhancements. MIT chairs the consortium but the consortium defines the extensions to X. This should ensure the stability and broad support of X for the foreseeable future.

MIT and the X Consortium administer the release of the public-domain X Window System code as well as the contributions of the various supporting vendors. In this way the software and enhancements are available to all interested parties at the same time.

X Consortium membership includes HP, Apollo Computer Inc., Apple Computer Inc., American Telephone and Telegraph Co., Control Data Corp., Digital Equipment Corp., International Business Machines Corp., Sun Microsystems Inc., Tektronix Inc., and Xerox Corp. This list represents a large segment of computer vendors in the technical market. Software vendors formally supporting the X Window standard include Adobe Systems Inc., Applix Inc., and Cognition Inc.

X has been chosen by the X/Open committee, a group that adopts UNIX operating system standards for a consortium of U.S.A. and European computer vendors. Following this lead, the recently formed Open Software Foundation has adopted X as its window system standard. Work is also occurring for formal acceptance of X by other standards groups.

X is the beginning of a new generation of software and systems design that takes a significant step forward in the era of distributed computing environments. A seamless integration of services in these multivendor environments now appears possible, allowing the scaling of computers to their appropriate tasks while maintaining open, productive access to their functions. A standard user interface style, which would allow the easy porting of users between computing systems and between applications, may not be far behind.

### References


1. D.S Rosenthal, "Toward a More Focused View," *UNIX Review*, June 1986, pp. 54-63.
2. R.W. Scheifler and J. Gettys, "The X Window System," *ACM Transactions on Graphics*, Vol. 5, No.2, 1986, pp. 79-109.
3. R.W. Scheifler, *X Window System Protocol, X Version 11, Release 2*, Massachusetts Institute of Technology, September 1987.
4. J. Gettys, R. Newman and R.W. Scheifler, *Xlib - C Language X Interface, X Version 11, Release 2*, Massachusetts Institute of Technology, September 1987.
5. "11 Companies Back Windowing Standard," *Electronic Engineering Times*, January 19, 1987, pp. 1,16.
6. "The Advantages of X," *Computer Graphics World*, August 1987, pp. 57-60.
7. "DEC, HP, Nine Others Adopt MIT X Window as Standard," *Electronic News*, January 19, 1987, pp.1,6.
8. E. Lee, "Window of Opportunity," *UNIX Review*, June 1988, pp. 47-61.

# Managing the Development of the HP DeskJet Printer

*Forays into unexplored regions of technology are inevitable in the development of breakthrough products, but they must be limited and carefully managed.*

**by John D. Rhodes**

THE CREATION OF A HIGH-TECHNOLOGY PRODUCT is an enterprise that requires the contributions and skills of many people. The articles on the HP DeskJet printer in this issue mainly explore the technical engineering problem solving that is essential to new-product development. But there is a bigger picture. There is much more to engineering than solving equations and setting up experiments.

In the case of the DeskJet printer, our organization and planning encompassed all of the functional departments of HP's Vancouver Division. Our development and management teams included members from R&D, manufacturing, marketing, and quality assurance, with special assistance from personnel and finance.

The core development teams, which worked on the product from its inception, consisted of about 25 engineers, split into three project teams—firmware, electronics, and mechanical—with a project manager leading each team. The three core project managers reported to a laboratory section manager who served to coordinate the entire project.

Within the lab, midway through the development process, additional teams of two to five engineers were formed. These teams developed character fonts, emulation software, and application drivers, and performed extensive verification and performance testing.

During the first year of development, the core management team's role was directed towards technical guidance, resource organization, planning, and progress tracking. In the second year of development, the management team's emphasis shifted to coordination and prioritization as the circle of people involved in the program grew larger, and as the date for product introduction grew closer.

Our project started with a loose collection of specifications—a list describing what features we felt customers needed—tempered by what we felt we could technically achieve given our time, people, and financial resource levels. In the broadest sense, the goal of the project teams was to transform that list into an engineering specification.

### DeskJet Printer Features

The HP DeskJet printer, Fig. 1, is a personal-convenience printer that produces laser-quality output at a price comparable to other low-cost personal printers. Among its features are 300-dot-per-inch resolution, merged text and

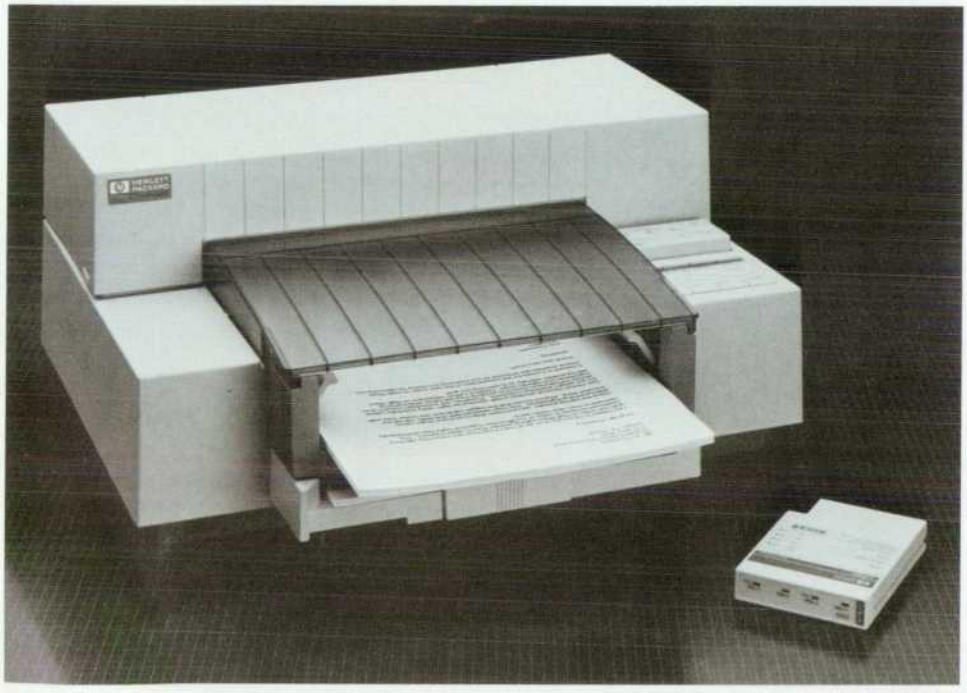


**Fig. 1.** *The HP DeskJet printer produces laser-quality output at a low price for personal computer users. It prints on standard office papers.*

graphics, multiple fonts, two slots for font or personality cartridges, 120-character-per-second letter-quality speed, built-in cut-sheet feeder for common office paper, desktop design, and quiet operation.

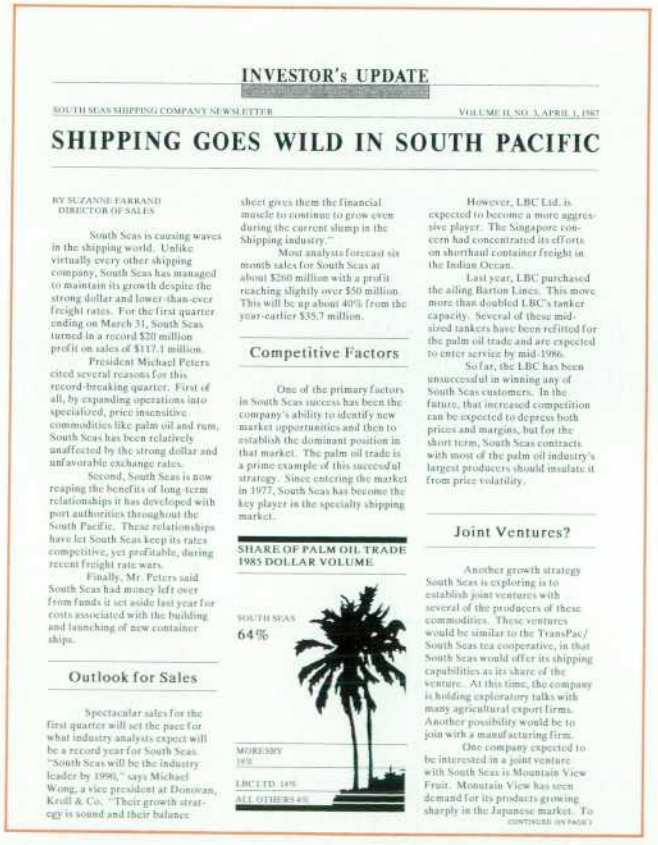
The DeskJet printer comes with Centronics parallel and RS-232-D interfaces. It is supported on HP Vectra, Portable, and Touchscreen personal computers, HP terminals, the Apple II series, IBM PC/XT, PC/AT, and PS/2 computers, and compatibles. It is also supported on HP 3000, HP 1000 A Series, HP 9000, and HP 260 systems.

Many applications software packages, such as spreadsheet and word processing programs, support the DeskJet printer. For other packages, the HP LaserJet Series II printer driver will work well because the DeskJet printer uses the HP PCL (Printer Command Language) Level III command set. An Epson FX-80 driver will also work if used with the optional HP Epson DeskJet personality cartridge, which fits into one of the DeskJet printer's option slots.

### Technical Challenges

The development path is never smooth or level. There are steep hills and traverses across unexplored territory. The technical challenges on the DeskJet printer project were many:

- Extend the 96-dpi inkjet printing technology to 300 dpi, keeping the printing speed above 120 cps
- Ensure that the operation of the ink delivery system is totally transparent to the user, eliminating the messy image that inkjet printing had inherited from its early days
- Make this printing technology available on all standard office papers, eliminating the dependency of inkjet printing on special papers
- Provide these product features in a small-footprint package that does not dominate the desk on which it sits
- Make this product of traditional HP quality, with reliability unrivaled by any other printer
- Accomplish all of this within a tight development schedule of 22 months with a design that can be built, distributed, and profitably sold for a low target price.



INVESTOR's UPDATE

SOUTH SEAS SHIPPING COMPANY NEWSLETTER VOLUME II, NO. 3, APRIL 1, 1987

SHIPPING GOES WILD IN SOUTH PACIFIC

BY SUZANNE FARRAND
DIRECTOR OF SALES

South Seas is causing waves in the shipping world. Unlike virtually every other shipping company, South Seas has managed to maintain its growth despite the strong dollar and lower-than-ever freight rates. For the first quarter ending on March 31, South Seas turned in a record $20 million profit on sales of $117.1 million.

President Michael Peters cited several reasons for this record-breaking quarter. First of all, by expanding operations into specialized, price insensitive commodities like palm oil and rum, South Seas has been relatively unaffected by the strong dollar and unfavorable exchange rates.

Second, South Seas is now reaping the benefits of long-term relationships it has developed with port authorities throughout the South Pacific. These relationships have let South Seas keep its rates competitive, yet profitable, during recent freight rate wars.

Finally, Mr. Peters said South Seas had money left over from funds it set aside last year for costs associated with the building and launching of new container ships.

#### Outlook for Sales

Spectacular sales for the first quarter will set the pace for what industry analysts expect will be a record year for South Seas. "South Seas will be the industry leader by 1990," says Michael Wong, a vice president at Donovan, Kroll & Co. "Their growth strategy is sound and their balance sheet gives them the financial muscle to continue to grow even during the current slump in the Shipping industry."

Most analysts forecast six month sales for South Seas at about $260 million with a profit reaching slightly over $50 million. This will be up about 40% from the year-earlier $35.7 million.

#### Competitive Factors

One of the primary factors in South Seas success has been the company's ability to identify new market opportunities and then to establish the dominant position in that market. The palm oil trade is a prime example of this successful strategy. Since entering the market in 1977, South Seas has become the key player in the specialty shipping market.

SHARE OF PALM OIL TRADE
1985 DOLLAR VOLUME

SOUTH SEAS 64%

MORESBY 19%

LBC LTD. 14%

ALL OTHERS 4%

However, LBC Ltd. is expected to become a more aggressive player. The Singapore concern had concentrated its efforts on shorthaul container freight in the Indian Ocean.

Last year, LBC purchased the ailing Barton Lines. This move more than doubled LBC's tanker capacity. Several of these mid-sized tankers have been refitted for the palm oil trade and are expected to enter service by mid-1986.

So far, the LBC has been unsuccessful in winning any of South Seas customers. In the future, that increased competition can be expected to depress both prices and margins, but for the short term, South Seas contracts with most of the palm oil industry's largest producers should insulate it from price volatility.

#### Joint Ventures?

Another growth strategy South Seas is exploring is to establish joint ventures with several of the producers of these commodities. These ventures would be similar to the TransPac/South Seas tea cooperative, in that South Seas would offer its shipping capabilities as its share of the venture. At this time, the company is holding exploratory talks with many agricultural export firms. Another possibility would be to join with a manufacturing firm.

One company expected to be interested in a joint venture with South Seas is Mountain View Fruit. Mountain View has seen demand for its products growing sharply in the Japanese market. To

CONTINUED ON PAGE 9

**Fig. 2.** *A page of DeskJet printer output.*

### Design for Reliability

The hallmark of a successful project is careful risk management, for tough or recalcitrant problems require intensive resources to solve. If the development team is considered as a problem-solving engine, then that engine has a specific capacity, and for a given complement of engineers, there is a limit to the number and difficulty of technical problems the engine can solve in a given time period.

With follow-on products, the design task is that of interpolating from a well-understood basis, peaking performance, adjusting features, or reducing costs.

In breakthrough products, the design task must include solutions that use unfamiliar technologies. It is these forays into unexplored regions that must be carefully limited to essential development, since there is little experience to guide progress or to gauge the potential difficulties. In other words, the design teams must carefully choose which problems they are going to solve.

An example will help to illustrate this point. Early in the DeskJet development project, the mechanical design team elected to use filled thermoset plastic for the major structural part (the chassis). This decision was based on experience with similar structural plastic parts in several HP Divisions. In fact, the material set chosen for the structure and gears was identical to that successfully used in the PaintJet printer. The DeskJet team sought to reduce its design load by using an existing and well-understood technology. Or so we thought! The first prototype printers assembled from the molded plastic parts showed rapid deterioration of bearing materials with resultant squeaking, galling, and seizing—often within a few tens of pages. It turned out that we had exceeded a critical PV (pressure-velocity) point in the bearing loads. Until solutions were found (it took several intensive weeks), all design teams were hampered in their development by a lack of working prototype printers.

In our laboratory, team techniques are an important contributor to rapid progress and reliable design. Development tasks always have a principal designer and a subsidiary designer. The principal designer has part responsibility, while the subsidiary designer is a valued consultant. This designer pairing is based on interacting part/subsystem functions. Thus, a web of pairings exists, connecting the designer teams.

This pairing has several advantages over solitary design. The synergism of two (or more) designers working on the same problem is remarkable, and the quality and quantity of potential solutions is superior. In addition, solutions always have two committed designers to argue their merits with the rest of the design team. Furthermore, the principal

designer has a backup in the event of illness, a trip, or reassignment to another design task.

We also encourage informal and frequent design reviews. Typically, these occur when the designers have a concept worked out that is supported by preliminary analysis. The designers meet with those who have expertise in the area (in mechanical designs usually including a procurement engineer and a manufacturing engineer), and walk through the design with their peers. It is important that these reviews occur early in the design so ideas and suggestions can be incorporated easily.

These design reviews occur close in time to the development of the first crude prototypes. Initially, the early mockups examine a subsystem or specific function, such as picking paper with a platen roller. Later refinements are incorporated in a product breadboard—a working printer that demonstrates all of the critical subsystem functions.

### Testing the Design

Coincident with the emergence of the prototypes is testing. First the basic concept is examined to see if it addresses all of the design constraints. Next, normal operations are explored to find where the design is deficient. (It always is!) Finally, the design limits are probed through accelerated tests or abuse testing.

Throughout, the engineer repeatedly cycles through the test-analyze-fix process. Initial testing yields many easily discovered defects, which can be quickly resolved with engineering analysis. Subsequent testing is aimed at improving ruggedness and reliability; these test scenarios usually require many unit-hours of experience, and the conclusions must be reached by careful statistical inference.

Compounding the statistical problem is that of securing representative parts. Much of the initial testing is performed with parts from prototype tools or processes. The design limits are not well-understood at this point, and the parts are varying because the process is still unstable and undeveloped. Tolerance analysis of the design is useful, but not sufficient, since the called-for tolerances must be satisfied by a production process. Much effort goes on at this time to allocate the tolerances and allowances between parts and process.

The final phases of testing involve tests under controlled conditions by impartial quality assurance engineers. Here, testing to rigorous HP standards is completed, including temperature and humidity excursions, shock and vibration tests, and transportation and use/abuse tests that seek out the weak links in the design.

Life testing proceeds under accelerated and nonaccelerated conditions, probing the design for deterioration, wearout, and contamination.

In summary, reliable design requires more than theoretical design skills and analyses. Although good first-round designs are an essential foundation, the bulk of the engineer's efforts go into executing well-thought-out testing programs whose intent is to stress the design and uncover its limitations so that improvements in the subsystems and the integrated product can be made.

## Market Research as a Design Tool

The DeskJet story is filled with thoughtful responses to design challenges, as are many product development histories. But in this case, the product is all the more successful because of the design team's speedy reaction to market research feedback, thus enabling the product to deliver a key benefit long sought by customers, but never before achieved.

For several years, low-end printer customers have been expressing the desire to have a "printer on my desk, comparable in cost to other low-cost personal printers, that produces output like a laser printer." Almost without exception, in qualitative research sessions (focus groups) conducted by the HP Vancouver Division, individual users expressed preference for the convenience of their own printer, on the desk, for low-volume printing to support their own work. Although inkjet printers were perceived by many as unreliable, it was discovered that customers really judge printers by the benefit they deliver—the quality of the output.

Thus was born the DeskJet concept. The challenge was to use thermal inkjet technology to satisfy that commonly heard wish for an inexpensive laser printer for the desktop, but to do so in a way that minimizes the inkjet technology issue and breaks through the clutter of the low-end printer market.

The challenge really boiled down to two components: (1) could the print quality be made good enough in a short enough time and at the right cost, and (2) how could we communicate the key benefit of the product, that is, what position should it occupy in the mind of the prospective buyer? To measure progress on the first point, early prototypes were taken to more focus groups, and print samples were taken to shopping mall test sites in late 1986 and early 1987. The results were discouraging. Even though we were proud of our achievements to date, printer users were not impressed. Specifically, the print was not black enough and not sharp enough. We learned exactly where we stood on a numerical scale with laser printers, daisy-wheel printers, and a major competitor: dead last. The product did not deliver the desired "laser printer on my desk."

After several months of intense work on ink formulation and font design, the DeskJet printer and print samples were again taken to shopping malls for testing. The print quality improvements were dramatic. Many respondents asked if the output came from a laser printer. We finally had what we felt customers had been asking for. Now the task was positioning the product in a fashion to communicate the message, in everything from advertising to training to sales tools to public relations efforts.

A key market research effort consisted of extensive telephone interviews of over eight hundred printer users. It showed that there is a large segment of users who yearn to own a laser printer but never really intend to purchase one. They may drop back to a low-priced impact printer, usually one of the 24-wire models, because they can't afford or justify a laser printer. A marketer's dream was born: to position a low-cost product around the benefits of an upscale product. And since the design team had done an extraordinary job of controlling cost and keeping on schedule, the DeskJet printer was brought to market ahead of the competition at a price comparable to other personal desktop printers that do not deliver laser quality.

The idea of laser-quality output for a personal-printer price has been so effective in communicating the long-sought benefits of the product that the DeskJet printer received over 26,000 orders in its first month.

*Alan Grube*
Product Marketing Manager
Vancouver Division

## Human Factors and Industrial Design of the HP DeskJet Printer

In refining the ergonomics of the DeskJet printer, our objective was to design a printer that offers simplicity of operation and convenient access to the user-interface areas. In addition, we strove to make using the printer as intuitive or self-explanatory as possible.

After extensive focus group studies and research, it was decided that DeskJet printer users have four basic printer requirements: high print quality, ease of use, compact size, and affordability. In all phases of the project, these user needs and human factors were important considerations in the decision-making process. The key design areas that were impacted were the paper path, control panel, printhead, setup, and switches.

The paper path is the most obviously different feature of the DeskJet printer. It is based on the concept of "front in, front out." This concept is both convenient for the user and efficient in design, effectively contributing to the small footprint of the printer. Loading and unloading paper require no unnecessary adjustments to align the paper or set it up for feeding. The paper is simply set in place and all feeding and aligning are done by the printer automatically.

The control panel is the most often used feature on the printer. As such, it needs to be well thought out and executed. Simplicity in function is important, but simplicity in appearance is equally important. If the control panel looks complicated, even though it isn't, the user will perceive it to be complex and may be intimidated.

The keys on the DeskJet control panel are divided by size and color into two groups. The primary group consists of four basic function keys that are regularly used, while the secondary group consists of four keys occasionally used for special functions. Color and LED annunciators provide a visual link between functionally associated details, allowing the operator to make quick visual associations rather than having to study the control panel. Font cartridges that are accessed through the control panel carry the same theme of visual association through the use of color and LED annunciators.

Along this same line of reasoning, we know that the look of the printer contributes to the user's perception of its value. For example, small size may signal the observer that the product is low-end, perhaps even cheap. When one designs a compact printer, special attention must be paid to details and styling so that a compact printer is still perceived as a printer of value. Thus, detail development and the calculated addition of a cosmetic output-tray cover help improve the user's perception of the combined price and performance value.

Focus group studies and feedback from customer warranty return cards revealed that customers were often critical of the mess and inconvenience associated with changing ribbons, printwheels, and print cartridges. With this in mind, we designed the DeskJet printer so that the print cartridge and paper path are easy to access and require a minimum of handling or special care.

The paper path is accessible by simply lifting the paper output tray and cover. The access door can then be opened to expose the entire paper path.

The print cartridge is loaded into the printer by inserting it into the carriage and rotating it forward. There is no secondary latching operation. Cartridge maintenance is accomplished through the use of an internal priming pump which is actuated from the control panel. Once the pen is installed, it is not handled again until it is replaced.

We also felt that access to the DIP switches and the power switch was important. Therefore, we located them on the front panel.

In keeping with our ease-of-use objective, we installed a user instruction label under the access door. The purpose of this label is twofold. First, it allows the user to hook up the printer and put it into operation, and second, it provides a ready reference for basic reconfiguration.

In a broad sense, ease of use applies not only to the printer but also how efficiently the printer uses the work surface. The objective is to make the footprint as small as possible, but that is not the only issue. What really needs to be considered is the projected area of the printer on the work surface. This includes paper trays, connectors, and so on. To make the DeskJet printer as compact as possible we did two things. First, we stacked the input and output paper bins and second, we recessed the I/O connectors. The printer can now be placed with the rear panel against a wall.

*Don McClelland*
Product Design Engineer
Vancouver Division

**Acknowledgments**

Thanks to Tom Braun, the DeskJet section manager, whose drive and insights guided the entire project, Bob McClung, under whose capable management the electronics team delivered their reliable, high-performance design without ever being on the critical path, and Mark DiVittorio, whose firmware team simultaneously satisfied the requirements of two printer projects. Special thanks to Susan Hoff, the DeskJet project coordinator, for her great skills in maintaining documentation order in the midst of development chaos, and to our model makers, who contributed valuable suggestions and worked many hours of overtime producing the mechanical prototypes of the printer. Special credit goes to all the members of the design team for their enthusiastic dedication to the development of the DeskJet printer and its predecessor projects. Finally, I want to acknowledge the outstanding contributions of Bill Buskirk and Niels Nielsen of our Corvallis Inkjet Components Operation, without whose professional and personal commitment we could never have effectively coupled the development of the printer and printhead.

# Development of a High-Resolution Thermal Inkjet Printhead

*The HP DeskJet printer's 300-dot-per-inch resolution is fundamental to its ability to produce laser-quality output.*

**by William A. Buskirk, David E. Hackleman, Stanley T. Hall, Paula H. Kanarek, Robert N. Low, Kenneth E. Trueba, and Richard R. Van de Poll**

IN THE SPRING OF 1984, Hewlett-Packard introduced the HP ThinkJet printer[1] with its replaceable thermal inkjet printhead. This was a revolutionary concept that validated the use of inkjet printing in a low-priced printer. The resolution of 96 dots per inch was better than that of the existing 9-wire printers (72 dots per inch), and provided better-formed characters.

The resolution was determined by two limiting factors. Lower resolution would have required larger drop volumes, which are difficult to eject, and the limitations of orifice plate contruction prevented higher resolutions from being achieved. The required drop volume at 96 dots per inch was minimized by the use of a specific paper, which spreads the ink and produces large dots.

In the summer of 1987 the next generation of Hewlett-Packard's thermal inkjet technology was introduced in the PaintJet Color Graphics Printer.[2] In addition to providing color output, this printer offers improved text print quality at a resolution of 180 dots per inch, almost twice that of the ThinkJet printer. Because of the possibility of paper-induced variations in color, the inks in the PaintJet printheads were also optimized for use with a specific paper. This ensures colors that are brilliant and consistent.

In parallel with the development of the PaintJet printhead at HP's San Diego Division, another team was busy at work at the Inkjet Components Operation in Corvallis, Oregon and at the printer division in Vancouver, Washington. Market research indicated that an inexpensive printer that could produce letter-quality output at an acceptable speed on commonly available office papers would be successful. Daisy-wheel printers were slow or expensive, and laser printers were rapidly redefining the meaning of letter-quality. The new products that were expected to meet this need were the 24-wire dot matrix printers. This led to the goal for the HP DeskJet printer design team: laser-quality output on "plain" paper for about the same price as the new 24-wire printers. Again, an increase of almost two times in resolution, from 180 dots per inch to 300 dots per inch, was required.

The higher dot resolution offered some real challenges to the technology. To print high-resolution characters at speeds greater than those typical of lower-resolution printers, drop ejectors must operate at a higher frequency. Fortunately, higher resolution requires smaller drop volumes and smaller drops are easier to fire at higher frequencies. Higher resolution also requires that each drop ejector be fired more times during the life of a printhead. This required the development of improved thin films that could better withstand the cavitation of the collapsing vapor bubble.

Higher-frequency operation creates more residual heat. The low-thermal-conductivity glass substrate used in the ThinkJet printhead is not capable of removing this increased heat from the resistor area. For the PaintJet and

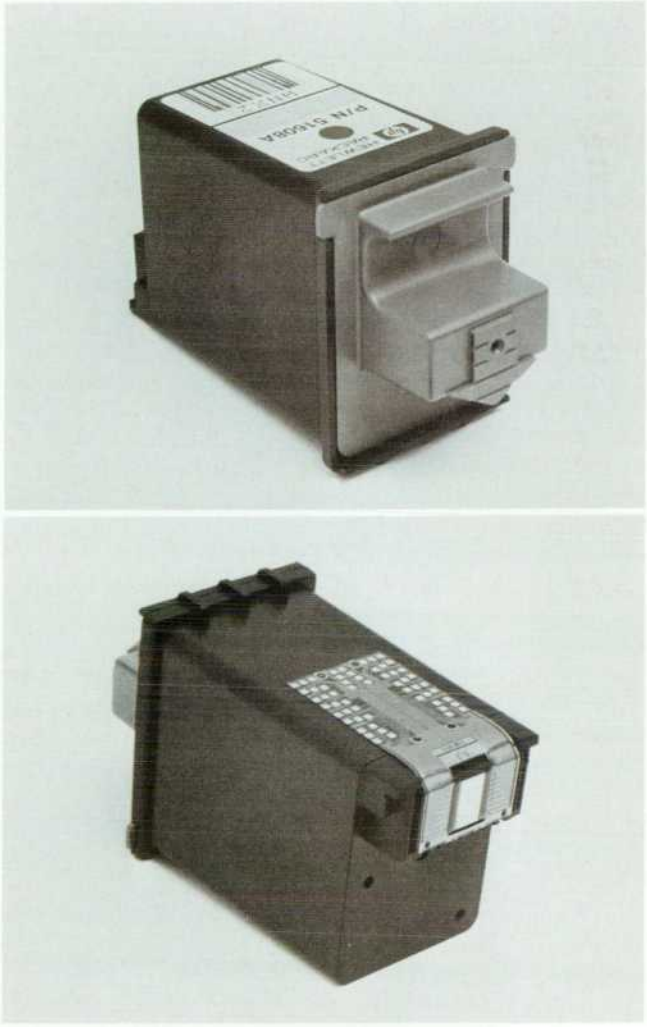


**Fig. 1.** *The 300-dot-per-inch print cartridge for the HP DeskJet printer.*

A portion of this paper appeared in the *Japan Hardcopy '88 Advance Printing of Paper Summaries*. 

DeskJet printheads, silicon was chosen for its excellent thermal conductivity. It had the additional benefit that commercial wafer handling and processing equipment was readily available.

In the ThinkJet printhead design, the orifice plate resolution was limited to 96 dots per inch. In the DeskJet printhead, 300 dot-per-inch resolution was achieved by separating the nozzles into two columns and removing the resistor separator barriers from the orifice plate. These barriers are now provided by a polymer, which is applied over the thin films and patterned before the orifice plate is attached.

Fig. 1 is a photograph of the DeskJet printhead.

### Ink and Paper

If existing printing technologies are examined, it is clear that no current office product actually prints on the incredible variety of paper stocks found in the typical business office. Duplicator machines require a paper capable of absorbing solvents, mimeographs need rough papers to improve the feed characteristics of the sheets and prevent binding, electrophotography requires papers capable of accepting a pigmented thermoplastic, and writing requires cotton bond papers for superior quality, whiteness, and absorption of inks from pens. Any paper manufacturer will point out that there is no such thing as "plain" paper.

As a consequence, one of our first tasks in developing the DeskJet printer was to determine the types of papers that would be considered the target for the product. Samples of papers from throughout the world were collected. Members of the ink development team visited paper company development laboratories, and joint work in understanding the mechanism of ink penetration into paper began. It became clear that thermal inkjet pens print in a manner very similar to that of fountain pens. The ink is placed upon the surface of the paper (in this case through the expulsion of a droplet) and it then both evaporates and penetrates to make a permanent mark. It was also found that the penetration rates of inks into paper vary widely. Papers with low penetration rates generally have high inkjet print quality, while those with very high rates have degraded print. In fact, the paper designed for the ThinkJet printer was found to provide extremely poor print quality with inks that looked promising on normal office papers.

A plain-paper inkjet ink must satisfy several criteria. The drying time of the ink needs to be short enough to allow the customer to handle the page during the time of printing without unnecessary care. The ink needs to provide a dark image similar in optical density to that of the best electrophotography. Fading needs to be unnoticeable in standard office environments. Acceptable print quality needs to be obtained on a wide variety of papers, such as copy and bond papers. The ink needs to be consumer-safe and pose no chemical risk to users of the printer. Materials compatibility demands that the ink be designed to avoid corrosion of or chemical attack on the printhead and printer.

From experience with the ThinkJet printer, it was apparent that achieving quality print on plain papers would be challenging. The mixture of glycol and water used as a vehicle in ThinkJet ink is designed to be ejected from an uncapped printhead and form an acceptable dot on inkjet paper. The ink is therefore designed to avoid a hard plug, or clogging of the nozzles, as evaporation of some of the ink components takes place. In the case of the new DeskJet printhead, it was determined that the appropriate solvent mixtures would not remain liquid if the nozzles were left uncapped for an extended time. For this reason, a capping/service station as discussed in the article on page 62 was required. Improved print quality on bond papers was first established through the observation that as the concentration of glycol in an ink is decreased, the shape of the dot formed on paper becomes more regular. At the same time, the total area of the dot decreases. By increasing the resolution to 300 dots per inch and modifying the ink solvent system, an optimized solution was achieved.

About this time in the project, the target paper set became better defined. It was determined that print quality on bond papers was important, but equal interest was developed in standard copy papers. From more than 300 reviewed worldwide, a set of eight papers was selected to represent the spectrum of print quality for the ink and printhead. The ink was reformulated and optimized to improve copy-paper print quality without reducing the time to the first misdirected dot (clogging of the nozzles). The dye loading was adjusted to provide an optical density equivalent to that of the HP LaserJet printer on high-quality papers.

### Ink Manufacturing

Currently, HP manufactures its own ink for the DeskJet printer. This involves a moderate-scale manufacturing facility and several state-of-the-art process steps. The ink is tested both during and after manufacture using an HP 5880 Gas Chromatograph and an HP 9000 Series 300 workstation to observe the solvent composition. Dye purity is tested using an HP 1090 Liquid Chromatograph and a Series 300 workstation, and absolute concentration is measured using an HP 8450 UV/Vis Spectrophotometer. Viscosity, surface tension, conductivity, pH, and specific ion concentrations are monitored as well.

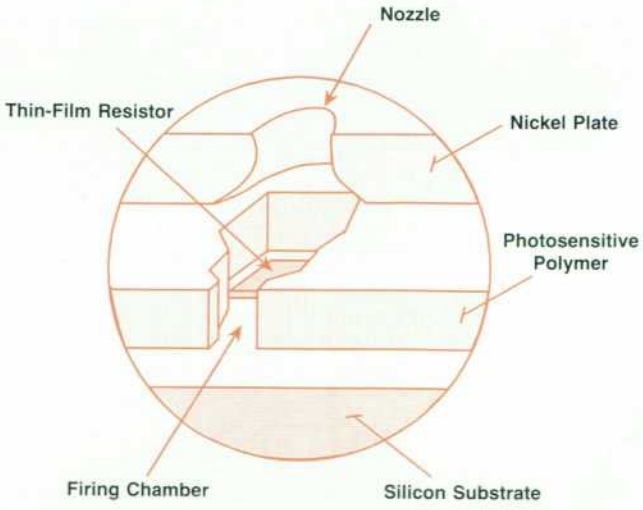



**Fig. 2.** *Cross section of an inkjet nozzle.*

### Firing Chamber Design

The basic printhead structure for HP thermal inkjet printers consists of a resistor surrounded by an ink channel, with both elements closely aligned to an exit nozzle (Fig. 2). When a resistor is heated, the adjacent ink is vaporized to create a drive bubble. This forces an ink droplet out through the nozzle. After the droplet leaves and the bubble collapses, capillary force draws ink from the feed channel to refill the nozzle. Each DeskJet printhead has 50 of these structures (Table I).

**Table I**
HP Thermal Inkjet Printhead Evolution

| | ThinkJet | PaintJet | DeskJet |
|---|---|---|---|
| Resolution (dpi) | 96 | 180 | 300 |
| Number of Nozzles | 12 | 30 | 50 |
| Frequency (Hz) | 1250 | 3000 | 3600 |
| Paper Type | Inkjet | Inkjet | Plain |
| Print Quality | Draft | NLQ | LQ |
| Drop Volume (pl) | 220 | 90 | 130 |
| Drops per Pen (million) | 10 | 108 | 120 |
| Ink Containment | Bladder | Foam | Foam |
| Interconnect | Paper Plane Bumped Flex | Paper Plane Springform | 90° to Paper Bumped Flex |

The early prototypes of the DeskJet printheads were designed for use with a specific inkjet paper. When printheads were first filled with plain-paper ink, they exhibited erratic performance and produced poor-quality print. Our testing showed that this was because of the lower viscosity of the new ink, which decreased the amount of fluid damping present in the printhead structure.

In an underdamped system, fluid rushes back into the inkjet nozzle area so rapidly that it overfills the nozzle, creating a bulging meniscus. The meniscus then oscillates about its equilibrium position for several cycles before settling down. A finite-difference fluid dynamics simulation shows that the meniscus configuration can have a dramatic effect on drop ejection (Fig. 3). Extra fluid in the bulging meniscus (Fig. 3a) adds to the volume of the emerging drop, while a retracted meniscus (Fig. 3b) reduces the volume of the drop. Fluid acceleration during bubble growth varies inversely with meniscus height, causing emerging drops to pull off-axis (Fig. 3c) or flare out (Fig. 3a).

An obvious approach to improving the damping of the system is to increase the fluid resistance of the ink refill channel. The easiest way to do this is to lengthen the channel. Fig. 4 shows damping for the initial channel length and Fig. 5 shows damping for a channel twice as long. Doubling the channel length yields a small increase in damping but the system now oscillates with a longer period. This means that a longer wait is required before a second drop can be fired, resulting in a reduced operating frequency. Fig. 5 also shows that the longer barriers reduce fluid interactions or cross talk between neighboring nozzles. Cross talk has an effect on the meniscus position, which in turn modulates drop ejection.

An alternative way of increasing the resistance of the channel is by decreasing the channel cross section (Fig. 6). This structure is well-damped and has a quick response. This means that a second drop can be fired immediately upon refill, resulting in a high-frequency system with well-controlled drop volumes. The decreased cross section also reduces system cross talk.

Many other problems had to be resolved to obtain an optimized printing structure. Computer modeling provided insight into areas where direct measurement was difficult. It also allowed evaluation of different structures without actually building them. Fig. 7 shows a simulated drop running into a residual droplet of spray left by a previously fired drop. This event would have been extremely difficult

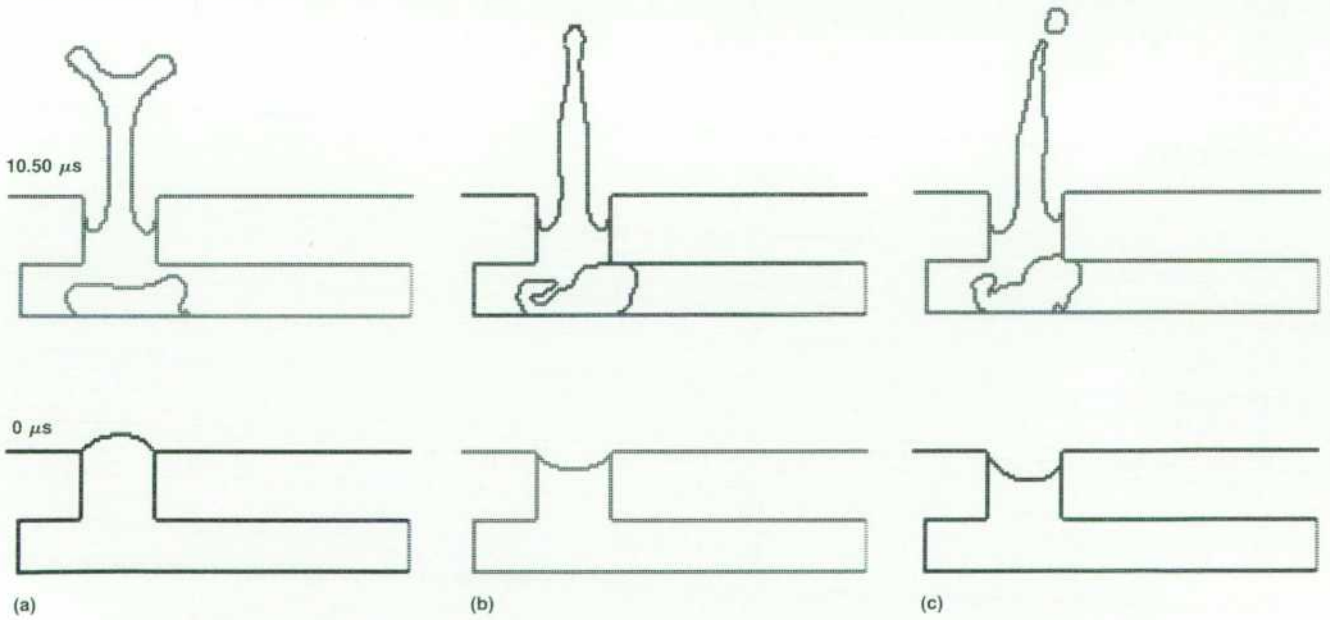



**Fig. 3.** *Effects of the meniscus configuration on the ejected drop. (a) A bulging meniscus adds to the volume of the drop. (b) A retracted meniscus reduces the volume of the drop. (c) Fluid acceleration during bubble growth causes emerging drops to pull off-axis or flare out as in (a).*

to record in a direct experiment.

### Ink Barrier

Channeling ink flow from the main ink feedslot to each firing chamber and preventing interference from adjacent firing chambers required the development of a new barrier material. It needs to act as a spacer between the orifice plate and the energized thin-film resistor, and be capable of producing straight firing chamber walls. The barrier material chosen to perform these functions is a polymer which is first photodefined and then cross-linked to make it impervious to water-based inks.

The barrier material (Fig. 8) is typically 26 micrometers thick, with its smallest lateral dimension measuring approximately 35 $\mu$m. A typical printhead consists of 25 opposed, individually spaced firing chamber pairs with a central main ink feedslot. Since the ink is heated by repeated firings of the resistor, it is imperative that the barrier material withstand ink and heat exposure for the life of the printhead. From a manufacturing standpoint, the material chosen for the barrier has to be cost-effective and manufacturable in high volume.

The polymer selected was not available in the thickness necessary for inkjet use. In addition, it was designed to image features on the order of 75 to 100 $\mu$m with a tolerance on the order of 50 to 75 $\mu$m. In the DeskJet printhead, the minimum feature opening is approximately 35 $\mu$m and has a much tighter tolerance. The technology had to be extended to produce these features on a consistent basis. Vendor-recommended process parameters were therefore investigated on a step-by-step basis, and a new production process was developed.

The polymer is applied to the silicon wafer by hot roll lamination. It is then exposed using an integrated circuit proximity aligner and chromed glass masks.

A conventional, scaled down, vertical developer is used to develop the laminated and exposed wafers. Additional developer and rinse solution filtration had to be added to eliminate the transfer of small particulate matter from wafer to wafer in the development step.

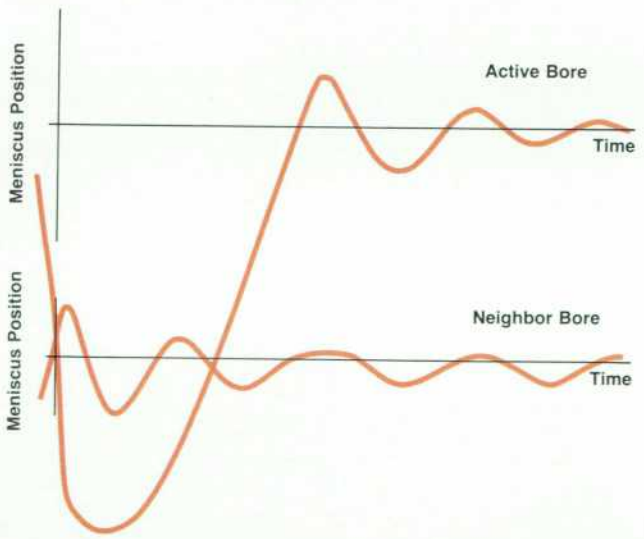



**Fig. 4.** *Fluid damping and cross talk for the initial ink channel length.*

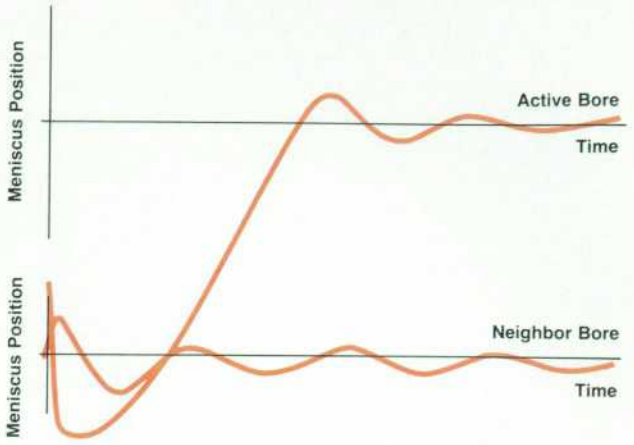



**Fig. 5.** *Fluid damping and cross talk for twice the initial ink channel length.*

### Orifice Plate

The orifice plate material (gold-plated nickel) is the same as that used in the 180-dpi PaintJet printhead. The higher-frequency operation and tight drop volume control needed for the DeskJet printer placed additional importance on orifice diameter tolerance. Particulate and chemical contamination controls were found to be keys to achieving the desired yields in manufacturing.

HP Laboratories had a major impact on the manufacturability of the orifice plate by conceiving a high-resolution electroplating process. This new process was transferred to the Inkjet Components Operation early in its development and completed there. The resultant orifice plates have a very well-controlled orifice diameter.

### Assembly Evolution

There are many differences between the DeskJet printhead and the ThinkJet printhead manufactured by the same HP Division. The DeskJet printhead has an interconnect that is perpendicular to the paper path (this removed sig-

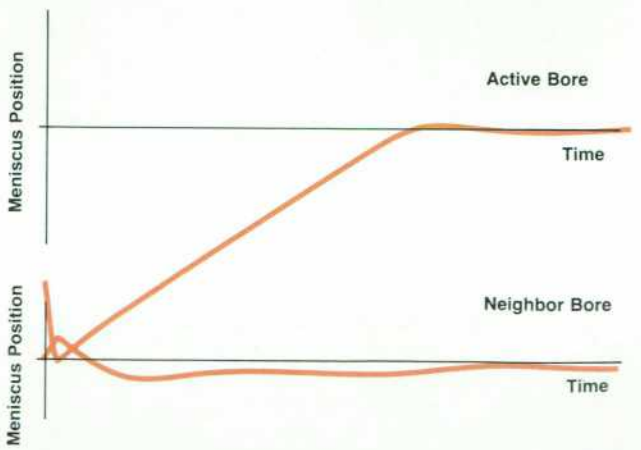



**Fig. 6.** *Decreasing the channel cross section provides good damping, a quick response, and minimal cross talk.*

nificant paper handling problems for the product design group). It has a larger ink containment volume for higher-resolution plain-paper printing. Its ink containment system uses a foam sponge, like the HP PaintJet printhead. High-precision alignment of the nozzle assembly to the plastic parts is required. The printhead installation method is different and more user-friendly.

This more-complex design made it necessary to reinvent almost every assembly process. The new printhead has 30% more mechanical parts and three times as many adhesives as the ThinkJet pen. A substantial increase in alignment precision was required as well to meet print quality objectives. The stacked-structure design (Fig. 9) is conducive to automated assembly.

### Precision Alignments

Three similar precision mechanical alignments are performed in the course of assembling the printhead. The orifice plate is aligned to the wafer, the singulated head is aligned to the plastic body, and the flexible interconnect circuit is aligned to the bonding pads.

In each of these alignment operations, parts handling varies significantly, but the scale and precision of the motions are very similar. For this reason, the fundamental framework of the three aligners is identical, with minor modifications to end effectors and software as required. By leveraging the design over the three different machines, engineering costs were significantly reduced.

Two generations of mechanical aligners have been designed to accomplish these three alignments. The first family of aligners provided equipment on a fast-track schedule, and at relatively low cost. These aligners were very labor and skill intensive, and were prone to a significant amount of downtime, inherent in a fast-track design. The current-generation aligners effectively eliminate the design deficiencies of the first generation. They are completely automatic and operate an order of magnitude faster.

### Interconnect Technology

To provide connections from the printer to the printhead resistors, conductors must be routed away from the nozzle/paper region and fanned out to a manageable 1.5-mm spacing. In the ThinkJet printhead, this objective is satisfied by simply extending the substrate far from the resistors. This did not provide a cost-effective solution for the DeskJet pen.

Many conventional interconnect technologies were investigated to perform this task, including gold and aluminum wire bonding and gang TAB (tape automated bonding). These processes were dismissed because of high welding temperatures, bumping of the bond pads, wire loop heights, or speed. The technology chosen was single-point tape automated bonding. TAB tape is a copper circuit printed on polyimide tape with cantilever beams extending beyond the tape perimeter. The beams are ultrasonically welded to pads on the die.

A standard gold-wire bonding machine was modified both in hardware and software to accommodate the process. The major bonding parameters of force, time, and ultrasonic

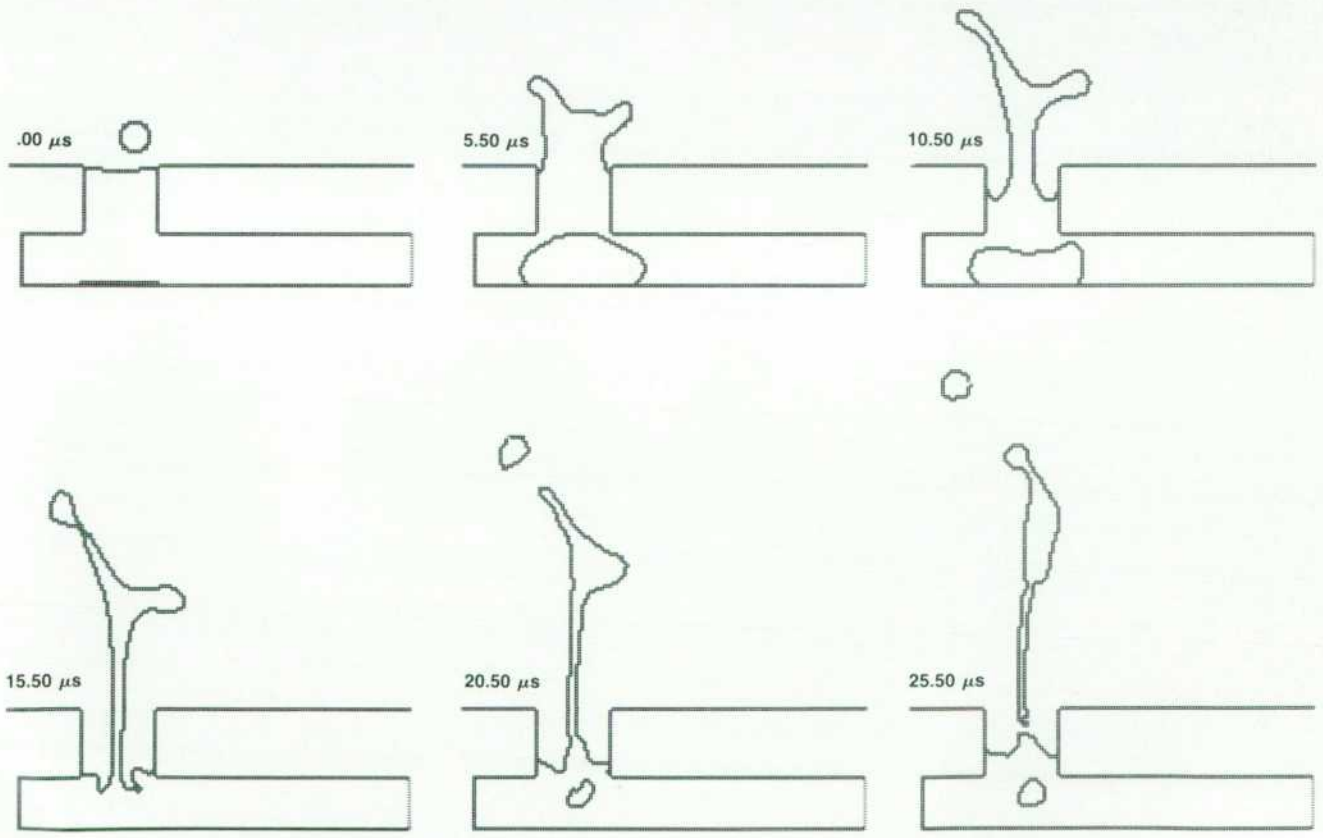



**Fig. 7.** *A simulation of a drop running into a residual droplet of spray left by a previously fired drop. This event would have been extremely difficult to record in a direct experiment.*


**Fig. 8.** *Photomicrograph of DeskJet printhead firing chambers and ink barrier.*

energy are program controlled to give the greatest operating window possible using a destructive beam pull test as the major process control parameter. Repeatability of flex circuit targeting required that the bonding machine have a pattern recognition system that aligns automatically without operator intervention (see article, page 91).

The bonding machines were further modified to accomodate a pallet transfer conveyor line where pens are automatically loaded, located, bonded, and then released to the next assembly process station. Production volumes were achieved by paralleling two bonders with a shuttle connection and having the host computer supply both bonders with pallets. This parallel arrangement also allows one bonder to be taken off-line for maintenance without stopping the entire production line.

Using the TAB circuit for the interconnect allows the silicon die size to be minimized, since the die only needs to be large enough to ensure sufficient gluing surface between the die and the plastic printhead body. The TAB method also cuts the die/interconnect cost by an order of magnitude.

After bonding, the beams are encapsulated to provide both a mechanical barrier and chemical resistance to the ink.

The printer interconnect design is quite similar to that of the ThinkJet printer, with a few subtle differences dictated by the modified geometry. It uses a bumped flex circuit in the printhead carriage.

### Tooling and Process Challenges

The experience obtained from the ThinkJet printhead development showed that this technology could not be automated until its many processing steps were well-understood. With this in mind, assembly methods that would allow automation to be added incrementally were sought. An asynchronous line was selected because it provides tremendous flexibility in tooling and process development. The line was erected a full year before production release. Prototype tools were installed on the line to evaluate assembly strategy, process flow, referencing techniques, and tooling methodology.

As more refined tools were developed, they were installed on the line and debugged before removing the old tooling. This provided a steady stream of prototype printheads to the development team. In the latter stages of development, several thousand prototype printheads per month were consumed in printhead and product qualifications. This volume of printhead production was instrumental in rapid training of the production personnel on a multi-shift operation. It also provided a critical volume of printheads necessary for debugging the assembly process and tooling.

Using a pallet transfer line concept, the number of mech-

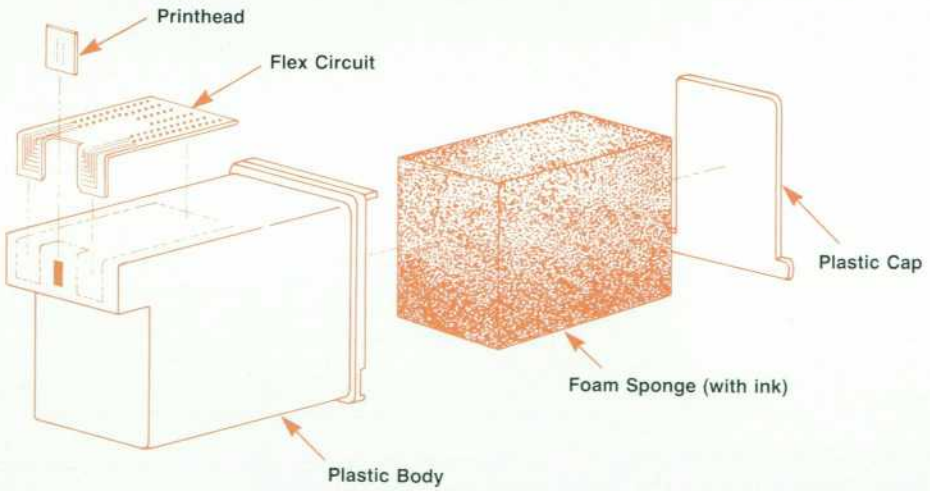



**Fig. 9.** *Exploded view of the DeskJet print cartridge.*

anisms required for the tools is minimized. Printheads are handled on a fixtured pallet and presented to the tool or operator by means of off-the-shelf lifting and locating mechanisms. Workstation frames are made out of standard aluminum extrusions that bolt together and can be easily reconfigured to meet changes. This concept also allows the relocation of tooling to meet changes in the process flow with no impact to the product schedule. During the first six months of production, eight assembly tools were relocated and 14 newly upgraded ones were added. Initially, a majority of the workstations required a full-time operator. Many of these have now been replaced with upgraded and automated tools. The goal is to be fully automated before the end of the first year of volume production.

Because the printhead design was not a simple evolution of the ThinkJet pen design, a significant fraction of the total printhead development effort was spent in the development of assembly tooling and processes for the DeskJet printhead component assembly line.

### Reliability Testing

In the development phase, printheads were continuously tested for adherence to specifications. At the beginning of the project, overall goals for reliability were set. At each transition point from one project phase to the next, the confidence level for meeting these goals was increased.

The DeskJet printhead is designed to be consumable, so the concept of annualized failure rate does not apply to it. Therefore, the reliability goals specify the percentage of printheads that pose no start-up problems to the customer and have no uncorrectable print quality failures before the ink has been used up. Goals also exist for the maximum number of correctable print quality defects over the life of a printhead and for the number of customer interventions (primes) required to correct any defect.

To evaluate start-up problems, testing was performed to the standard HP environmental specifications. Many of the specifications are not applicable to the printhead alone, so printheads were tested as part of the printer system testing. These real-life tests were completed early enough in the printhead's development to allow time for corrections to be made. Testing for print quality degradation over the temperature and humidity range posed an interesting problem. The paper could not tolerate the environmental extremes specified for the printhead and printer. Sheets of paper would stick together, causing misfeeds. Special procedures were developed to verify printhead operation at the extreme conditions.

A major focus of the testing went into verifying the reliability of the printhead over its life after it was inserted into a printer. A test that simulates actual customer use was developed. A sample page of mixed text and graphics was used as the standard page. A customer print job was defined as 15 pages of this pattern and the printheads were run out of ink printing jobs. The operators running the tests intervened as customers would to correct any print quality defects that occurred.

Initial life testing on modified plotters began early in the printhead's history when printers were not available. While these tests pointed out major printhead problems, the subtle problems that customers were likely to see were not apparent until DeskJet system printing was performed. Once printers were available, printheads for these life tests were obtained from actual production lots and tested on a daily basis.

Test results were at times difficult to evaluate. Print quality defects and failures were not always attributable to the printhead or printer alone, but rather to the interaction between them. An extensive failure analysis system was developed to trace and sort out the underlying problems.

### Acknowledgments

It is important to note that the authors listed represent a small portion of the people who worked on the DeskJet printhead. The entire team was extremely dedicated, spending many long days in developing tools and processes and in understanding the operation of the printhead. There was a high degree of cooperation between all of the divisions involved. HP Laboratories provided process research, processes were leveraged from and jointly developed with the San Diego Division, and significant and detailed design trade-offs were jointly developed with the Vancouver Division. Two people in particular need to be mentioned: Frank Cloutier, who was the R&D manager from the beginning of the ThinkJet printhead development through most of the DeskJet printhead development, and Cheryl Katen, who was the DeskJet printhead program manager. Their support and direction were fundamental to the success of this program.

The authors would like to thank Bill Knight, Judy Layman, Marzio Leban, Jim Pollacek, and Shell Whittington for their contributions to this article.

### References

1. *Hewlett-Packard Journal*, Vol. 36, no. 5, May 1985, entire issue.
2. *Hewlett-Packard Journal*, Vol. 39, no. 4, August 1988, pp. 6-56.

# Integrating the Printhead into the HP DeskJet Printer

*The printhead support systems provide signals to energize the ink-firing resistors, electrical connections to the pen, a carriage to hold and move the pen, and elements to protect and maintain the pen.*

**by J. Paul Harmon and John A. Widder**

THE HP DESKJET thermal inkjet printhead requires a higher level of support from the printer than earlier generations. There are more nozzles to drive, they have to be driven faster, and more electrical connections have to be made to the head. Smaller nozzles with fast-drying, plain-paper ink require protection to prevent the head from drying out and mechanisms to recover nozzles that have clogged. And like earlier disposable thermal inkjet printheads, a carriage is needed to move the printhead across the paper.

Several overall design constraints guided the design of the printhead support systems. Each element had to meet the longevity goals set for it, perform its task, be robotically assemblable, and be low in cost. To meet the last two goals our project attempted as much as possible to design the carriage mechanism for top-down assembly and minimum part count.

### Head Drive Electronics

The printer circuitry to interface with the printhead is located on a printed circuit board mounted on the carriage mechanism. Two custom ICs on the board drive the printhead, with each IC driving half (Fig. 1). Each driver IC contains two 4-to-13-line decoders and has four address inputs, which are shared by both decoders in the IC, and two enable inputs, one for each decoder. The four address inputs indicate which of the thirteen outputs of each decoder will be turned on if its enable input is selected.

The decoder architecture only allows four dots to be fired at a time. It is necessary to minimize variations in the energy dissipated in the printhead thin-film resistors to maintain good print quality, so we did not want varying voltage drops across the commons between the power supply and the printhead when firing resistors. This requires that there be a separate common for each resistor when several are energized simultaneously. Fifty commons would have been impractical from a cabling standpoint, and one would have been been impractical from a timing standpoint, so as a good compromise, four commons are provided. Since only four nozzles can be fired at a time, decoders are used to minimize the number of connections between the main board and the head driver board.

It takes approximately 120 microseconds to fire all 50 nozzles in letter-quality mode. The distance moved during this time could produce a noticeable skew in the column, so the nozzle positions in the head are skewed to compensate. The cycle time for dot firings is cut in half when printing in draft mode (at twice the normal speed) so that the skew compensation is correct at both speeds. The nozzles are fired in reverse order when printing backwards.

There are three other functions on the head driver board. The first of these is the continuity test circuit. This circuit monitors the voltage drop across resistors in series with the printhead commons to determine if current flows when a driver is activated. A comparator monitors the voltage

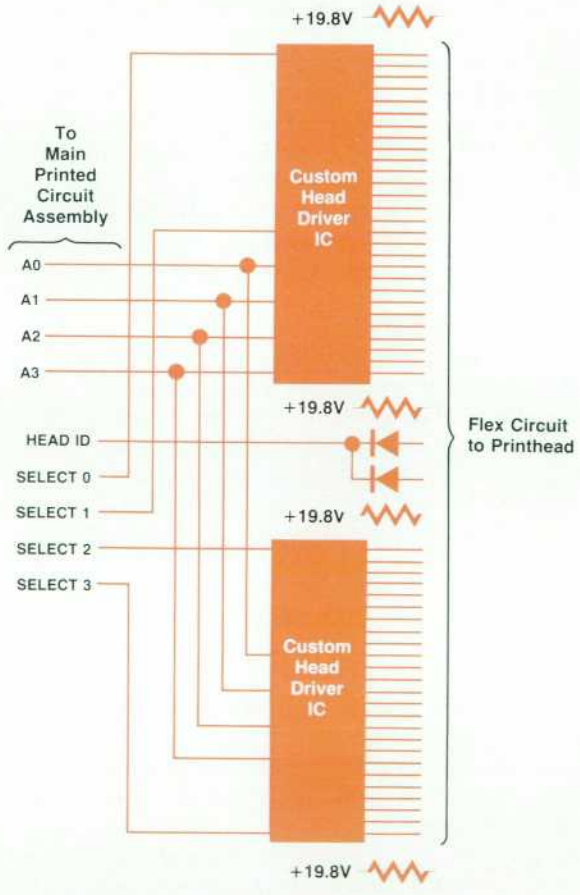



**Fig. 1.** *Two custom ICs on a carriage-mounted printed circuit board drive the DeskJet printhead.*

drop across each series resistor, and its output goes low if the current through the printhead is high enough. The comparator's output is monitored by the microprocessor on the main board during self-test, and it will print out the numbers of any nozzles that have bad continuity. This allows the user to determine whether a nozzle failure is caused by a resistor failure in the printhead.

In addition to the fifty drive lines and four commons going to the printhead, there are two sense lines that are used for encoding a printhead ID. These lines are mask-programmable on the printhead, and they will be used in the future to sense different types of printheads that might be installed in a printer. These ID sense lines normally connect to two of the printhead drive lines so that when the drive line is pulled low, the sense line connected to it will also go low. If the sense line is open, however, it will not go low when the drive line is activated. In this way, the printer can detect whether the ID sense lines are open. The two ID sense lines are wire-ORed together and a single line is fed back to the microprocessor on the main board.

An optointerrupter mounted on the head driver board detects the presence of paper in the paper path. A vane with a window normally rests in a position where the infrared light from an LED in the optointerrupter shines through the window in the vane and is detected by a phototransistor. When paper is in the paper path, the vane moves up so that the light from the LED is blocked and the phototransistor is turned off. The output of the phototransistor is fed back to the main board, where it is detected by the microprocessor.

### Printhead Energy Window Budget

The amount of energy dissipated in the thin-film resistors of the printhead is critical to both print quality and printhead reliability. If too little energy is delivered to the printhead, the print quality will be poor. On the other hand, if too much energy is delivered to the printhead, the printhead life will be reduced. There is a narrow window where good print quality is achieved without adversely impacting the printhead life.

There are several factors that can impact the amount of energy dissipated in the thin-film resistors when they are energized. The amount of energy dissipated in the resistors is:

$$E = I^2 R_{tf} t_{fire}, \quad (1)$$

where

$$I = \frac{(V_s - V_{sat})}{(R_{tf} + R_{trace} + R_{source} + R_{cap})} \quad (2)$$

$R_{tf}$ is the printhead thin-film resistance, $V_s$ is the head supply voltage, $V_{sat}$ is the driver saturation voltage, $R_{trace}$ is the stray resistance on the head driver board, flex circuit, and printhead, $R_{source}$ is the resistors in series with the printhead commons, $R_{cap}$ is the effective series resistance of the head power supply output capacitor, and $t_{fire}$ is the time during which current flows through the printhead. Variations in any of these parameters will cause the energy dissipated in the printhead to vary. To complicate matters further, the energy required to produce good print quality (the turn-on energy) varies from printhead to printhead.

In addition to specifying tight tolerances on components, two specific measures have been taken to minimize parameter variations and their effects. The first step is incorporation of temperature compensation into the head driver ICs. The ICs are designed so that the temperature coefficient of the output saturation voltage tracks the temperature coefficient of the output storage time, so that as the saturation voltage increases, decreasing the dissipation in the drivers, the storage time also increases, increasing the dissipation. The temperature variations of these two parameters cancel each other, resulting in zero variation in energy with driver temperature.

The second step taken to minimize energy variation is the use of resistors in series with the printhead commons. As mentioned previously, source resistors are placed in series with the printhead commons to sense current flowing through the printhead. The principal reason for having these resistors, however, is to minimize the variation of applied energy with changes in printhead resistance. The change in energy dissipated in the printhead resistors when their resistance changes is at a minimum when the sum of all other resistances in the circuit is equal to the resistance of the printhead resistors. The sum of the other resistances is not quite equal to the printhead resistors, but having the source resistors helps reduce the change in printhead energy when the printhead resistance changes.

### Monte Carlo Model

Normally, a conservative designer will use worst-case analysis to ensure that a design will perform satisfactorily under all conditions. However, selecting head drive circuit parameters to ensure that all printheads receive the minimum required energy under worst-case conditions would have resulted in a nominal energy high enough to shorten the life of the printhead. Since the variables affecting the energy delivered to the printhead are independent of each other, it was recognized that the probability of worst-case conditions occurring is very small. We decided

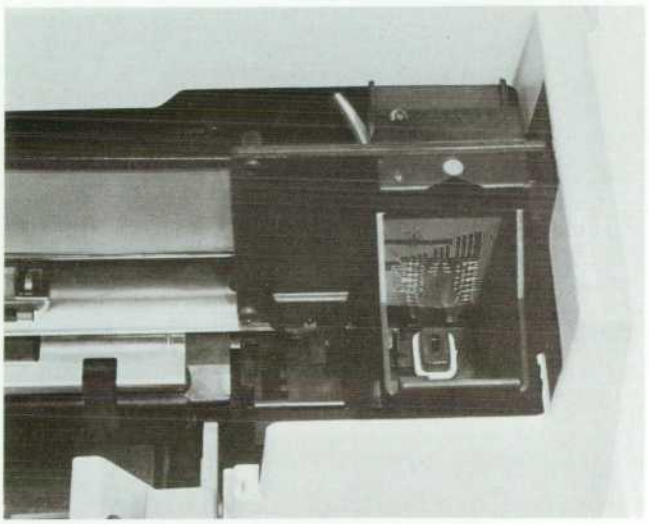


**Fig. 2.** *For interconnecting the printer and the printhead, the DeskJet printer uses a bumped flex circuit on the carriage.*

to use Monte Carlo analysis to determine the nominal circuit values. Monte Carlo analysis uses a statistical approach to determine the distribution of a variable based on the distributions of its constituent parts.

The first step in Monte Carlo analysis is the creation of a model of the circuit. In our case, this model is the equation for printhead energy given in equations 1 and 2, along with equations based on the various printhead process parameters that affect turn-on energy. The second step in Monte Carlo analysis is determining the distributions of the variables in the model. In our analysis, a normal (Gaussian) distribution was assumed for all but two of the variables. These two variables are printhead resistance and printhead turn-on energy, which we test during the head assembly process. If a printhead is outside of preset limits on either of these parameters it is discarded, resulting in a normal distribution with the tails of the distribution cut off.

Once the model has been constructed and the distributions of the model variables are known, simulations can be run to determine the distribution of the variable of interest. We were interested in monitoring the distribution of the ratio of applied energy to turn-on energy, which is a key indicator for both print quality and reliability. The simulation involves "building" a printer and printhead by picking parameter values at random using their distributions, calculating the applied energy and turn-on energy using those values, and tracking the distribution of the ratio of these two energies. The two truncated distributions are easily simulated by checking the parameter values after they have been selected and before they are used in the model. If a parameter's value falls outside the allowable limits, that value is discarded and a new value is calculated. This process allowed us to set the nominal energy high enough that virtually all printheads and printers will work together, while keeping the nominal energy low enough to avoid impacting printhead life.

### Printhead Interconnect

For the HP ThinkJet printer, HP developed a bumped flex interconnect technology that has proved very successful in practice. After examining other alternatives for the DeskJet printer, the design team settled on the same basic design expanded from 12 connections to 56. In the ThinkJet printer, the connections are made to gold-plated pads on the glass chip that contains the resistors that fire the drops of ink at the paper. For the DeskJet printer, a silicon chip with 56 of these connections would have been too large to be cost-effective. In addition, the large chip would have placed unacceptable constraints on paper path design, because it has to lie almost in a plane with the nozzles, which have to be very close to the paper.

After careful analysis, a design employing gold pads on a tape automated bonding (TAB) flex circuit was executed (see article, page 55). The TAB circuit pads attach to the chip pads and the TAB traces fan out to the bottom of the printhead, where they are contacted by the bumped flex

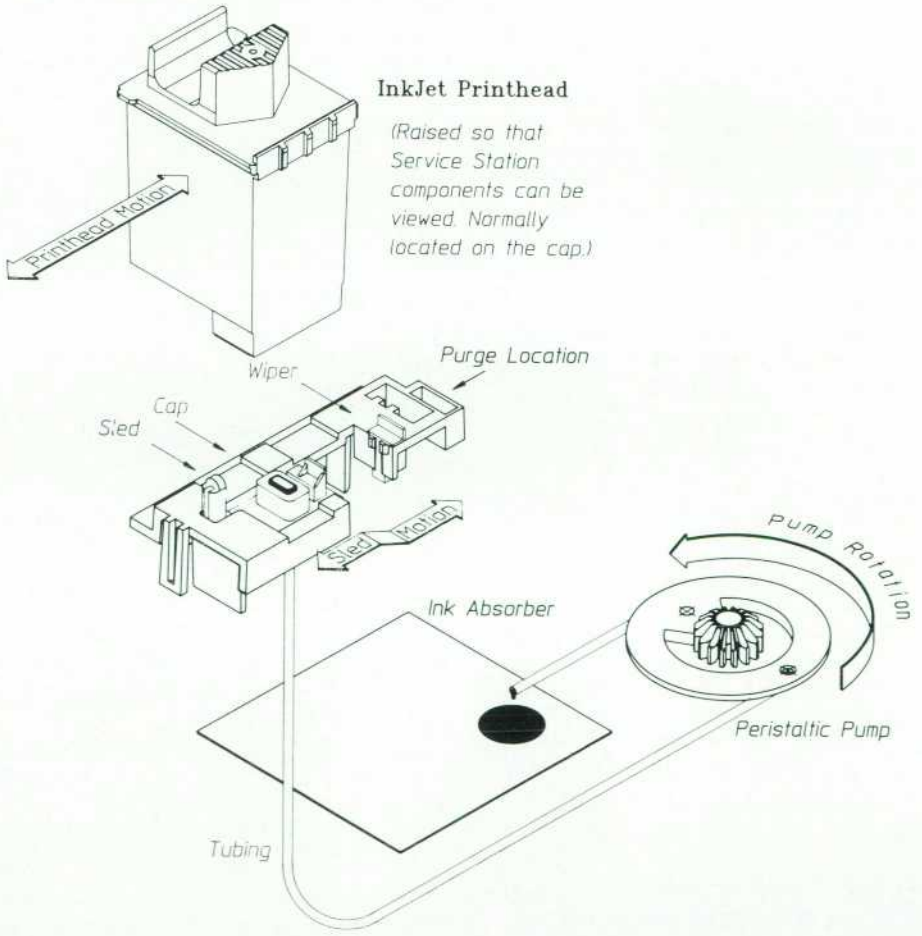



**Fig. 3.** *DeskJet printer service station assembly.*

circuit on the carriage, which is shown in Fig. 2. Using this technology brings the pen cost to an acceptable level and frees paper path design considerably, because the interconnections, though actually larger in area, can be wrapped around to the bottom of the pen where they are out of the way.

### Pen Maintenance

One of the major challenges the project team faced was making trade-offs associated with printhead maintenance. Since the head was being developed concurrently with the printer there were many unknowns. Constant contact with HP's Inkjet Components Operation (where the printhead designers reside) was maintained and proved invaluable to the creation of a successful product.

The product had five main objectives for printhead maintenance:

- Prevent the nozzles from drying out
- Keep paper dust off the nozzles while not printing
- Wipe off any paper dust that might accumulate during printing
- Provide a location for purging viscous plugs from the nozzles before beginning print
- Provide a method for the user to clear a dried-out or plugged head without taking the pen out of the printer.

The service station that accomplishes these objectives is shown in Figs. 3 and 4.

A rubber cap that seals to the orifice plate is provided to protect the nozzles. This cap prevents the nozzles from drying out and covers the nozzles while they are inactive, keeping them from being covered with paper dust. The cap is brought into contact with the printhead by a combination of carriage motion and ramps molded into the dc servo motor mount. The ramps raise the cap as the carriage goes to its parked position.

An elastomer tube attached to the underside of the cap provides a long diffusion path while venting the cap. The diffusion path maintains high humidity in the cap. The venting prevents the cap from acting as a bellows and blowing bubbles into the nozzles as the head comes to rest on the cap. The tube also serves as a primary component in the peristaltic pump used to service the printhead. If the nozzles are dried out (from sitting in a desk drawer uncapped, for instance) or a stubborn piece of paper dust is clogging nozzles, the user can actuate this pump to draw about 0.05 ml of ink from the head. This flushes the nozzles so the head can be returned to service.

The mechanism that performs the pump function with the tube is located near the end of the tube. The DeskJet pump employs only one roller rather than the traditional three that are usually used for peristalsis. This allows the tube to be vented to atmosphere when the pump is not operating without the need of a separate vent in the system. Ink from the pump is dumped into an absorber located in the bottom half of the printer, from which it evaporates.

A soft rubber wiper is built into the printer as a means of clearing from the nozzles any paper dust that might accumulate during printing. The paper dust is scraped into pockets on either side of the printhead. Thus, every time the printhead is changed, the accumulated paper dust is thrown out with the old printhead.

Since printhead service is an ancillary function of the printer, low cost was an absolute must. This was achieved by implementing a minimal-part-count passive design. There are no motors dedicated to printhead maintenance. The pump is powered through a carriage-actuated transmission using the paper drive motor, and the cap is automatically engaged when the carriage comes home after printing. As much as possible, elements required for these functions are integrated with structural elements of the chassis. Thus, a peristaltic pump is molded into the main frame and the capping function is achieved with ramps molded into the dc servo motor mount.

The design team was aided in making this integration by HP ME Series 10,[1] a computer-aided design tool from HP, which was effectively used to study interactions at an early phase of the project. The DeskJet mechanism team was one of the first sites in the corporation to begin using ME Series 10.[2] (The plastic part on the cover of the May 1987 Hewlett-Packard Journal is the main chassis for the DeskJet printer.) The geometries of many of the parts in the product contain inputs from more than one engineer. Managing ongoing development of parts when so many are

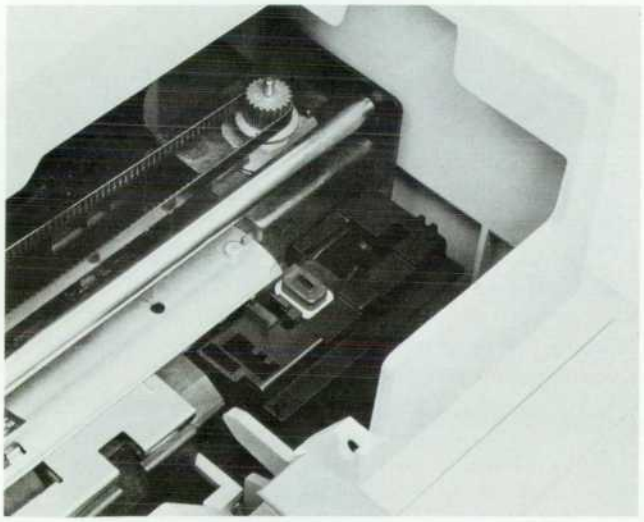


**Fig. 4.** *Photograph of the service station.*

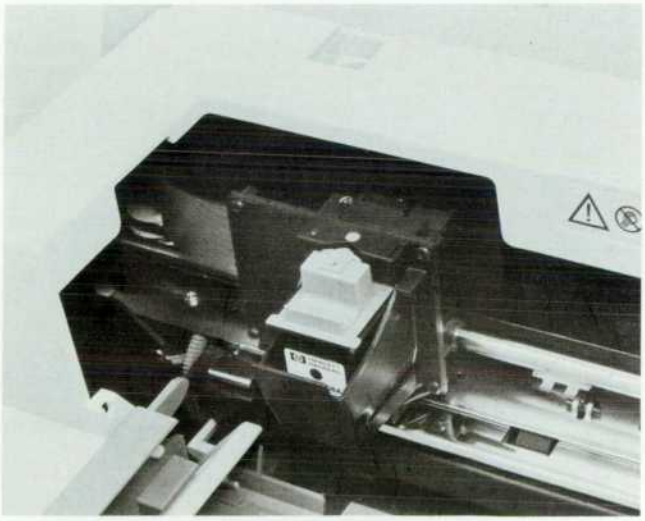


**Fig. 5.** *DeskJet printer carriage with printhead.*

multifunctional and have several designers is made easier when ME Series 10 is available.

### Carriage

The DeskJet printhead provides 300-dpi resolution. Taking advantage of this resolution requires a mechanism capable of locating the printhead accurately. Keeping track of tolerances was unusually important in the design of the printhead carriage, which is shown in Fig. 5.

The interconnect described above is contained in the carriage. Good connection between the printhead and the head drive circuit requires between three and five pounds of force. This load level is sufficient to deflect thin plastic parts more than can be tolerated, and here a problem was realized. The top-down manufacturability requirement allows little space for carriage alignment points. Carriage acceleration goals require relatively thin walls in the carriage structure for low mass. Thus, little plastic can be used for interconnect load support points. Since these points are also the head alignment points, stiffness is vital in whatever structure is used. To convince ourselves that it was indeed possible to satisfy these requirements, a linear static finite element model was created using HP-FE, and the carriage design is based on the results of that computer model. Several members of the team participated in the design of the carriage during various phases of its development and ME Series 10 was extremely useful in handing off the geometry from engineer to engineer with a minimum of documentation.

Carriage bearing life was also a concern. The DeskJet carriage employs molded-in-plastic bushings riding on a polished steel rail. This is both a cost-saving measure and a way to minimize carriage-to-shaft tolerances. The design is laid out to prevent undue bearing loads caused by cocking. The success of this design was proved early in the development cycle by an accelerated test bed, which ran ten carriages to over twice the projected printer life.

Helping the customer intuitively understand how to install a printhead was another design goal for the carriage. Several days of brainstorming and many failed designs led to the current implementation, which in one test was rated by users as being easier to use than the power switch. A funnel-shaped chute to drop the pen into, mating geometry between the printer and the printhead, and color coding all play a role.


### Acknowledgments

The head driver IC was designed by Pat Byrne and Dan Nguyen at the HP Santa Clara Technical Center, and Chuck Jarvie was responsible for procurement. Mark Lund developed the continuity test circuitry. Henry Flournoy and Bill Royce at the HP San Diego Division helped specify the head driver IC characteristics and developed the idea of using source resistors. Early carriage conceptualization was done by Dave Pinkernell. The DeskJet mechanism team is indebted to Brian King for his work on carriage development.


### References


1. W. Kurz, et al, "State-of-the-Art CAD Workstations for Mechanical Design," *Hewlett-Packard Journal*, Vol. 38, no 5, May 1987, pp. 4-15.
2. P. Harmon, "Alpha Site Evaluation of ME Series 10," *ibid*, pp. 30-33.

# DeskJet Printer Chassis and Mechanism Design

*One mechanism moves the carriage while another uses a single motor to pick, feed, and eject paper and prime the pen. The polycarbonate chassis supports everything.*

**by Larry A. Jackson, Kieran B. Kelly, David W. Pinkernell, Steve O. Rasmussen, and John A. Widder**

THE CHASSIS OF THE DESKJET PRINTER is an injection molded plastic part that supports the mechanical and electrical systems (Fig. 1). Besides meeting its own objectives, the chassis design helps accomplish some of the overall objectives for the printer. Part count is minimized by the large amount of functionality built into this single part, and ease of assembly of the other parts to the chassis is a feature of the design. Both factors help reduce the cost of the printer.

Two important criteria for the chassis design were material selection and tooling. The material needs to be very good structurally, have good dimensional stability, and help dissipate electrostatic charge created by the paper motion. It also needs to be a good bearing material and a good snap material. Requirements for the tooling are that it be simple, fast, and durable (i.e., good for 200,000 parts).

The chassis is designed as one large part that takes the place of many parts and functions as the main structure for the printer mechanism. The chassis also integrates many of the functions of the printer.

The following is a list of the part attachment details of the chassis:

- Pressure plate spring locators
- Pressure plate location and bearing surface
- Adjustable wall and lever location and bearing surface
- Pinch roller location and bearing surface
- Drive roller location and bearing surface
- Transmission location and bearing surface
- Gear train location and bearing surface
- Paper motor screw holes
- Head driver cable and ferrite core location and holding snap for core
- Belt tensioner assembly location
- Head driver board location and retaining snap
- Prime pump bottom and location for the other parts of the pump
- Right wall retaining snaps
- Motor cable routing details

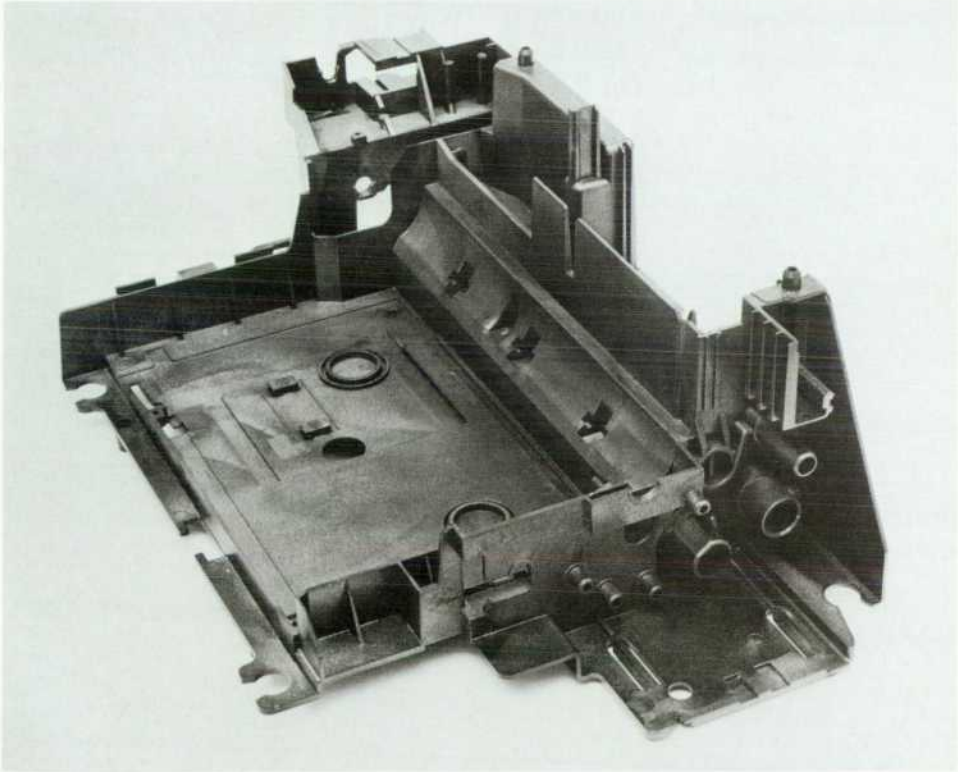


**Fig. 1.** *The DeskJet printer chassis is a complex injection molded part that takes the place of many parts and serves as the main structure for the printer mechanism.*

- Flex cable retaining detail
- Preloader assembly and location detail
- ESD clip and carriage rod retainer location detail
- Carriage rod location detail
- Carriage guide and paper guide location and screw hole
- Pump tube location.

The following are functional details of the chassis:

- Paper path guide surface
- Part of the input paper tray
- Envelope loading guide and surface
- Details for six grommets that mount the mechanism to the case parts
- Right corner separator used for picking a sheet of paper
- Details to reference the mechanism to the assembly pallet.

As the assembly of the product begins, a pin for the paper drive motor gear cluster is pressed in, pump tubing is put in place, and the head driver cable with its ferrite core is snapped into place. The chassis is placed on a pallet which locates on details within the chassis. The chassis then travels down the assembly line on a belt and all the other parts are assembled to it.

### Material and Tooling

The material selected for the chassis is a polycarbonate with a 15% milled carbon fiber filler. This material is dimensionally stable and structurally strong enough to hold up well in the operating environment. The carbon filler helps conduct electrostatic charges from other parts and the chassis itself to ground. The material also works very well for the snap details that retain various parts and makes assembly easy. With the proper selection of mating part materials, the chassis material wears very well.

Even though the chassis is complex, the injection molding tool is fairly simple, containing only two slides and a few shutoffs. The tool is made from P20 steel, which is about as easy to machine as aluminum, but much more durable. Because the part is fairly big and complex we opted to build two gating systems into the tool, so it can be filled by a single sprue or four pin gates. As the initial parts were molded we used the single gate, but with experience, better parts were formed using the four-pin-gate system.

During the design and development of the chassis, four or five engineers were able to work on the part at the same time. There were at least two reasons for having more than one engineer involved. First, the engineers were working on parts that would eventually mate to different parts of the chassis, so it made sense for them also to develop the chassis details to fit their mating parts. Second, since the chassis is complex (fourteen E-size sheets are required to describe it), it was the pacing part in the schedule. To shorten the schedule, other engineers helped do some of the cross sections and dimensioning so the chassis could be tooled sooner. This multiple-engineer design was made possible by the HP ME Series 10 CAD system.[1]

## Paper Handling System

In the design of the DeskJet paper handling system, the primary goals were to pick a single sheet of paper from the input stack, present it to the print cartridge with the precision demanded by the 300-dot-per-inch resolution specification, and eject the sheet to an output bin. These goals were to be accomplished with a reliable, compact, and easy-to-use mechanism, and for minimum cost.

The worldwide market for this printer requires that the paper handling system work equally well with papers of different sizes (U.S.A. A-size and legal-size and metric A4), weights (U.S.A. basis weights of 16 to 24 pounds), compositions (photocopier through fine cotton bond), and textures (smooth through rough). In addition, the system must work equally well in a variety of environmental conditions,

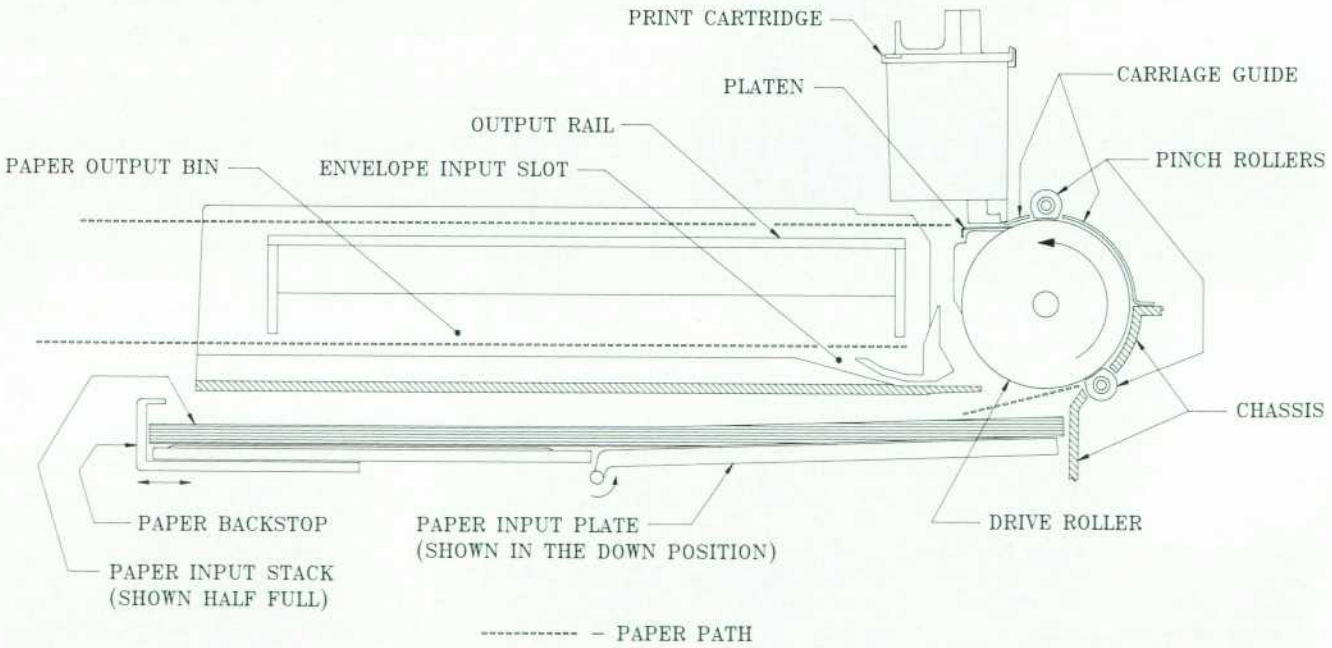



**Fig. 2.** *DeskJet printer paper handling mechanism.*

with temperatures ranging from 10°C thru 40°C and relative humidities ranging from 10% to 70%. It should be no surprise that the engineering properties of paper change significantly across those ranges. It was also a goal to allow the user to hand-feed envelopes to satisfy occasional needs.

A major constraint on the paper handling system comes from the use of inkjet technology. The ink is sprayed onto the paper wet, and requires a short drying time before it can be handled. As a result, the printed surface cannot be touched by either the printer mechanism or by another sheet of paper until the ink is dry. If it is touched too soon, the ink may smear or blot.

The paper handling system has three major functions: the picking of a single sheet of paper from the input stack, the movement of that sheet past the print cartridge, and the ejection of that sheet into an output bin. Fig. 2 shows the elements of the system.

### Paper Pick

The goal that the printer be easy to use requires that paper be easy to load and that adjustments for the various sizes be minimal and simple. Paper is loaded into the front of the printer by selecting the appropriate width (typically once in the lifetime of the printer), sliding the backstop of the paper input tray out, inserting up to 12.7 mm (approximately 100 sheets) of paper into the tray, and sliding the backstop forward until it is against the back of the stack of paper. There is a small tab on the inside of the backstop, 12.7 mm off the floor of the paper input tray, which prevents the backstop from being fully seated if more than 12.7 mm of paper is inserted. This gives instant feedback to the user if there is too much paper. The adjustment for paper width is accomplished by moving a two-position (U.S.A./metric) front-panel lever to the appropriate position. The lever is attached to a sliding wall on the left side of the paper input slot. The sliding wall, a fixed wall on the right side, and a plate on the bottom form the paper input tray. The adjustment for paper length is accomplished by sliding the paper input tray backstop forward until it just touches the stack of paper in the input tray. To accommodate normal cutting tolerances (±0.7 mm) and environmentally induced tolerances (±2.0 mm), the left wall of the paper input tray is equipped with a spring-loaded guide, which takes up this tolerance and encourages the stack of paper to press against the fixed right wall. The right wall is used as an edge reference between the paper and the printer. The printer logic circuits expect the edge of the paper to be at that position and command the print cartridge to begin printing at a point just past it (allowing for an appropriate margin).

The plate that forms the bottom of the paper input tray is pivoted and spring-loaded such that, if unrestrained, the edge supporting the top of the paper (the edge inserted into the printer first) would pivot up. The plate is held down by a cam on the right side which rotates on demand from the transmission (described later). As the cam rotates, the plate rotates about its pivot and the top edge of the input stack of paper rises. The top sheet is simultaneously forced into the drive roller and into two corner separators, one at each top corner of the sheet. The drive roller is a set of three medium-soft rubber rollers (one each near the right and left edges of the paper and one centered) on a single shaft, which is rotating whenever a sheet of paper is being picked. When the top sheet of paper comes in contact with the rotating rubber drive roller, it is pulled forward and wraps around the drive roller. To ensure that only one sheet is picked, the corners of the sheet are forced to buckle over corner separators. This buckling force acts as a restraining force on the sheets of paper in the stack. Because the coefficient of friction between the drive roller and the top sheet is greater than that between the following sheets, and because the normal force is the same, the drive roller can impose enough force to overcome the buckling on the top sheet only. Because of the difficulty in modeling the engineering parameters of various papers in various environmental conditions, the geometry of these corner separators (essentially small triangles of plastic overlapping the corners of the sheet) was optimized by building a prototype printer with screwdriver-adjustable corner separators. This model was tested with the various papers and under the various environmental conditions until the best geometry was obtained. While the geometry selected is not radically different from the original design, this testing and adjusting improved the range of reliable picking considerably. Finally, after the sheet is picked, the cam continues to rotate and forces the plate, which is supporting the input stack of paper, down and away from the drive roller.

Envelopes are loaded manually by the user, bypassing the automatic pick system. The user simply presses the **Load Envelope** buttons on the keypad. This starts the drive roller rotating. The user then inserts an envelope into a convenient slot until it contacts the drive roller. The envelope follows the same path as an automatically picked sheet of paper.

### Paper Motion

The key component of the paper motion system is the drive roller. Paper wraps around this roller as soon as it is picked and remains in contact with it until it is ejected to the output bin. The diameter of the drive roller is ground to the precision required to maintain the 300-dot-per-inch print resolution, and it is important that the paper remain in intimate contact with it. Two sets of deformable pinch rollers, one located after the pick zone and one located near the print zone, and a set of leaf springs located immediately before the print zone, ensure this intimate contact. The goal of minimum cost motivated this single-roller design, but the conflicting requirements for paper pick and precise paper feed complicated the design process. Pick rollers are generally soft, which makes the diameter difficult to control, while feed rollers are generally hard and much easier to control. This dilemma was resolved by using a large-diameter roller (the absolute diametrical tolerance is relatively constant over a wide range of diameters, so the percentage tolerance is reduced as the diameter is increased) and by careful selection of the roller material and hardness.

As the paper feeds around the drive roller and approaches the print zone, it is pushed into a sheet-metal platen and forced to peel off the drive roller and through a slot formed by the platen and a parallel piece of sheet metal, the carriage guide. The bow caused by this change

in direction stiffens the paper considerably and forces it to lie flat against the platen. The platen is spring-loaded against the carriage guide so that the printable surface of any thickness of paper will be touching the bottom surface of the carriage guide and will be parallel to the platen. This is very important with inkjet printing technology, because the distance from the print cartridge to the paper (nominally 1.0 mm) must be carefully controlled over the entire print zone (8 inches wide by 1/6 inch deep). Using the carriage guide as a reference, the carriage is able to reference the print cartridge accurately to the paper. The printable surface of the paper is not touched by the printer after entering the print zone, and the quality of the document is protected while the ink dries.

While the printer logic circuits can assume that one edge of the sheet of paper is referenced to the right wall of the paper input tray, no similar assumption can be made about the top and bottom edges of the sheet. The out-of-paper switch is located so it can detect both the top and bottom of the sheet and signal the logic circuits. The switch is located between the pick zone and the print zone and consists of a lever which trips an optical switch when paper is present. Noting when paper is first detected, and when it is subsequently absent, and knowing the distance from the out-of-paper switch to the print zone, the logic circuits can calculate the location of both edges of the sheet. Furthermore, because paper is detected well before it enters the print zone, the unprintable region at the top of the sheet is limited only by minimal tolerances. The unprintable region at the bottom of the sheet is somewhat more limited by the distance between the leaf springs and the print zone because the paper cannot be driven precisely after it passes these springs, but this distance is small.

### Paper Eject

After the sheet of paper is printed, the ink is wet for a short while and must dry undisturbed, or it will smear or blot. Rather than wait for each sheet to dry before starting the next, the mechanism includes a one-sheet buffer which allows the first sheet to dry undisturbed while the next is printed. This is accomplished by sliding the sheet onto a set of output rails as it is being printed. When the sheet is complete, the transmission selects the eject cycle, which rotates the platen down, releasing the paper from between the carriage guide and the platen. As the platen reaches the bottom of its rotation, two tabs press against mating

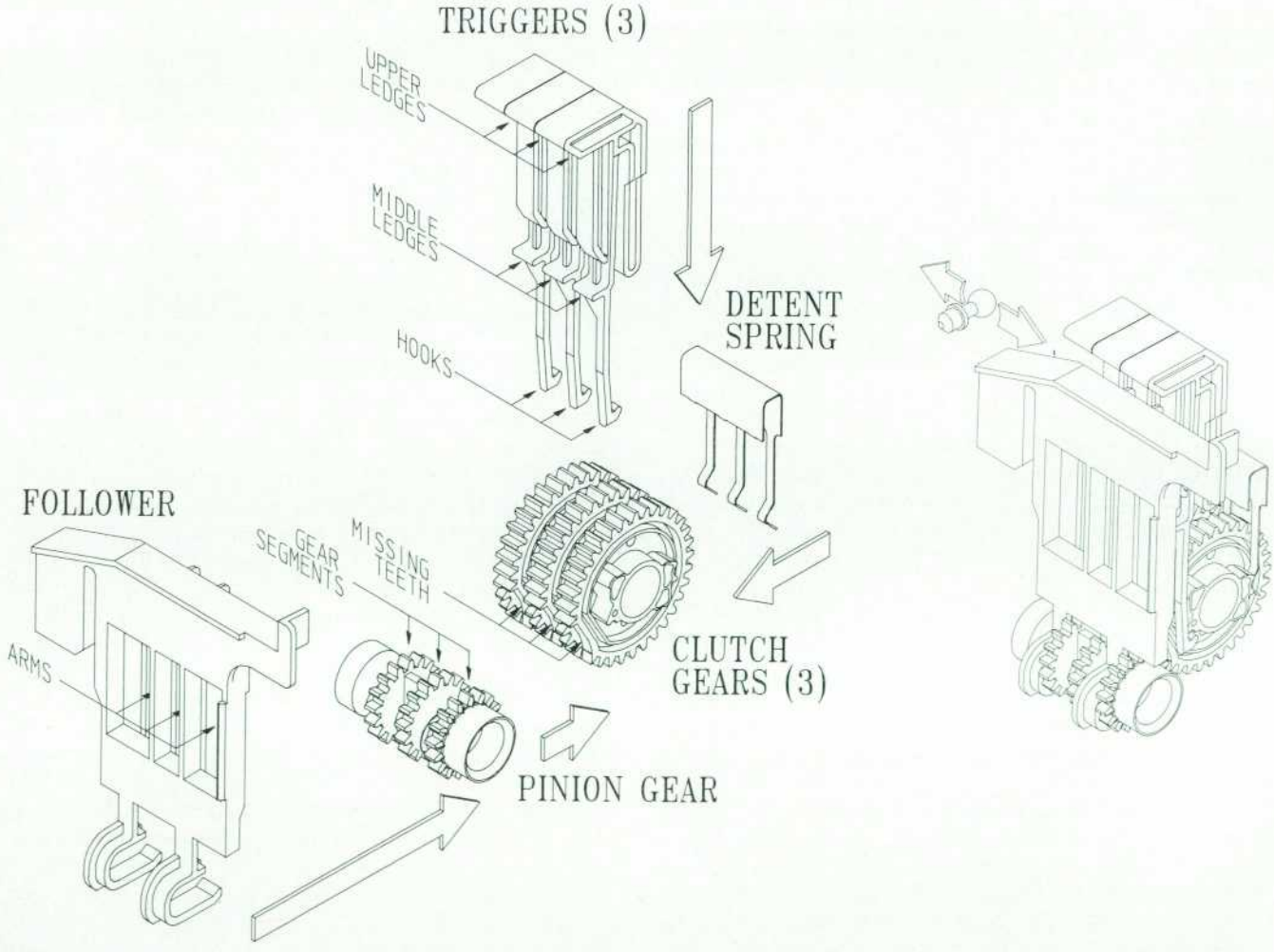



**Fig. 3.** *(Left) Exploded view of the transmission, which supplies power to three mechanisms from a single motor. (Right) Assembled transmission.*

tabs on the output rails, pushing them back from under the sheet of paper. Suddenly unsupported, the paper drops into the output bin to dry. By the time the following sheet is printed and dropped, this first sheet will be dry. When the print job is complete, the output will be stacked in the output bin, conveniently facing the user.

Prototypes were built with fixed output rails that did not pull back from under the paper, and in most cases, the paper dropped into the output bin successfully as soon as it was no longer pinched between the carriage guide and the platen. With certain particular graphics patterns, and under certain environmental conditions, however, heavy bars of wet ink would swell the paper slightly, effectively forming stiffening ribs in the paper. This reinforced paper was stiff enough to support itself with the minimal support offered by the fixed output rails, and the paper would not drop. While this result was rare and not disastrous (the following sheet would knock the troublesome sheet either into the output bin or onto the floor), the moving rails eliminated the problem completely.

## Transmission

The Deskjet printer is designed with low cost in mind. One of the ways of keeping costs down is to get the maximum use out of the motors. Three mechanical operations are required of the Deskjet mechanism in addition to positioning the paper and the printhead. First, the pressure plate must be raised and lowered to load a sheet of paper into the print zone. Second, the platen needs to be rotated down and the output rails opened to eject a sheet into the output tray. Finally, the peristaltic pump is operated to prime the pen. A low-cost multiplexing device to supply power for these three independent operations from one of the motors was a design goal of the Deskjet mechanism.

There are several constraints placed on the design of this multiplexing transmission. Minimal cost is a primary constraint. Cost must be below the cost of adding additional motors, clutches, or solenoids and be sufficiently lower in cost to warrant the added development costs. Laser-quality print requires that loads placed on the carriage servo be minimal, so the paper drive motor is the motor of choice for supplying power for the three operations mentioned. The carriage motor can be used, but only as an actuation device with light loads. This constraint takes some of the burden off the development of the carriage servo system.

Another constraint on the design of the multiplexing transmission is that, because the paper motor must accurately position the paper while paper is in the print zone, no additional loads are allowed on the paper motor while printing. A final constraint is to implement the design without the use of additional electronic components such as sensors, switches, or solenoids. This constraint is intended to keep costs down and can be removed if that appears to be the lowest-cost alternative.

The DeskJet transmission provides the desired functionality within the constraints listed. Power is delivered to each of the three systems through three gear trains. The transmission takes the power from a gear driven by the paper motor and transfers it to one of the three gear trains. The selection of a particular gear train is done by an actuator on the printhead carriage. The eject operation occurs after printing is complete. The priming sequence is used before printing starts, and the paper load operation is timed to be complete before the paper is positioned for printing.

### Transmission Design

The transmission consists of five parts plus the carriage actuator (Fig. 3). The five parts are a segmented pinion gear, a clutch gear, a follower, a trigger, and a detent spring. The carriage actuator consists of a spring with an effector attached to its end.

The segmented pinion gear consists of three gear segments placed next to one another, one for each gear train. One of these gear segments meshes with a gear driven by the paper drive motor. The gear segments are spaced apart with axial hubs. Two of the hubs are offset from the gear centerline to act as cams. The follower rides on these two cam hubs. The carriage actuator slides along the upper surface of the follower. The follower has three arms, one for each gear train. Each arm has two ledges, an upper ledge and a lower ledge. These ledges are used to position the triggers at the limits of their travel when not actuating a gear train.

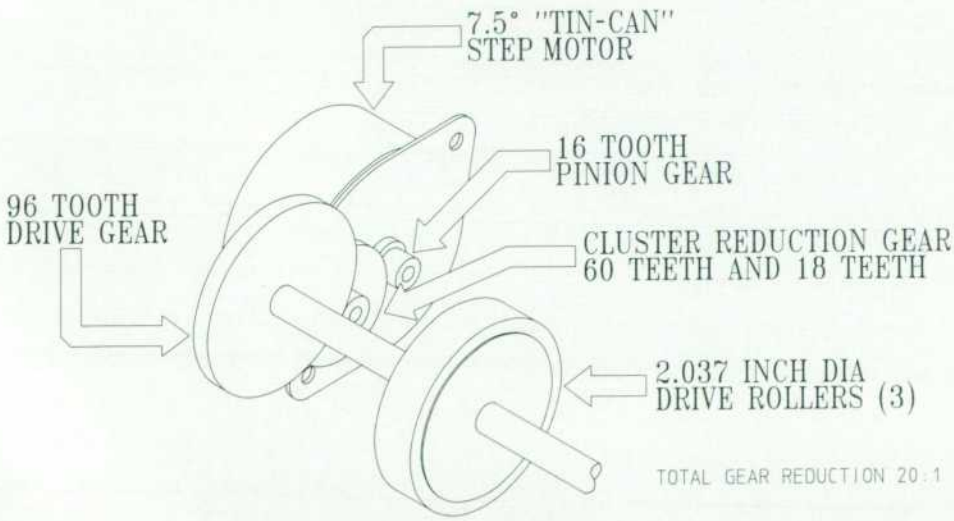



**Fig. 4.** *Paper drive motor assembly.*

There is one clutch gear for each gear train (three total). The clutch gear teeth have a face width that is twice as large as the gear segments on the pinion gear. These two gears are positioned so the pinion gear segment uses only one side of the clutch gear. The clutch gear has several teeth cut away so the pinion gear cannot drive the clutch gear. For the pinion gear to drive the clutch gear, the clutch gear must be engaged by an external driving element, which is supplied by the trigger (discussed further below). Once the clutch gear has engaged the pinion gear, the clutch gear is driven for one revolution. At that time, the missing teeth prevent the clutch gear from continuing. The clutch gear is detented into position by the spring.

The final part of the transmission is the trigger. There is one trigger for each gear train (three total). There are three details on the trigger. At the top is the ledge, which the carriage actuator uses to lift the trigger and engage the clutch gear. At the bottom is the hook, which mates to a detail on the clutch gear. The third detail is a ledge in the middle of the trigger. This ledge is used by the follower. When the follower is at the upper limit of its travel, the trigger is lifted to pull the clutch gear into its detented position. When the follower is at the lower limit of its travel, the trigger is pulled down to ensure that it will hook the clutch gear as it completes its cycle.

The transmission activates a specific gear train when actuated by the carriage. The gear train remains in motion for one revolution of the clutch gear. The system is self-initializing and requires no additional electronic input beyond the two motors. The components described met cost requirements. An additional feature of the design is that additional gear trains can be driven by adding one more clutch gear and trigger per gear train.

## Paper Drive Motor

The objective for the DeskJet printer's paper drive motor and gear train was to attain laser-printer quality at 300 dots per inch with a quiet, low-cost, high-torque, easy-to-assemble drive system.

In the investigation phase, many drive systems were assembled and evaluated. We first attempted to use a small 1.8-degree hybrid step motor commonly found in disc drives. The advantage of this drive was the high torque and high precision inherent in the hybrid step motor. Unfortunately, this system did not meet our needs because of the large amount of mechanical vibration the motor produced when overdriven. The motor had to be overdriven with a high voltage drive to deliver the large amount of torque necessary to drive the printer. For reasons of manufacturability, the printer has many loose-fitting snap-together parts. These loose parts amplified the motor's vibration so much that it sounded like a fire alarm. Since the DeskJet printer is supposed to be silent, we decided to switch to a permanent-magnet, 7.5-degree "tin-can" step motor. The advantages of the tin-can motor are low cost and no vibration, but with the sacrifice of resolution and accuracy.

Since the resolution of low-cost tin-can motors is limited to 7.5 degrees per step, we were forced to use a high gear reduction to attain the 300-dots-per-inch accuracy requirement. With the printer's large 2.04-inch-diameter drive roller, the gear reduction that yields 1/300 inch per step is 40:1. At 40:1, the step motor would have to rotate at 600 full steps per second to feed paper at the minimum desired form-feed rate. With tin-can motors, the available torque drops off rapidly with increasing speed, preventing the use of the 40:1 gear reduction. The compromise was to use a 20:1 gear reduction that yields 1/150 inch stopping resolution, and to use the firmware to shift the dots in the printhead to achieve the 1/300-inch drop placement resolution.

The final gear layout is illustrated in Fig. 4. One advantage of this drive system is the large diameter of the drive roller feeding the paper. The linefeed error because of runout of the drive roller and drive gear is relatively small because the error is inversely proportional to the drive roller diameter. This allows us to use inexpensive methods for manufacturing the roller. The biggest disadvantage to this system is its high susceptibility to runout in the tin-can step motor shaft. The error caused by shaft runout is:

$$\text{Error} = A \int (\text{FIM}/2)\sin(\theta)\,d\theta$$

where A is a constant taking into account the gear reduction and drive roller diameter, and FIM is the full indicator runout of the shaft and pinion. This function is maximum at $\theta = \pi$ radians. Unfortunately, the trade-off between speed and resolution forced us to design a gear train that rotates the motor shaft $\pi$ radians for a standard 1/6-inch linefeed. Therefore, we had to work with the motor manufacturer to minimize the large amount of runout inherent in tin-can step motors.

### Problems and Solutions

Using one motor to pick paper, prime the pen, eject paper, and accurately position the paper required some design trade-offs. The motor has to supply a high torque at slew speed to drive all these functions. Life testing with the first molded parts revealed that the torque required to operate the paper-pick cam increased rapidly with time. The motor

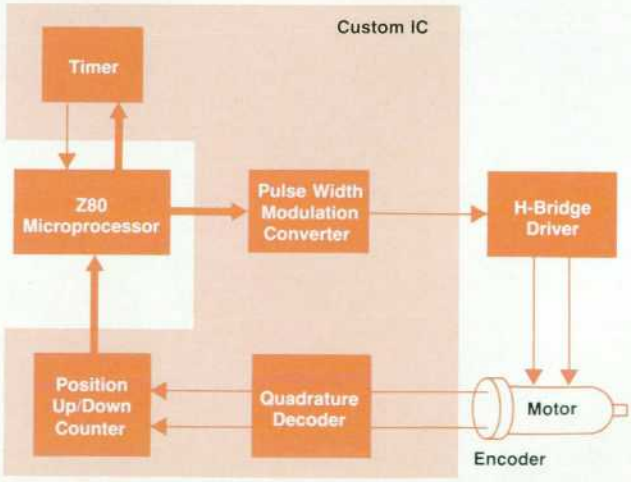



**Fig. 5.** *Block diagram of the carriage servo system for the DeskJet printer.*

would stall after only 100 pages of print. The increase in torque was caused by wear between the plastic chassis posts and the nine plastic gears in the paper-pick gear system. After testing many different material combinations, we finally were able to come up with a combination of carbon, glass, Teflon, and silicon fillers for polycarbonate, polypropylene oxide, and acetal plastics that would not cause an increase in torque.

To measure and evaluate the linefeed errors, a vision system was developed. This allowed us to measure dot placement accuracy down to ±0.0002 inch, which is necessary when attempting to evaluate a system with linefeed errors below 0.002 inch. The vision system allowed us to determine the periodic nature of the errors and pinpoint which gear was causing a problem. For example, a large error would occur every time the follower in the transmission reversed direction. The fluctuation in the torque load caused the drive roller shaft to lift up in its sloppy bushing, thereby preventing the drive roller from rotating the proper amount. The solution was to design in a preloader spring that keeps the shaft preloaded in one position within the bushing.

### Quality Assurance

Since at least ten different tolerances can affect the linefeed accuracy, a method of monitoring the linefeed had to be implemented in production. The error is measured twice daily and plotted on control charts to ensure that the DeskJet printer maintains its laser-quality print.

## Carriage Motion Control

The decisions made in selecting and designing the motion control systems for the DeskJet printer reflect the overall goals for the printer. The primary goal, excellent print quality with 300-dot-per-inch resolution, requires precise knowledge of the carriage position and the ability to position the paper precisely. Cost goals required that parts costs be kept to a minimum, and the need for silent operation dictated the use of quiet motors.

### Carriage Motor

A brush-type dc servo motor fit our requirements best and was chosen to drive the DeskJet printhead carriage. A hybrid step motor was considered, but a servo motor was chosen instead, for two reasons. The first reason is the relatively silent operation of the servo motor, which is important for desktop operation. The second reason is that the encoder needed to control the servo motor can also be used to provide position information for firing the printhead, ensuring accurate dot positioning over varying operating conditions. A brush-type servo motor was chosen over a brushless motor for its lower cost.

### Carriage Mechanical Hardware

The DeskJet printhead rests in a carriage that slides on a stainless steel rod. Paper is fed along the bottom of a sheet-metal guide and the carriage slides along the top of the guide, so the spacing between the printhead and the paper is controlled. The carriage is held against the guide by gravity. The carriage motor is mounted on the right side of the chassis and drives the carriage via a toothed belt and pulley. The pulley has 21 teeth and a tooth pitch of 0.08 inch/tooth, giving it a circumference of 1.68 inches. Mounted under the motor is an encoder with 504 slots which, over 1.68 inches, give the encoder an effective resolution of exactly 300 dots per inch. This allows the encoder output to control the firing of the printhead.

### Carriage Servo Electronics

Most of the electronics required to control the carriage servo are contained within the custom CMOS IC described on page 77. The principal components of the servo are a timer that sets the servo sampling rate, a quadrature decoder and 16-bit up/down counter that convert the two outputs of the encoder into a position that can be read by the microprocessor, and a pulse width modulator that converts an eight-bit output from the microprocessor into a pulse train with a duty cycle proportional to the processor output (Fig. 5).

The printer's Z80 microprocessor is interrupted by the timer, causing the microprocessor to read the carriage position out of the up/down counter. The microprocessor then applies a control algorithm to the position and computes an output, which goes to the pulse width modulator. The output of the pulse width modulator drives a high-power monolithic H-bridge driver, which drives the carriage

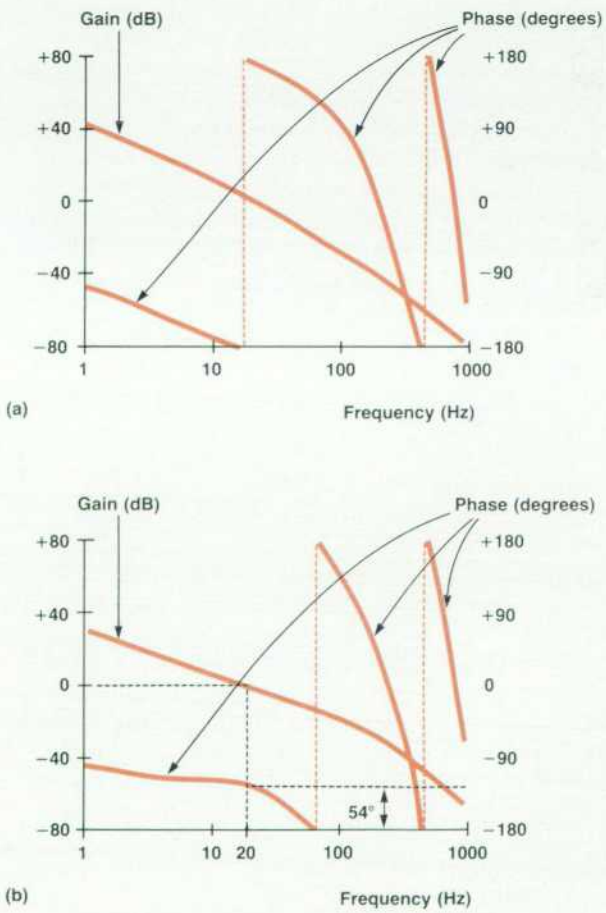



**Fig. 6.** *Loop gain of the low-gain printing servo. (a) Uncompensated. (b) Compensated.*

motor. Pulse width modulation is used to drive the motor for its high efficiency. This keeps the power dissipation in the drivers low, eliminating the need for heat sinks or fans. The pulse width modulator operates at 19.2 kHz to prevent it from generating audible noise in the motor.

### Servo Performance Requirements

The main goals for the carriage motor control system are accurate velocity and positioning control. It is important to have good velocity control while printing to maintain the print speed while also maintaining good print quality. If the carriage speed drops while printing, the throughput of the printer will decrease and printing will take longer. If the carriage speed increases while printing, the maximum fire rate of the printhead can be exceeded and print quality will suffer. Good position control is necessary to move the carriage to the correct position before starting to print and to stop the carriage in the correct position.

A velocity control servo would have provided good velocity control while printing, but it would not have provided the positioning capability we desired. A position control servo gives us the ability to position the carriage accurately, and also gives us more accurate control of the carriage velocity while printing. A velocity control servo will have a small steady-state velocity error, but a position servo has zero steady-state velocity error.

Use of a position control servo does introduce problems that would not occur with a velocity servo, however. The first problem occurs when moving at a constant velocity. In a velocity control servo, the reference is a constant velocity. When using a position control servo, the reference is a constantly changing position, and the change in the position reference divided by the time between servo samples is equal to the desired velocity. The other problem created by a position control loop is the introduction of a dc pole, which makes compensation of the servo more difficult.

### Physical Plant Model

The motor/carriage system is modeled as a second-order system, with perfect coupling between the motor shaft and the carriage. The transfer function for the motor/carriage system is:

$$M_1(s) = \frac{W(s)}{V(s)} = \frac{K_t}{(sL + R)(sJ + D) + K_eK_t}$$

where the voltage from the pulse width modulator amplifier, V(s), is the input and the shaft velocity, W(s), is the output. $K_t$ is the torque constant and $K_e$ is the voltage constant of the motor (these variables are equal when MKS units are used), L is the motor terminal inductance, R is the motor terminal resistance, J is the system inertia, and D is the system damping. However, this is a position servo loop, so

$$M_2(s) = \frac{O(s)}{V(s)} = \frac{K_t}{s((sL + R)(sJ + D) + K_eK_t)}$$

where the position, O(s), is the integral of the velocity.

The encoder is modeled as a simple gain,

$$K_p = 4K_{enc}/(2\pi)$$

where $K_p$ is the encoder gain in counts/radian, and $K_{enc}$ is the encoder line count in lines per revolution. The encoder outputs go to the position counter, which keeps track of position and is read by the microprocessor when it is interrupted by the timer.

The pulse width modulator amplifier is also modeled as a simple gain,

$$K_a = V_s/K_{pwm},$$

where $V_s$ is the motor supply voltage and $K_{pwm}$ is the pulse width modulator count corresponding to 100% duty cycle (full output).

The loop gain without the controller is the product of the three models,

$$M(s) = \frac{(2K_tK_{enc}V_s)/(\pi K_{pwm})}{s((sL + R)(sJ + D) + K_eK_t)}.$$

This system has three poles. The first is at zero and is caused by the integration of velocity to position. The other two poles are located at about 4 Hz (the "mechanical" pole) and 225 Hz (the "electrical" pole).

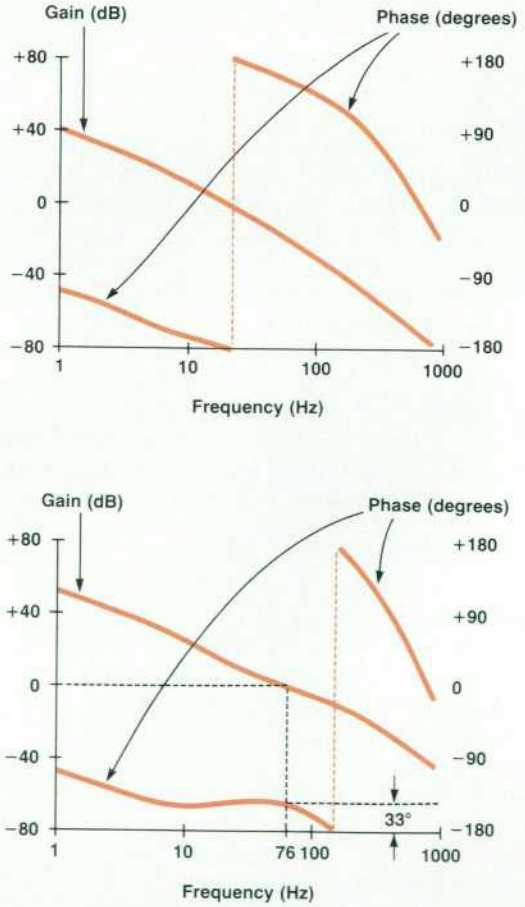



**Fig. 7.** *Loop gain of the high-gain positioning servo. (a) Uncompensated. (b) Compensated.*

### Stability and Sampling

The design goal for stability of the servo was 30° of phase margin. The process of sampling adds an additional phase shift to the loop. That phase shift is modeled as

$$\phi = e^{sT/2},$$

where T is the sampling period. The phase shift at any given frequency increases as T increases, so it is desirable to keep the time between samples as short as possible to minimize the phase shift caused by sampling. The system microprocessor has many other tasks to perform while printing, so the amount of time that can be dedicated to servicing the servo is very limited. This requires that we limit the closed-loop bandwidth to obtain the phase margin that we require, and also limits the dc gain of the servo. Fortunately, when printing we want good velocity control, but precise positioning accuracy is not as important (we still know the position accurately, however). At those times when we do want to be able to position the carriage more accurately (e.g., when positioning the interposer arm in the transmission) the printer is not printing, so we can use some of the processor bandwidth that would ordinarily be used for processing dot data for servo control instead. This led us to implement a two-servo system: one for printing and the other for positioning.

### Servo Design Methodology

Lead-lag compensation is used for both servos. A pole and a zero are placed at equal ratios above and below the desired crossover frequency to maximize the phase margin. The compensation was designed using classical control theory in the continuous domain and then converted to the discrete domain.

The Z80 microprocessor does not have much arithmetic capability, so an iterative design process was used to keep the compensation algorithm as simple as possible. The compensator transfer function was converted to a discrete control algorithm using a bilinear transform with prewarping at the crossover frequency, and then the coefficients were truncated to make the arithmetic easy. The new algorithms were then converted back to the frequency domain and compared to the original goals. This process resulted in simple algorithms that meet our performance requirements.

### Low-Gain Algorithm for Printing

The low-gain servo operates at a sampling frequency of 300 Hz to minimize the demand on processor bandwidth when printing, while maintaining good velocity and position control. The design goals were for a crossover frequency of about 24 Hz (required for the low sampling rate) with at least 30 degrees of phase margin. The actual crossover frequency is about 20 Hz and the phase margin is about 54 degrees (Fig. 6).

### High-Gain Algorithm for Positioning

A higher sampling rate (about 1200 Hz) is used when positioning the carriage in the transmission or in the service station. The higher sampling rate allows a wider servo bandwidth and higher gain, which in turn allows more accurate positioning. The initial design goal for the high-gain servo was a crossover frequency of about 72 Hz and 30 degrees of phase margin. The compensated loop gain for the algorithm actually implemented has zero dB gain at about 76 Hz, and has about 33 degrees of phase margin (Fig. 7).


### Acknowledgments

Paul Harmon, Bill Huseby, Kevin Moon, and John Rhodes assisted with the design of the mechanism. Randy Krauter, Vance Stephens, and Bob Stavig provided manufacturing engineering support. Gene Frederick was responsible for plastic part tooling. Jeff Ward was responsible for sheet metal tooling, carriage motor procurement, and various design concepts. Jim Burruss at HP's Vancouver Division and Steve Witte and Mark Majette at HP's San Diego Division provided valuable advice and assistance in the development of the DeskJet servo.


### Reference


1. W. Kurz, et al, "State-of-the-Art CAD Workstations for Mechanical Design," *Hewlett-Packard Journal*, Vol. 38, no. 5, May 1987, pp. 4-15.

# Data To Dots in the HP DeskJet Printer

*A microprocessor-controlled custom IC manipulates dot data to provide double-width, half-width, compressed, half-height, draft-quality, bold, underlined, and tall characters, and graphics, too.*

**by Donna J. May, Mark D. Lund, Thomas B. Pritchard, and Claude W. Nichols**

THE BASIC FUNCTION of the HP DeskJet printer is to transform input data into tiny ink dots on a page. The DeskJet printer offers high-quality characters in a variety of algorithmic character enhancements. As a result, the data must be transformed by a number of processes before being sent to the printhead.

### Microprocessor and Custom IC

The first few of the processes are performed by a Z80 microprocessor. These processes include receiving the data from the data communications hardware (RS-232-D or parallel), parsing and formatting the data, and translating this data into the form required by the hardware. At this point in the transformation, a custom coprocessor IC determines the pattern of dots to be printed and generates the printhead firing signals required to print the dots.

### DeskJet8 Character Set

The first process of interest in this transformation is the character set mapping. For the DeskJet printer to meet the needs of an international market it has to support a number of different character sets. These character sets include HP Roman8 (from which 13 ISO substitution character sets can be obtained), PC-8 (IBM character set), PC-8 Denmark/Norway, ECMA-94 Latin 1, and Legal. To maximize the number of character sets that can be stored and formatted, it was decided to combine all of these character sets into a single character set, eliminating all duplicate symbols.

To represent any of the 309 unique symbols in these five sets, 9 bits are required. This is wasteful of precious storage in an 8-bit environment. However, many of the 309 symbols contain a component that can be found in other symbols. Examples of such components include diacritical marks and segments of line-drawing characters. By further reducing the 309 symbols to unique components it is possible to achieve a symbol set of 256 components, with no symbol reduced to more than two components. The resulting component set, known as DeskJet8, can be used to create any of the 309 symbols. A symbol that consists of two components is known as a compound character. Special hardware ORs the two components together so they can be printed in a single pass. Thus, any one of the 2,240 symbols contained in the 18 character sets can be represented by at most two DeskJet8 characters. Character set mapping involves determining which character from the DeskJet8 set (or character components, for compound characters), is equivalent to the requested character from the requested set (HP Roman8, PC-8, etc.). Because the DeskJet8 set can provide five character sets from one, there are fewer fonts for the user to purchase and fewer fonts to be supported.

### Character ROM Data Storage

Without any sort of character ROM compression, DeskJet8's 2,240 symbols would require about 1M bytes of ROM per font. The above-mentioned DeskJet8 reduction in the number of characters actually stored in ROM and a further zeros byte compression on the ROM data result in practical DeskJet character ROM sizes of approximately 30K bytes.

In the zeros byte compression scheme, a flag byte is associated with each column of a character. The flag byte denotes which bytes of the 50-bit column are nonzero (contain at least one dot to be printed). Only the nonzero bytes are stored, reducing the amount of ROM required.

### 300-dpi Characters on a 600-dpi Grid

Although the DeskJet printhead can only fire each nozzle at the rate of 300 dots per inch, characters are designed on a horizontal grid of 600 dpi. By restricting dots within a row to being at least 1/300 inch apart, the printhead firing limits are not exceeded.

Fig. 1 shows the improvement gained by doubling the horizontal placement opportunities. Some of the algorithmic enhancements, which will be explained later, result in dot patterns with dots only 1/600 inch apart in a dot row. A cleanup circuit in the custom IC removes any

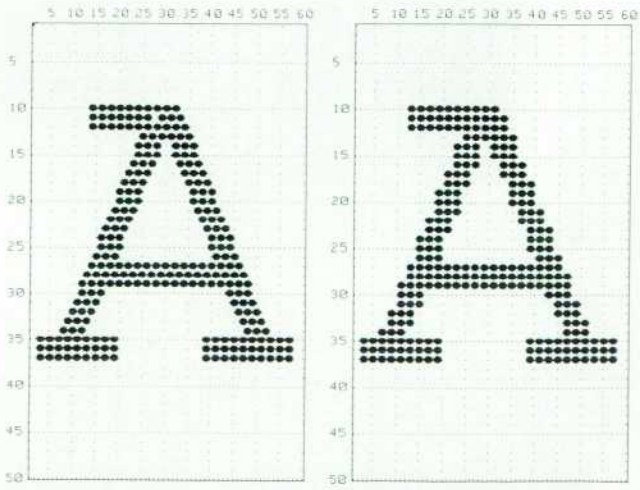


**Fig. 1.** *Character appearance is improved by doubling the opportunities for dot placement. (left) Positioning on a 1/600-inch grid. (right) Positioning on a 1/300-inch grid.*

## The DeskJet Printer Custom Integrated Circuit

Even a very fast microprocessor would not have the time to perform all of the required character enhancements and other dot manipulations at the 300-dot-per-inch resolution of the DeskJet printer. So instead, a relatively slow, inexpensive, 4-MHz Z80 processor is used to control a large custom IC.

Approximately 85% of the logic in the custom IC is there to handle the dot data, as described in the accompanying article. The IC also handles serial and parallel data communications, controls many logic functions required by the paper and carriage motors, provides timer functions to the Z80, and performs several external chip selects.

Contained within an 84-pin plastic leaded chip carrier package is logic laid out as two standard cell blocks, as shown in Fig. 1, and a large custom 50-bit wide data path corresponding to the 50 nozzles of the printhead. There are approximately 80,000 field effect transistors in a die measuring 6.8 millimeters by 7.6 millimeters. A high-density CMOS process is used to fabricate the chip. Gate widths are 1.2 micrometers.

Thanks to excellent design tools and good communications and cooperation among all involved, the first-pass silicon passed the complete set of test vectors without modification, and no turnarounds were required to get the IC into production. The custom IC was totally designed and is currently tested and manufactured within Hewlett-Packard.


#### Acknowledgments

Many people were necessary for the successful completion of the DeskJet printer custom IC, but special thanks go to Greg Hillman, Ray Pickup, Bob McClung, Daryl Anderson, Paul Liebert, Paul Krueger, Tim Brown, Liz Myers, Phil Wingebach, and Chuck Jarvie.



*Tom Pritchard*
Development Engineer
Vancouver Division

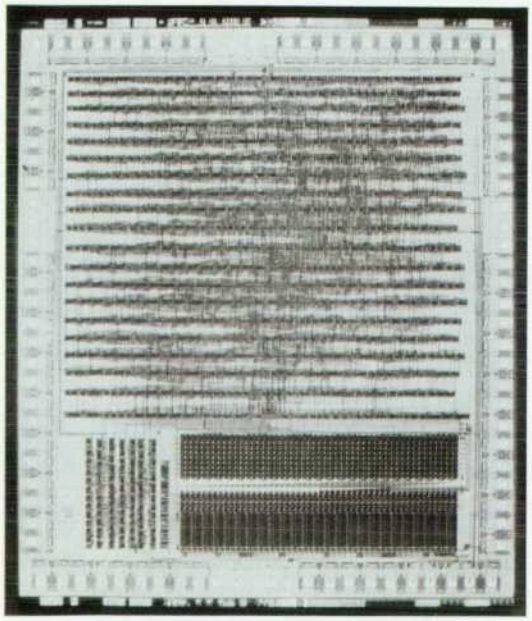



**Fig. 1.** *The custom IC in the DeskJet printer is a coprocessor that handles dot manipulations, data communications, and various logic, control, and timer functions.*

dots that exceed the limits, since they cannot be fired by the printhead, and even if they could, would result in too much ink on the paper. This dot dropout is performed after all requested enhancements have been applied to the dot pattern.

The dots in Fig. 1 are smaller, relative to the character size, than the actual dots. This is to make the dot positions easier to see at the 1/600-inch spacing. Fig. 2 shows the character of Fig. 1 with the actual dot-to-character proportions.

### Algorithmic Enhancements

The link in this process between the firmware and the custom IC is a buffer in RAM, referred to as an image buffer. The firmware translates the character information into the form required by the custom IC and places it into the image buffer. The custom IC accesses the information by direct memory access.

The way in which information is arranged in the image buffer allows the dot data of two separate characters to be ORed together and printed in a single print pass. This feature is used to print compound characters, overstruck characters, and characters that partially overlap. It is also used in the algorithms for half-width and double-width characters.

### Double-Width Algorithm

The simplest way to generate double-width characters is to repeat each column of dots. So a character consisting of columns A, B, and C (1/600 inch apart) could be printed as AABBCC to double its width. However, the cleanup circuit, which eliminates dots that are too close together, would drop out the second A column, the second B column, and the second C column, since all the dots in these columns are 1/600 inch away from the identical dots in the previous columns. The result would be the original three columns spaced 1/300 inch apart, which would typically leave gaps within the character. The algorithm used in the DeskJet printer takes advantage of the ability to OR two characters in the image buffer together. It ORs the simplest case, AABBCC, with the same pattern shifted by one column:

```
1/600 inch→ |   | ←
            A   A   B   B   C   C
     OR     A   A   B   B   C   C
     ---------------------------
            A   A  A+B  B  B+C  C   C
```

The result, after the dot dropout, consists of column A, the OR of columns A and B, the OR of columns B and C, and column C, all 1/300 inch apart. This produces a more filled-out character than the simple algorithm.

### Half-Width Algorithm

Similarly, the easiest way to generate a half-width character is to drop out every other column. The DeskJet printer avoids this loss of data by ORing every two columns together. In the image buffer, this is done by ORing a "character" that consists of the odd-numbered columns of the original character with a "character" consisting of the even-numbered columns. To illustrate, consider a character with

columns A, B, C, D, and E. The odd-numbered columns, A, C, and E, would make up one "character" and the even-numbered columns, B and D, the other. The result would be:

| | | | |
|---|---|---|---|
| | A | C | E |
| OR | B | D | |
| | A+B | C+D | E |

By preserving all of the dot data, a higher-quality character can be achieved.

### Compressed-Width Algorithm

Compressed characters, 16.67-pitch, are generated by printing only selected columns of the characters. The columns to be printed are selected by the font designers, and these columns are flagged in the character ROM data. When the hardware reads the dot data for a compressed character, it ignores any columns that are not flagged. To achieve the best-looking compressed characters, the dot patterns of the original characters must be designed with this algorithm in mind.

### Half-Height Algorithm

In the half-height algorithm, the hardware ORs every two consecutive rows together and prints the character with the bottom half of the printhead. This algorithm produces a higher-quality character than one that throws away every other row.

Applying the half-width algorithm along with the half-height algorithm is useful for characters that are to be subscripts or superscripts. The subscript or superscript mode alone merely causes the characters to be printed 1/12 inch below or above the text line; the height and width of the characters are not altered. Using the half-width and half-height modes for superscripts and subscripts, the characters appear more balanced in size in relation to the rest of the text.

### Draft-Quality Algorithm

The draft-quality mode available on the DeskJet printer doubles the print speed from 120 cps to 240 cps. In letter-quality mode, 120 cps, the character columns are printed 1/600 inch apart. Since the printhead travels twice as fast in draft-quality mode, the distance between columns is doubled to 1/300 inch. Instead of simply printing every other column of the characters, the hardware ORs every two columns together after any other algorithmic processes have been applied. Besides increasing the throughput, draft mode uses less ink because of the greater distance between columns.

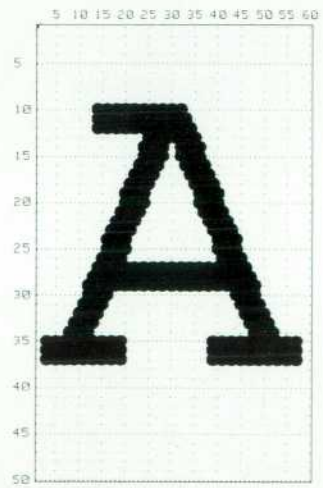



**Fig. 2.** *The dots in Fig. 1 are reduced in relative size to show dot placement more clearly. This shows the actual proportions.*

### Bold Algorithm

The hardware can generate two levels of bold characters. The level to be used is specified in the character ROM for each font, and the firmware communicates this information to the hardware through the image buffer. The lighter bold is used for smaller fonts and the darker bold for larger fonts. For the lighter bold, the hardware adds one dot per row, spaced horizontally at 1/300 inch, to the trailing edge of each character, resulting in a slightly wider and darker-looking character. For the darker bold, the hardware adds two dots instead of one, as shown in Fig. 3. This algorithm produces a result similar to a daisy-wheel printer, where a character is printed, the horizontal position is moved

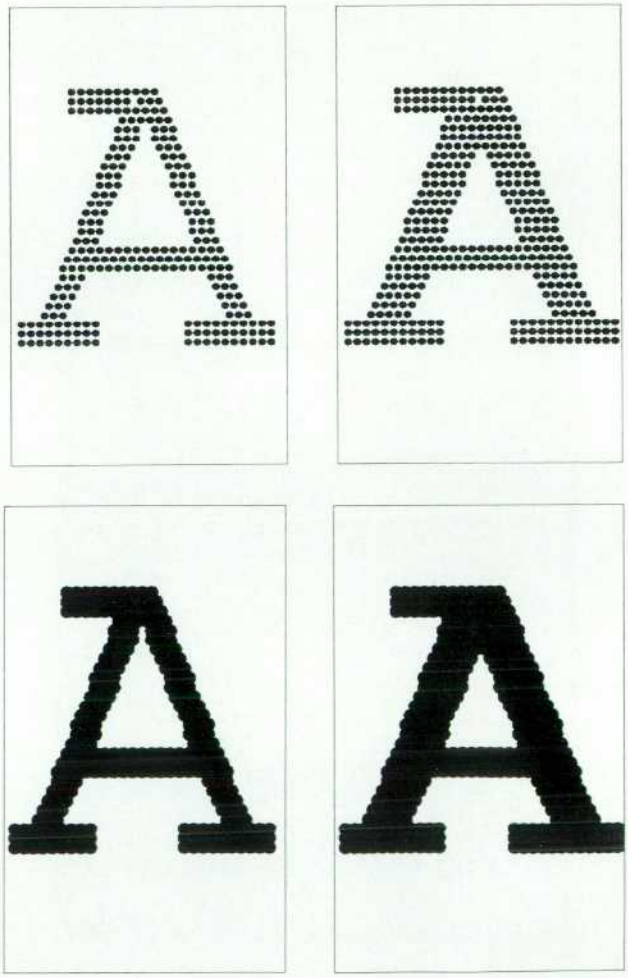


**Fig. 3.** *Normal and dark bold characters. (Top) Dots reduced in size. (Bottom) Normal dot size.*

## DeskJet Printer Font Design

In the printing business the printed page is the result that the customer sees and feels. Its look measures the quality of the information it contains. On the DeskJet printer project, print quality was one of the highest priorities, and hence, major emphasis was placed on character design, spacing, and ink-on-paper characteristics.

With the aggressive schedule, parallel design efforts were going on in all areas: mechanism, font design, firmware, printing characteristics, and final adjustments to the printhead specifications. This created moving targets for quality and implementation, making design flexibility a requirement for achieving quality goals.

The head technology allows a 50-dot, 300-dpi vertical resolution. The repetition rate allows the printing of every other dot in the horizontal direction with a 600-dpi resolution. With the 0.004-inch dot size, this half shift capability is very beneficial in smoothing the arcs, radii, and nonvertical angles of which characters are composed. Fig. 1, a bit map of a capital S in Times Roman Italic, shows an example. By composing the outline of the character first and then filling the center, optimum character definition is achieved. Unfortunately, during the development project no automatic filling algorithms were available, and this was done manually for over 300 dots in an average 12-point character.

The font/character editor we have used is a home-designed system, initially written in BASIC and rewritten in C for HP 9000 Models 350 and 320 workstations. The system provides dot-editing capabilities, character compiling, ROM and soft font generation, and screen-visible algorithms (bold, half size, compressed, and bold half size) using windowed menus and a mouse. After a character edit, a printed page can be available in two minutes for 30K bytes of information. This quick user feedback is essential when making minute detail changes.

Attaining overall page quality is a progressive process. We used outside typographers along with an internal Vancouver Division team to criticize our output and develop typesetting goals. The external consultants proved very beneficial in that they were unaware of the limitations of the technology and the product. Their opinions greatly raised our understanding of print quality.

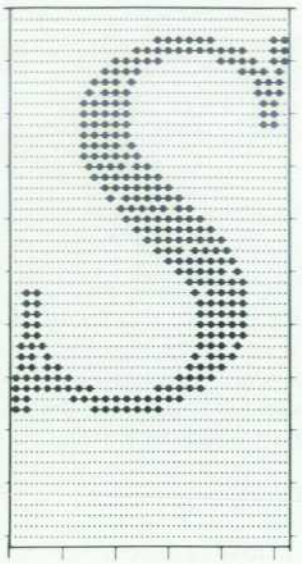


**Fig. 1.** *Half-shift dot positioning helps font designers smooth the arcs, radii, and nonvertical angles of characters, as shown by this bit map of a Times Roman Italic S.*

Major improvements in word and character spacing, character design, and contrast have been made and these will become more evident in later DeskJet printer products.

**Acknowledgments**

John Ignoffo, Robin Murray, and Julie Mixon were key individuals on the team that developed the DeskJet printer fonts.

*Bruce Yano*
Project Manager
Vancouver Division

slightly, and the character is printed again. Because the DeskJet printer is an inkjet printer, multiple-pass printing would put too much ink on the paper. With the DeskJet algorithm, the amount of ink is controlled.

### Underlining

Besides containing information about character data, the image buffers can contain auto-underline information. The custom hardware allows the firmware to specify any combination of the bottom 12 nozzles of the printhead to be used for an auto-underline. HP's Printer Command Language (PCL) defines two basic types of underline—fixed and floating. Floating underlines are specified in the character ROMs on a font-by-font basis. This allows for underlines of the optimum thickness and vertical placement for each font. Fixed underlines are independent of the font used. They provide a uniform underline when fonts with different floating underlines are to be underlined with a continuous underline. Besides being able to select fixed or floating underline type, a user can select single or double underlining. From the user's request and the information in the character ROM, the firmware determines the pattern to be used for the selected underline and communicates this information to the hardware through the image buffer. It also communicates the horizontal positions at which to start and end the underline. The hardware ORs the underline pattern with any character columns that fall in the same horizontal region.

### Tall Fonts

A feature provided by some of the optional font cartridges is tall fonts. These are fonts that have a point size greater than the height of the printhead (12 points). This means that these characters must be printed with two print passes separated by a paper advance. All of the algorithmic enhancements described above can be applied to these fonts.

### Graphics

Different data manipulation is required for graphics than for text. PCL specifies that graphics data be received in horizontal raster row format, yet the printhead needs data

in a vertical column format. This is accommodated by the firmware's rearranging the received bytes and sending them into a custom 8-bit-wide-by-50-bit-high memory array. This array can both shift bytes up and shift 50-bit-high data sideways. Graphics vertical resolutions lower than 300 dots per inch are achieved by repeating data entry into the memory array for multiple rows, causing a vertical expansion of the graphics data. Similarly, low horizontal resolutions are achieved by repeating data reads out of the memory array for multiple columns, causing a horizontal expansion.

### Printing

Once the firmware has put all the necessary information for a print pass in the image buffer, the printing can be done. The carriage velocity is controlled by the firmware, while a physical position register is updated by signals generated by a position encoder on the carriage motor. The carriage position determines when the columns are to be fired. Columns are typically fired every 139 microseconds.

### Printhead Considerations

The firing elements on the printhead are arranged in two columns spaced 10/300 inch apart. The 25 firing elements in each of these columns are spaced 1/150 inch apart vertically. The two columns are skewed vertically with respect to each other by 1/300 inch, so the result is 50 dots that are 1/300 inch apart vertically. A 25-bit-wide, 20-bit-long shift register cell is located in the custom IC to delay the firing of the trailing column of firing elements so that the two columns end up being printed at the same horizontal location.

One final manipulation of the column of data is needed before printing. The printhead is electrically organized as four groups of 12 or 13 resistors. One interconnect pad is used for each firing element and one for each of the banks. This configuration allows only one resistor from each bank to be fired at a time. The custom IC divides the 139-microsecond column time into 13 periods, and fires up to four resistors at a time. The orifice holes on the printhead are slightly staggered from two true columns to account for the positioning error caused by this sequential firing.

### Summary

From data to dots, all of this processing provides DeskJet printer users with a wide variety of enhancements that yield high-quality print. From the internal Courier 10-pitch set alone, a user can select one of 18 character sets (including the 13 ISO sets), one of four pitches (normal, half-width, double-width, or compressed), normal or half-height, normal or bold weight, and one of three vertical placements (normal, superscript, or subscript). Since all of these characteristics can be applied in any combination, the internal font alone can be used to generate 864 different fonts.

### Acknowledgments

All of the people involved in the DeskJet printer project contributed in some way to the success of the data-to-dots process. Particular mention goes to Mark DiVittorio for his role in guiding the decision-making process.

# Firmware for a Laser-Quality Thermal Inkjet Printer

*The firmware resident in the HP DeskJet printer is divided into generic printer code and printer specific code. An optional cartridge provides Epson FX-80 emulation.*

**by Mark J. DiVittorio, Brian Cripe, Claude W. Nichols, Michael S. Ard, Kevin R. Hudson, and David J. Neff**

THE DEVELOPMENT PROCESS for the firmware in the HP DeskJet printer represents a departure from the traditional firmware development process at Hewlett-Packard's Vancouver Division. Before the DeskJet project, the process for developing firmware for printing products was tolerably simple. All of the firmware was written in assembly language for the host processor of a single dedicated product. When another printer was to be developed (possibly with a different microprocessor) the firmware was totally rewritten and reuse was nil. All the tools used (editors, assemblers, and debuggers) were resident on HP 64000 Logic Development Systems, and other tools were virtually nonexistent.

The feature set of the DeskJet printer matches HP's PCL (Printer Command Language) Level III. Capabilities include printing graphics at 75 to 300 pixels per inch, mixing multiple type styles and sizes on a given page, performing enhancements such as bolding and underlining, downloading RAM-resident fonts, and printing letter-quality and draft-quality text. This command set had not been implemented previously in a high-volume product at the Vancouver Division. Given the DeskJet printer's ability to print at a very high resolution of 300 dpi and the objective of providing a very high level of formatting, we felt that a new approach was required.

A high-level language, C, was chosen to implement the DeskJet feature set, with the target processor being a Z80 microprocessor. The firmware is basically split into two categories: code that implements the generic feature set, called generic printer code, and code that interacts with the custom electronics and mechanism, called product specific code. Both segments of the code set are almost entirely written in C, although there is a small amount of assembly language code that performs paper motor control and provides feedback for the servo in the carriage velocity control system. This was necessary because these functions have to be done quickly and in real time.

This separation of generic printer and product specific code allows the generic printer code to be shared with another product, the HP RuggedWriter 480, an impact printer. The resolution, use, and functionality of the two products are quite different, but splitting the code into two parts paves the way not only for code sharing between these products but also for reuse of the generic printer code in products of the future. As an experiment, the generic printer code was ported to a different processor architecture under development. This effort was completed in approximately two months. The majority of defects found in the firmware of new products can now be expected to occur in the product specific code, since the generic printer code has been reused and is in a mature state. In addition, the engineering resources required to do products of the future should be reduced.

The development of the DeskJet printer code was facilitated by an HP 9000 Series 500 computer system. The HP-UX operating system provided access to a number of tools that were otherwise unavailable. This system also allowed updated versions of the working code set to be made available to all the designers for integration of the generic printer code and the printer specific code. Operating in the UNIX® environment also allowed extensive use of DTS, an in-house HP defect tracking system, to track defects as they occurred and were resolved. On the DeskJet project, defects were tracked from the breadboard phase through to manufacturing release. Originally, the system was used to track software defects, but eventually, separate tracking was also done on the electronic and mechanical defects that were found in various phases of testing.

## Generic Printer Code

Responsibilities of the generic printer code include pars-

The HP-UX operating system is HP's version of AT&T's UNIX System V.2 with extensions. UNIX is a registered trademark of AT&T in the U.S.A. and other countries.

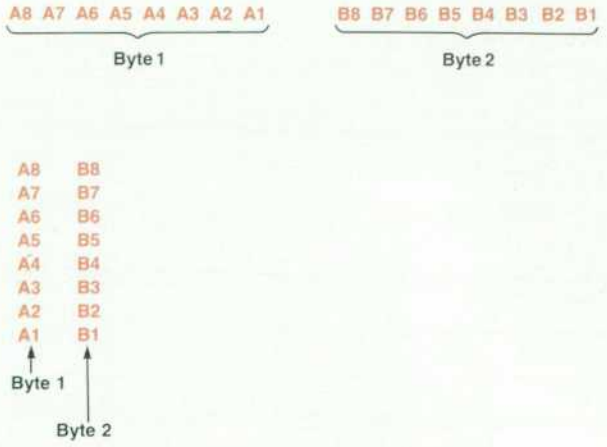



**Fig. 1.** *Column versus raster graphics format. Column graphics bytes are printed in consecutive 8-bit columns. Raster bytes are printed as rows of bits.*

## Slow-Down Mode

A common problem experienced by many printer users is that some applications for the MS-DOS® operating system have a tendency to disregard the MS-DOS system printer values. Particularly annoying is when an application ignores the parallel I/O time-out retry value. This value can be changed by the MS-DOS command MODE LPT1:,,P, which instructs the system to continuously retry all time-outs on the parallel port.

To prevent time-outs, the DeskJet printer parallel I/O firmware includes a mode known as slow-down mode. This mode is automatically invoked when data received over the parallel I/O port fills the input buffer of the DeskJet printer to within 60 characters of its capacity. In slow-down mode, the DeskJet printer accepts only one byte of data per second from the host, and the host will not time out. As data is removed from the input buffer to be printed, slow-down mode is exited and data once again is accepted as fast as the DeskJet printer can handle it.

MS-DOS is a U.S. registered trademark of Microsoft Corporation.

*Claude Nichols*
Development Engineer
Vancouver Division

ing of PCL data, PCL level III+ page formatting, dynamic memory management for the page buffer, and font support. The printer specific code is responsible for any special data formatting required by the print engine, for mechanism control, for management of the keys and lights on the control panel, and for I/O.

Supporting both the DeskJet printer and the RuggedWriter printer with a common set of firmware was a challenge because of the many differences between the two printers. The DeskJet printer's main contributions are high-resolution graphics, powerful font support, and data formatting, while the RuggedWriter printer emphasizes speed and paper handling.

### Generic Printer Operating System

The generic printer code is broken up into independent processes such as the parser and the page formatter. Most processes are written such that when they are called they will take some data out of their input buffer, process it, place some data in their output buffer, and return.

Process control is performed by the MCP (master control program) which is a minimal operating system designed to support processes without adding significant execution time overhead.

The MCP maintains a table of active processes. The scheduling algorithm implemented by the MCP simply invokes each process in the table in a round-robin fashion. The MCP has no ability to wrest control from a process that has been invoked. Processes are responsible for voluntarily relinquishing control.

In the normal sequence of events, the MCP calls a process's entry point, the process finds its input data and turns it into output data, and then the process returns control to the MCP. For example, when the parser is called, it attempts to read bytes from the input buffer until it has parsed a string of printing characters or a complete PCL escape sequence. It then sends out a token representing the data and returns to the MCP.

In addition to returning, a process can give up control by calling the MCP function Suspend. The Suspend function saves the state of the process stack and returns to the scheduling loop of the MCP. To reinvoke a suspended process, the scheduler restores the stack of the process and executes what looks like a return from the call to Suspend.

Suspending takes much more time and RAM than returning, so processes are encouraged to return whenever possible and use Suspend sparingly. Most processes suspend for one of two reasons. The first is that they have created some output data but there is insufficient room for it in the output buffer. In this case they must suspend and wait for a downstream process to create room in the output buffer. The other reason is that the input data stream has dried up unexpectedly. For example, the parser will suspend if it runs out of input data in the middle of an escape sequence.

Another feature provided by the MCP is that processes can be dynamically added to and deleted from the process table. This is called process activation and deactivation. For an example of why dynamic process activation is useful in a printer, consider the self-test process, which is responsible for creating the data for a printing self-test. This process is deactivated until the user explicitly requests a printing self-test, at which time it is activated.

The Z80 implementation of the MCP in the DeskJet printer requires approximately 1500 bytes of ROM and 350 bytes of RAM. Approximately 85% of the code is written in C with the other 15% in Z80 assembly language.

### Parser

One of the responsibilities of the generic printer firmware is the interpretation of printer commands sent in the form of PCL escape sequences. The parser translates these escape sequences into tokens, which are used by the page formatter to indicate functions to be performed. The parser determines whether an escape sequence is syntactically correct. The parser does not place product specific limits on the values used in the escape sequences. The product specific limits are evaluated by printer specific code at a later time. The parser receives data from a printer specific code function so that the details of I/O and data communication buffering are left under printer specific code control.

Another responsibility of the parser is the handling of graphics data. PCL graphics data is sent in horizontal raster format. The parser must store the data in a manner that facilitates removal of the data for printing in a vertical format. Furthermore, the data may be transferred to the printer in one of three different modes. Two of these modes involve data compaction and the parser must expand the data to the uncompacted form.

### Page Formatter

The DeskJet feature set includes many PCL Level III and IV commands for positioning to any arbitrary point on a page. However, because of potential ink-smearing problems, the DeskJet mechanism does not support backing up paper. Because of these two requirements, the generic printer code includes a page formatter, which is responsible for sorting all of the data before printing.

For cost reasons, the DeskJet printer only contains enough RAM to hold about 20% of a typical page, which is fundamentally incompatible with the need for a page formatter. The solution implemented in the DeskJet printer is a RAM-limited page formatter. The firmware is written as a page formatter that attempts to process all of the data on a page before sending off any of the page to be printed. However, if the formatter runs out of memory while processing a page, it sends off the data that is highest on the page (closest to the top) to be printed. The result is that the DeskJet feature set supports backward positioning for applications such as creating superscript characters, but it does not support full, random page addressing. This compromise requires DeskJet drivers to perform most of the vertical sorting of the data before it is sent to the printer, while still allowing features such as superscript characters to be easily accessed.

Data in the page buffer that can be printed with a single pass of the printhead is kept on a linked list, sorted by horizontal position, known as a task. Each task also has a header which is used to link all of the tasks on a page, sorted by vertical position.

Because data is typically received for a page in top-to-bottom, left-to-right order, it was found that the page buffer could be built much faster by keeping the lists sorted in reverse order. Thus the lowest task on a page is kept at the head of the task chain and the rightmost data of a task is kept at the head of the task. This organization means that most data is inserted into the page buffer at the head of the first task, which considerably reduces the time spent searching through the buffer for a place to insert the data.

### Font Support

PCL classifies fonts by a set of font attributes such as height, pitch, and stroke weight (boldness). Escape sequences sent to the printer can be used to specify desired values for each of the font attributes. Because the user can request any combination of font attribute values at any time, the firmware cannot rely on the fact that the requested font actually exists in the printer.

For example, the user may request a 12-point, 10-pitch font even if the only fonts available are 12-point, 12-pitch and 14-point, 10-pitch. In these cases, PCL specifies that the printer must perform a prioritized closest-fit algorithm to select a font. PCL specifies both the priority of the attributes and the rules used to determine closest fit. In the above example, the 12-point, 12-pitch font would be selected because pitch has a higher priority than height.

The generic printer select-font implementation must meet several requirements. The first is that it must handle the large number of fonts that can be stored or generated by the DeskJet printer. A second constraint is that there is no standard set of algorithmic font enhancements. For example, the DeskJet printer can algorithmically generate half-height characters and the RuggedWriter printer can italicize characters. Therefore, the select-font code cannot take advantage of a known set of font enhancements. The final requirement is that the select-font algorithm must be fast. Some printer drivers do foolish things such as force a select-font operation to occur for every character, so time spent in the select-font process can quickly reduce the throughput of the printer.

To satisfy all of these requirements, at power-up a printer specific code function (not part of the generic printer code) builds tables in RAM that describe all of the available fonts. There is one table for each of the font attributes. For example, there is a pitch table and a height table. An attribute table contains one entry for each of the unique values available for that attribute. Along with the attribute value, the entry specifies ID numbers for all of the fonts that can supply that value.

The following are examples of these tables:

| Algorithmic enhancements: | Available fonts: |
|---|---|
| - half-height | font 1 - 12-point, 12-pitch |
| - half-width | font 2 - 14-point, 10-pitch |
| - double-width | font 3 - 7-point, 24-pitch |

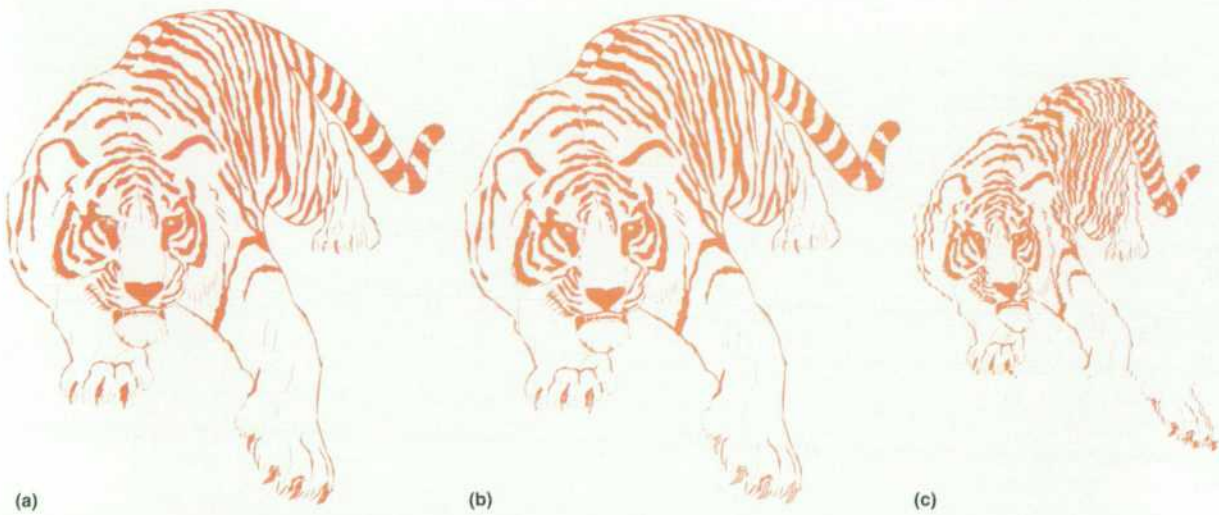



**Fig. 2.** *(a) Epson FX-80 printer output. (b) Output of HP DeskJet printer with Epson FX-80 emulation cartridge for same input. (c) Output of another printer without scaling ability.*

| Pitch table: | Height table: |
|---|---|
| 5-pitch - font 2 | 3.5 point - font 3 |
| 6-pitch - font 1 | 6 point - font 1 |
| 10-pitch - font 2 | 7 point - font 2, font 3 |
| 12-pitch - font 1, font 3 | 12 point - font 1 |
| 20-pitch - font 2 | 14 point - font 2 |
| 24-pitch - font 1, font 3 | |
| 48-pitch - font 3 | |

To select a font from these tables, the select-font code first initializes an eligible list with the IDs of all of the fonts in the printer. It then processes the highest-priority attribute by examining all of the available values, choosing the closest match, and then forming a new eligible list by a logical AND of the old eligible list and the list of fonts that provide the chosen value. This process is repeated for each of the font attributes in the order of priority specified by PCL.

The implementation of this algorithm uses bit strings for both the eligible lists and the lists of font IDs in the tables, so the AND step of the algorithm is very fast.

## Epson FX-80 Emulation Cartridge

The Epson FX-80 Emulation Cartridge was developed to maximize support of the DeskJet printer on existing software applications.

The DeskJet printer's native escape sequence or command set is HP's PCL (Printer Command Language). Prominent software vendors were preintroduced to the capabilities of the DeskJet printer command set to allow adequate time for them to support the printer before its introduction. There was still a major concern for those users who had previously purchased software or were using applications not targeted for preintroduction support.

Since the HP LaserJet II printer also implements the PCL command set and has the same basic print resolution of 300 dots per inch, an extensive effort was mounted to verify support of the DeskJet printer by software applications that have LaserJet drivers. Information about such support on a large number of software packages, including limitations, has been documented and included within the *DeskJet Owner's Manual*. There was still, however, a concern for existing applications that do not support either the LaserJet printer or the DeskJet printer.

To support the DeskJet printer on applications without LaserJet or DeskJet support, a decision was made to develop a cartridge that provides emulation of an escape sequence standard that has long been in existence and is commonly supported by software applications. Although Epson FX-80 emulation provides support on the multitude of existing applications as intended, it should be noted that the functionality of the FX-80 does not provide optimal support of all the features offered by the DeskJet printer. The FX-80 emulation support strategy is a backup strategy, not an attempt to maximize feature access and control of DeskJet capabilities.

The remainder of this article reviews two key elements in the firmware development of the FX-80 Emulation Cartridge. The first portion discusses how a major challenge, providing graphics aspect ratio compliance between an emulated printer and a target printer of greatly dissimilar base dot densities, was met. The second portion describes how an aggressive development schedule was attained while leveraging from a partitioned high-level-language firmware design that was also under development.

### Graphics Emulation

Besides supporting the command set, fonts, and features of the Epson FX-80 printer, it was important for graphics output to be dimensionally correct despite differences in DeskJet and FX-80 resolution. For graphics output, the DeskJet printer is a 300-dot-per-inch (vertically and horizontally) raster device. In contrast, the Epson FX-80 is a column-oriented graphics device with a vertical resolution of 72 dpi and selectable horizontal resolutions of 60, 72, 80, 90, 120, and 240 dpi. Fig. 1 shows the difference between column and raster formats.

Current inkjet products emulating Epson printers support graphics only in their native resolutions. Without some sort of printer driver customization, the resolution differences result in incompatibilities in both size and aspect ratio (see Fig. 2). To attain graphics output from the DeskJet printer that matches the FX-80 printer's using a standard Epson printer driver, incoming graphics data is scaled (to match resolution and aspect ratio) and rotated to DeskJet raster format.

When the resolution of the target device (the DeskJet printer in this case) is finer than that of the emulated device, scaling can be done simply by approximating the emulated dots with multiple target device dots. A simplified example, where the horizontal and vertical resolutions of the emulated system are equal and scaling error is neglected, is shown in Fig. 3. The horizontal or vertical expansion for an emulation dot can be determined by the following generalized equations:

$$d_t = (S_e + S_t/2 - E(n))/S_t$$

$$E(0) = 0$$

$$E(n + 1) = S_t d_t - S_e$$

$d_t$ is the number of target system dots to produce. $S_t$ is the number of units per target system dot. $S_e$ is the number of

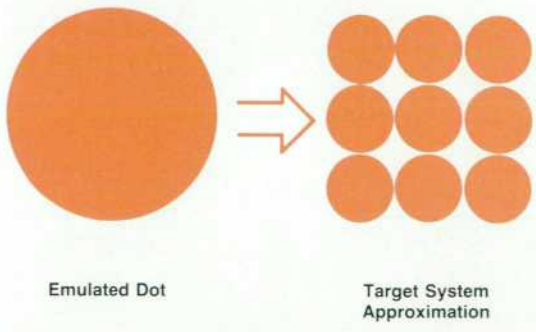



**Fig. 3.** *When the resolution of a target printer is finer that that of an emulated printer, scaling can be done simply by approximating the emulated dots with multiple target device dots.*

units per emulation dot, which may be different in the horizontal and vertical axes. E(n) is the cumulative scaling error.

As an example, take the case of scaling to an emulation resolution of 72 dpi vertically and 90 dpi horizontally. Use a convenient virtual resolution of 10800 units/inch—the least common multiple of the target system resolution and all emulated resolutions—so that only integer arithmetic is used. Thus, $S_t = 10800/300 = 36$ units/dot, and $S_e = 10800/90 = 120$ units/dot for the horizontal expansion or $S_e = 10800/72 = 150$ units/dot for the vertical expansion. Given this, each column of graphics can be expanded as it is received using the generalized scaling equations above.

In practice, it is desirable to eliminate the use of multiply and divide instructions to increase speed, which is critical when handling large amounts of graphics data. Fortunately, the equations can easily be reduced to algorithms requiring only addition and subtraction. The number of target dots on one axis for an emulation dot, $d_t$, is determined by repetitively adding $S_t$ to the constant $S_t/2 - E(n)$ until the result exceeds $S_e$. A target dot is output at each addition step. The next error term, $E(n+1)$, is simply the result of the additions minus $S_e$.

Continuing with the example, Fig. 4 shows the result of converting two rows of column graphics to DeskJet resolution and raster format.

For throughput enhancement, the emulation takes advantage of the fact that the vertical scaling is just duplication of the raster rows. Dots are not scaled vertically until the complete graphics line has been scaled horizontally. The reason for this is that once the horizontal scaling is done, the vertical scaling is simply a matter of copying the raster rows a given number of times, which is much faster than bit-by-bit scaling. This cannot be done with the horizontal scaling since scaled dots can cross byte boundaries horizontally. Another speedup involves trapping out white space—dot positions that are unused. Since the raster buffer is always cleared before it is used, blank emulation dots or whole columns do not have to be scaled into the buffer; only the position for the next dot or column needs to be calculated. Throughput is increased for any image that has at least 5% white space. The more white space, the greater the throughput gain.

The emulation allows users to replace the built-in characters with characters of their own design. User-defined characters are downloaded by specifying each column of the character, in the same manner as column graphics data is sent. Because the download data is sent assuming Epson FX-80 resolution, it must be scaled before printing, the same way graphics is scaled. In fact, any downloaded character is printed as graphics. Pica, elite, and compressed character pitches are produced by changing the horizontal scaling factor. Other enhancements such as bold, expanded, superscript, and subscript characters are added by manipulating the character data before printing.

### Mechanism Compensation

Even though the scaling algorithm can map all of the FX-80 resolutions to the DeskJet printer's 300-dpi resolution, there still exists a mismatch between the resolution of the paper and printhead placement in the DeskJet printer (1/300 inch) and the Epson vertical resolution (1/72 inch). Because of this, the mechanism error must be synchronized with the scaling process. This is done by monitoring the page position to detect when vertical positioning will move the paper 1/300 inch too far because of roundoff. When such a move is predicted, the scaling routine produces an extra row to compensate. The table below shows how mechanism and scaling positions are reconciled when scaling columns of eight bits.

| Virtual Position (1/10800 inch) | Mechanism Position (1/300 inch) | Rows Produced |
|---|---|---|
| 0 | 0 (0.0) | 33 |
| 1200 | 33 (33.33) | 33 |
| 2400 | 66 (66.67) | 33 |
| 3600 | 100 (100.0) | 34 |
| 4800 | 133 (133.33) | 33 |

### Leveraged Emulation Cartridge Design

The Epson emulation cartridge was highly leveraged from the DeskJet PCL code and the RuggedWriter Epson LQ-1000 emulation code. Over 85% of this code is written in C and the software architecture is partitioned into blocks with well-defined interfaces. These factors allowed the Epson cartridge development to meet a very aggressive schedule. Only eight engineer-months were required for software engineering. The total project time was four months.

As mentioned earlier, the DeskJet firmware is partitioned into major blocks. The parser, key panel handler, DIP configuration switch handler, and self-test modules all send tokens to the formatter. The formatter receives these tokens and builds tasks for the task processor. For the Epson emulation cartridge, the LQ-1000 parser was leveraged from the RuggedWriter firmware and the rest of the code was leveraged from the DeskJet firmware. Although the individual token handlers needed to be modified to handle the different parameters and their meanings, the formatter's

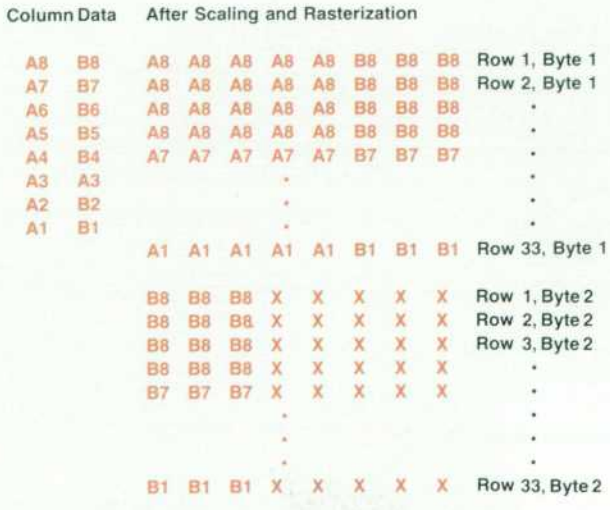



**Fig. 4.** *The result of converting two rows of column graphics to DeskJet resolution and raster format.*

task management code was kept intact. Since the key handler, dip handler, and self-test modules send PCL tokens, these modules had to be modified to send the corresponding Epson tokens whenever possible, but a few PCL tokens and their handlers had to be preserved to allow for functions that had no analogous Epson function (for example, the draft-mode key on the DeskJet printer).

### Monitoring Changes in Leveraged Code

The Epson cartridge was under development in parallel with the DeskJet firmware. Although the DeskJet code was the starting point for the Epson code, many changes had to be made throughout the code to send the appropriate Epson tokens and to perform the proper font selection behavior needed for Epson emulation. Typically a particular routine had several lines of code added or deleted to handle the specific needs of Epson emulation.

While the DeskJet code was being changed to incorporate Epson emulation, the DeskJet code was also being enhanced or changed to fix defects. Some of these changes needed to be incorporated into the Epson version, but some of the changes were PCL specific and had no relevance to the emulation project. To monitor these changes, a tool named difftree was created to make the tracking process semiautomatic.

difftree is a UNIX shell script based on the standard diff utility, which finds differences in specified source files in a directory and all subdirectories and creates a parallel directory structure containing files of the differences. In other words, diff finds differences between all source files in particular directory trees. It also logs situations when a source file or subdirectory is in one directory tree but not the other.

For example, the DeskJet project has a directory structure of:

```
/projects/deskjet/source
```

and the Epson project has a structure of:

```
/projects/fx80/source
```

and the subdirectories of source in both of the projects are largely the same. difftree can be run on the entire source tree or on selected subtrees. In practice, we ran difftree on selected subtrees of the code, since certain modules (e.g., the parser) were totally different. For the sake of this example, however, assume difftree was run on all source trees. A third directory name was supplied to contain the differences. A fourth directory name was supplied to contain a tree of expected or "OK" differences (explained below). Hence, difftree could be run by entering:

```
difftree /projects/deskjet/source /projects/fx80/source /projects/fx80/diffs
                                  /projects/fx80/okdiffs
```

This creates the directory /projects/fx80/diffs/source which contains all the differences. The diffs tree has files of the same names as the files compared in the source and destination trees. Hence, if deskjet/source/fileA is different from fx80/source/fileA, there will be a file fx80/diffs/source/fileA which contains the differences.

The most useful feature of difftree is its okdiffs directory tree, which is passed as a fourth directory tree parameter to difftree. When difftree finds differences between two files it compares these differences with a file of the same name in a corresponding okdiffs directory tree. If there is a file in this okdiffs tree of the same name as the files being compared and if this okdiffs file is identical to the differences just found, the differences file is purged. Hence, if a difference is detected and the reason for the difference is an intended divergence in the code, the difference file is simply moved manually to the okdiffs tree and difftree will no longer create a difference file (until one of the files changes again).

Manual involvement was still required to identify the need for incorporating changes in the DeskJet code into the Epson code, and to determine the best way to make the changes (add the Epson modifications to the new DeskJet code or the DeskJet modifications to the Epson code). difftree, however, provided a quick semiautomatic way to keep the Epson code up to date with the DeskJet code. During the last two months of the Epson project, difftree was run an average of once per week. Incorporating and testing any changes typically required only half a day of one engineer's time each week.

As long as the directory structure of the Epson code was kept the same as the DeskJet code, difftree performed quite well. At times, for various reasons, the DeskJet code had its directory structure rearranged. This required an analogous manual rearranging of the Epson directory trees to use difftree.

Although difftree is not totally automatic and requires human decision to decide which differences are irrelevant and which differences need to be addressed, it has proved to be an extremely useful tool. In our opinion it is unreasonable to expect a totally automatic solution, since human judgment is required to decide which changes need to be incorporated into leveraged code.

### Acknowledgments

The DeskJet software development team also included Martin Hash and Donna May. The authors would like to acknowledge their efforts and patience in the development of the software and their ability to adapt to a major change in the development process.

# Robotic Assembly of HP DeskJet Printed Circuit Boards in a Just-in-Time Environment

*A high-speed machine places most of the surface mount components while a vision-guided robot places small components and plastic leaded chip carriers.*

**by P. David Gast**

THE VANCOUVER DIVISION of Hewlett-Packard designs and manufactures personal workstation printers. Advances in printer design and more efficient use of printed circuit board space have made major contributions to improving the printers' price/performance ratio. The number of printed circuit boards in each printer has declined significantly, reducing both size and material cost.

The printed circuit boards in the HP DeskJet and RuggedWriter 480 printers are hybrid boards composed of a mixture of surface mount and through-hole components. Assembly of these boards is done on an automated high-volume assembly line capable of processing the boards for both printers in a mixed-mode production environment. This robotic printed circuit board assembly system is carefully designed to fit into the Vancouver Division's just-in-time (JIT) manufacturing system.

A component mix of many small surface mount components coupled with a few large odd-shaped parts is a key factor in the design of the surface mount assembly line. A high-speed pick-and-place machine is used to place the majority of the small components. The remaining odd components are placed by the slower but more flexible robot workcell. The line is designed to allow mixed-mode production of different products with a minimum batch size of one. Each printed circuit board is automatically identified by the robot. Board loading information is communicated automatically to the robot controller via an RS-232-D connection to an HP 9000 Model 320 Computer.

### Manufacturing Strategy

A just-in-time manufacturing philosophy has been effectively implemented at the Vancouver Division over the past five years. Work-in-process inventory has decreased dramatically since the implementation of the JIT system. The throughput time to process a complete printer from start to finish has dropped from over one week to less than four hours. On-line inventory space has been reduced to a level that allows twice the value-added shipments from the same floor space.

In a JIT system, demand for each subassembly exists only when the subsequent assembly creates the demand. Pulling a finished printer from the end of the line ultimately creates the demand for each preceding subassembly. For this reason, JIT production is termed a "pull" environment, with each assembly pulling the production from the previous workstation.

Extensive cooperation with the R&D design team is a requirement for products destined for automated assembly. Often the addition of a small feature that has no effect on the product's function or part cost can greatly simplify the automatic assembly of the product. Design changes made to improve manufacturability by automated equipment generally simplify hand assembly as well.

Robustness and flexibility are important features for automated assembly equipment used in a JIT environment. In a serial-flow JIT assembly line, the failure of any piece of equipment will cause the entire production line to stop.

The robot workcell described below is designed to learn the location of all important points in the work envelope

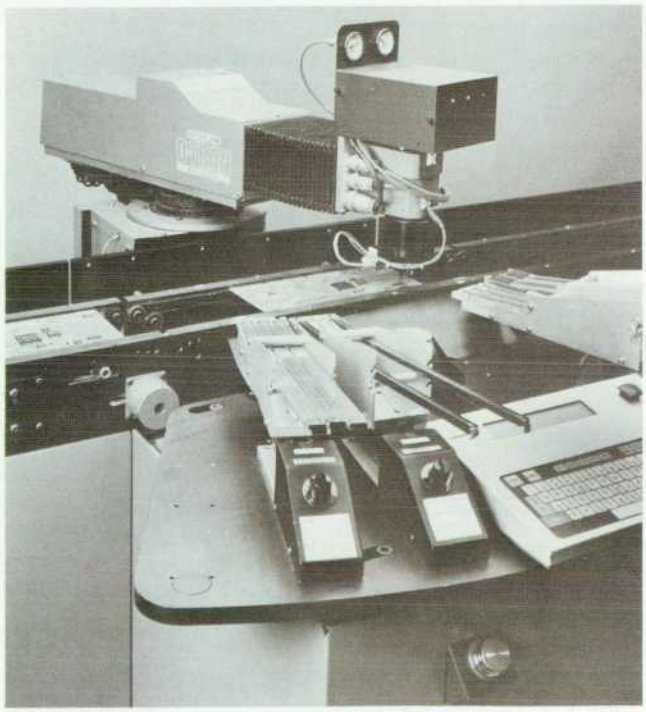


**Fig. 1.** *Robotic workcell used to build DeskJet printed circuit boards.*

## DeskJet Printer Design for Manufacturability

The need to minimize the DeskJet printer's time to market meant that a smooth production ramp was as critical to the introduction as meeting the other project checkpoints. Design for manufacturability was crucial to building quality products and meeting the required production rates.

**Product Design**

The focal point of the effort was defining the manufacturing processes. Once this was done, we defined the boundary conditions that the design team had to meet. We decided on primarily surface mount technology for printed circuit assembly, and sequential, single-plane assembly using a conveyor line for top-level assembly. This enabled us to write guidelines for the designers to use in designing the product. The manufacturing engineering team was assigned at the transition of the project from the investigation phase to the lab phase. Of course, it was also necessary for the designers to expand their horizons to consider the manufacturability along with the functionality of their designs. From the beginning of the prototype phase, we built all of the units on the final manufacturing line. This enabled us to debug the manufacturing processes in parallel with the product.

**Part Design**

We used the same tactics for part design, identifying part processes early and then designing to match these processes, assigning a procurement engineering team at the beginning of the lab phase, and using final part production processes from the first prototype on.

When we reached the production prototype phase, our manufacturing organization and our suppliers had built over 500 units and had found and corrected many of the typical start-up problems. We have met our production targets to date, and have satisfied the requirement to have ample units available for shipment at introduction.

*Don Harring*
Procurement Engineering Manager
Vancouver Division

using sensors mounted in the robot end effector. This feature has paid for itself many times over by greatly decreasing the reconfiguration time when adding or changing part feeder locations. A complete robot changeover has been accomplished in four hours with the aid of the self-teaching features incorporated into the robot software.

### Use of Automated Equipment in a JIT Environment

It should be pointed out that automated assembly and JIT manufacturing are two entirely separate ideas. Each can be implemented independently of the other. By no means does one imply the other. Some of the most successful JIT assembly lines employ little or no automated assembly equipment. Highly automated factories exist that operate strictly in a traditional batch-type production climate with large work-in-process inventory levels. A key step in the Vancouver Division's manufacturing strategy is to integrate automation into the JIT environment.

Automation for automation's sake usually results in automating waste and the production of scrap material. To avoid this, each automation project is considered on the basis of improving quality, lowering production costs, and eliminating difficult or tedious manual tasks.

The precision assembly tolerances of ±0.005 inch required for surface mount component placement makes this task difficult or unattainable in a manual workstation. A decision was made early in the robot workcell design to employ a machine vision system to ensure the placement accuracy required for zero-defect assembly. Quality levels have improved significantly since the implementation of the automated surface mount placement line.

### The Robot Workcell Hardware

A two-phase process is used to place surface mount components onto printed circuit boards. A high-speed pick-and-place machine places all components whose size and packaging are compatible with the machine parameters. The larger PLCC (plastic leaded chip carrier) and SOIC (small-outline integrated circuit) components, which do not fit the size and packaging requirements of the high-speed machine are placed by the robot workcell. Several components are available in packaging compatible with either the robot or the high-speed machine. These parts can be switched to either machine to balance the machine cycle times for changes in product mix.

The robot workcell (see Figs. 1 and 2) consists of a Seiko RT3000 robot and controller, a machine vision system, a printed circuit board conveyor system, electrical interconnect hardware, and control computers. The workcell accommodates up to 16 tube-type parts feeders. Each feeder has space for up to four tubes, dependent on the component size. One machine operator is able to run the entire surface mount placement and solder assembly line. The operator's main task is to place printed circuit boards screened with solder paste on the input conveyor of the high-speed placement machine. The operator is also responsible for filling the component feeders as they are emptied. All placement operations and the movement of printed circuit boards between machines occur automatically.

An adjustable-width printed circuit board conveyor system connects the two pieces of placement machinery. Boards are moved along the conveyor using antistatic belts which grip the outer edges of the boards. The belts are driven by step motors controlled by the Model 320 host computer. The workcell design will allow a second robot to be put in place as capacity and product mix needs dictate. Conveyor control and electrical interconnect for the second robot are already in place.

Pressure sensitive safety mats are used to guard the front and rear of the robot work envelope. These mats have proven effective in maintaining a safe zone around the robot while still allowing access and visibility. Facade plates mounted on both sides of the workcell guard against dust and enhance the aesthetics of the production environment. The facades are easily removed to allow access to electrical interconnect hardware.

### Automatic Transfer of Component Loading Data

Component placement coordinates are downloaded di-

rectly from the printed circuit design computer system to the controllers of the placement machines. This capability greatly simplifies prototype builds of new boards and revisions to existing boards. Prototype builds, which previously were done manually, can now be run through the standard placement assembly line with minimal impact on production throughput.

### Robot Operating Sequence

Boards arriving at the robot loading position of the conveyor are detected by a reflective sensor. The width of the loading section of the conveyor is narrowed to hold the board for component placement. This approach eliminates tooling pins and other associated tooling fixtures, but requires some extra software development to accommodate the lower-tolerance board fit in the clamped conveyor. A combination of vision and board graphics supporting the extra software overcomes the looser fit at a considerable savings in tooling costs.

The robot initializes the I/O lines and begins a search for the reflective "landing pad" located on the leading edge of each board. Printed circuit board design guidelines specify that all new boards will have a landing pad located at the same coordinate along the leading edge of the board. A reflective sensor in the robot end effector locates the reflective pad. Along with the coordinates of the clamped conveyor edges, the landing pad provides a reference point for the robot to locate critical board artwork details.

In the next step the reflective sensor is used to read a binary code incorporated into the board artwork, which tells the board type. The ID code dimensions and location are specified for all new boards. In most cases the component type and location coordinates can be called directly from the robot controller memory. When prototyping a new board, the robot will query the host computer for the file that contains component loading data.

Feeder locations and part parameters (number of leads, part height, etc.) reside in a data base in the robot controller memory. The component data base also contains vision system parameters necessary for the camera to identify and evaluate each part. The vision system will reject any part that does not have the proper number of leads. This feature has proven very successful in guarding against placing parts with bent or contaminated leads.

When the robot controller has identified the board location and type, the component loading sequence begins. The loading program was specifically developed so that the same program can load any board after the board has been identified by the robot. The board ID code points to a file that contains the loading information for each board. This file includes the number of components, the type of component, the feeder number, and the location of each component on the board.

Each part is picked from its feeder using a vacuum nozzle in the robot end effector. Each feeder is turned on for a fixed time before and after each pick. Feeder control is through the Model 320 host computer. Four robot input/output lines are used to send control signals to the host computer for each of up to 16 feeders. Each part is moved over an upward looking camera which takes a machine vision image of the part. The part centroid and rotation are calculated from the reflections of the lead tips. A screening algorithm is used to block out the center section of the part and avoid confusion between the part leads and the writing that appears on the bottom of the part.

While the vision system computer is processing the part offsets, the robot moves to the ideal placement location. The robot arm adjusts for the x, y, and theta (rotational)

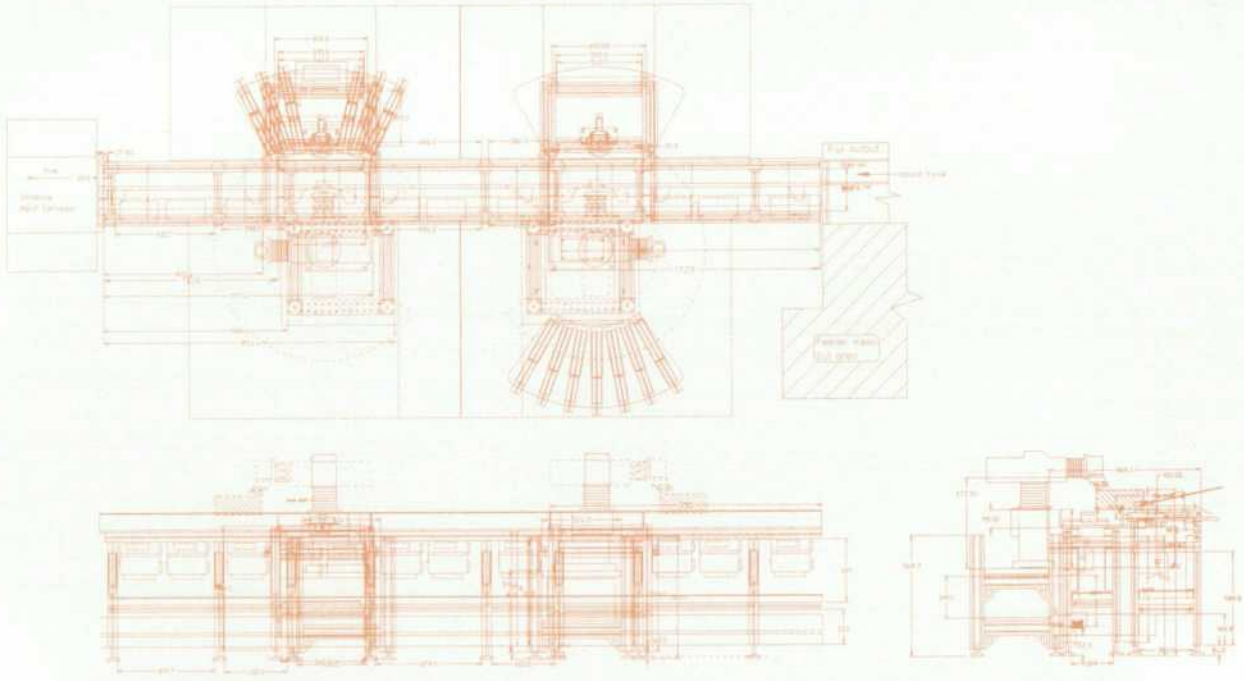


**Fig. 2.** *Layout drawing of the robotic workcell. The design was done on an HP ME Series 10 CAD system. This drawing was printed on an HP DeskJet printer.*

## Fabricated Parts Tooling Plan

The fabricated parts tooling plan for the HP DeskJet printer was designed to minimize tool development time and thereby decrease the time to market. Elements of the plan were:

- Prioritizing part design by tooling lead time so as to complete all of the fabricated parts on the same date
- Initiating tool quotations using preliminary drawings
- Using overnight mail service to convey prints to suppliers
- Compressing quotation cycles to half the normally allowed time by working with our suppliers to queue quotations
- Completing all the paper work necessary to order the tooling before receipt of quotations so tools could be ordered within a day of receipt
- Using soft tooling for noncritical parts and hardenable grades of tool steel for others, which allowed for easier modification
- Using a manufacturer of molded parts tooling for our case parts and chassis with a lead time half that of the normal suppliers
- Minimizing the use of multicavity tooling on plastic parts, reducing lead time and time to rework tools
- Developing multiple manufacturing processes in parallel on high-risk manufacturing parts, with one of the processes ultimately chosen for its certainty of producing acceptable parts
- Using multistage stampings rather than progressive dies, thus both decreasing tool development time and allowing for the addition of extra operations
- Manufacturing both first articles (the first tool test cycle parts) and prototype build parts at the same time
- Using suppliers to perform first article inspections instead of doing them in-house
- Achieving measurement correlation at an early phase to decrease part measurement discrepancies and allow us to use statistical measurement data supplied by the part manufacturers to determine part quality.

*Jeff Ward*
Manufacturing Engineer
Vancouver Division

offsets specified by the vision system and presses the part into the solder paste. When all parts have been placed, the robot signals the conveyor control program to unclamp the board and move the board onto the exit conveyor. The fully populated boards exit directly from the robot workcell into the paste cure oven.

### Workcell Design

An HP ME Series 10 computer-aided design (CAD) system was a key factor in the successful completion of the workcell design project. Design proceeded from a rough layout drawing, and each piece was added to the ME Series 10 drawing as the layout proceeded. Detailed drawings required for the fabrication of each piece of custom hardware were generated from the layout drawing after the layout was frozen. The importance of having a layout drawing that includes both the electrical and mechanical hardware cannot be overemphasized. The additional accuracy of the CAD tools leads to significant quality improvements and time savings in any complex design project. The integration of electrical interconnects and safety systems is greatly enhanced by including these features in a single layout drawing.

### Acknowledgments

The success of this project would not have been possible without the contributions of many people. Special thanks to Marc Hartquist, Britt Freund, Ken Wade, and Rick Hughes for their participation in the workcell development team.

# CIM and Machine Vision in the Production of Thermal Inkjet Printheads

*Machine vision systems for DeskJet printhead production range from open-loop go/no-go systems to process verification systems to completely integrated process control systems.*

**by Mark C. Huth, Robert A. Conder, Gregg P. Ferry, Brian L. Helterline, Robert F. Aman, and Timothy S. Hubley**

MACHINE VISION was first introduced to the HP inkjet production process in the original ThinkJet printhead assembly area. It was recognized then that some objective measure of the quality of each printhead was needed. With the installation of the first print quality evaluation system came the recognition that many other processes could be monitored as easily. This led to machine vision applications that inspected for leaks, evaluated the quality of the special paper required for the first ThinkJet printhead, checked alignment of parts, inspected incoming material, and verified glue pattern completeness.

The experience gained in these first applications proved that much useful information could be retrieved by visual inspections. The original concept of simply rejecting parts that didn't pass inspection was replaced by an inspection scheme that shut down the line if more than three printheads were found bad. An extension of this process monitoring, collecting print quality data for each printhead, began to close the loop between inspection and process control. Tying this print quality data to the wafer and orifice plate lot numbers allowed the process engineers upstream from the final assembly area to see the effects of varying parameters. Without machine vision and the automatic link to the data collection system, this wouldn't have been feasible. Operator inspection and entry from a keyboard would not only have been cumbersome, but also prone to the subjective differences between operators in measuring the darkness of the print or the sharpness of a line. By interconnecting the vision system with the rest of the computer systems, another link was forged in the overall computer integrated manufacturing (CIM) strategy.

Another category of vision system applications is the complete integration of the measurement data into the process. Applications in this category include the alignment systems used in placing the head assembly on the printhead body, aligning the interconnect circuit to the head assembly, and aligning the orifice plate to the substrate. These three vision system applications are discussed in more detail later as the head attach machine, the TAB (tape automated bonding) attach machine, and the orifice attach machine.

The process involvement of vision systems varies, as shown in Fig. 1. Vision system applications have evolved from totally open-loop go/no-go test systems, through process verification type systems, to completely integrated process control applications. Examples of all of these categories are implemented throughout the DeskJet printhead production area.

### Preassembly TAB Circuit Inspection

On the left side of Fig. 1 is the go/no-go type inspection of incoming material for the TAB preparation machine. Here the only process monitoring function is to keep a count of how many bad parts the system has found.

Briefly, the task of the TAB prep machine is to present good TAB circuits to the TAB attach station. TAB circuits are the flexible interconnect circuits used on the DeskJet print cartridge. The machine can be divided into three parts: mechanical, programmable logic controller, and vision engine. The TAB circuits come from the vendor in tape form on a reel and need to be blanked (separated).

The primary objective of the vision engine is to check for bent beams with a tolerance of about 40 micrometers. These beams are TAB bonded to the substrate of the thermal inkjet head. A major challenge for this vision project was determining the camera configuration. The Nyquist criterion says that for a minimum resolution of 40 $\mu$m, one would

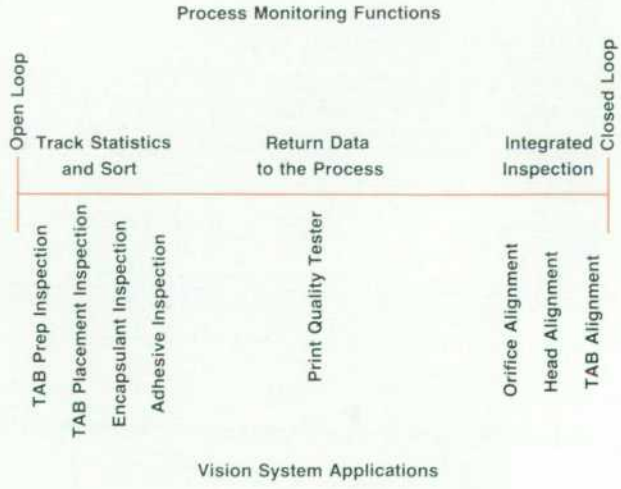



**Fig. 1.** *Machine vision system applications have varying degrees of process involvement, shown here increasing from left to right.*

# Whole Wafer Assembly of Thermal Inkjet Printheads

The challenges of assembling the new thermal inkjet pens have become greater as printhead design architectures and alignment resolutions have become finer. In addition, customer demands require high-volume capacity and low-cost products. For DeskJet printhead manufacturing, meeting these challenges required new assembly techniques and tools. As a result of close work between manufacturing engineering and R&D, allowances for new assembly techniques were made in the printhead design, without degrading printhead performance. The most significant new assembly technique is called whole wafer assembly.

The materials used to build an inkjet printhead include a thin-film substrate and a metal orifice plate. The substrates are processed on a silicon wafer and the orifice plates are manufactured in sheet form and individual parts are then excised from the sheet. The basic requirements for assembly of the printhead are alignment and bonding of the orifice plate to the thin-film substrate. Before whole wafer assembly, each substrate was diced out of its parent wafer before the orifice plate was attached. The whole wafer assembly technique bypasses the dicing operation and mounts the orifice plate to each substrate before dicing the wafer. The method simplifies material handling and part orientation, since all the substrates are on one wafer. Whole wafer assembly and machine vision have made assembly of the new inkjet pens a fully automated process.

### Product Design

The Deskjet printhead design is schematically fairly simple. The printhead is made up of a thermal printhead, a flex or TAB (tape automated bonding) circuit, and an ink container or printhead body. The thermal printhead is approximately 5 mm by 7 mm by 0.5 mm thick and is made from a silicon wafer that has been plated and photochemically etched to produce 50 individual resistors used in the generation of the ink droplets. An orifice plate with 50 nozzles to match the resistor locations is accurately aligned to the substrate and bonded in place. This printhead assembly is then aligned and bonded to the printhead body. Tight assembly tolerances are necessary to meet the print quality requirements. Furthermore, assembly costs need to be kept at a minimum, requiring high-volume automated assembly.

### Assembly System Description

Two different assembly systems were developed around the whole wafer process. These are named the orifice attach machine and the head attach machine. The orifice attach machine aligns and bonds an orifice plate to each die location on the wafer. After the wafer is fully populated with orifice plates it is sawn into individual printheads. Then the head attach machine aligns and bonds each printhead to a pen body.

Alignments of the respective printhead parts are provided by high-precision positioning tables built into the assembly systems. These positioning tables provide the required assembly accuracies along with part orientation and transfer, alignment adjustments, and bonding motions. Each system also includes a vision system which provides the means to determine component part misalignments. The misalignment measurements are sent to the system controller, which adjusts the position tables to align the two component parts. Substrates are supplied to the system in whole wafer form and all assembly of the printheads is done before the wafer is sawn apart. The orifice plates are fabricated initially in sheet form. The sheet is mounted to a low-tack tape and stretched over a metal film frame. The sheet is then broken to separate the individual orifice plates. The frame provides easy handling of the orifice plates and the mounting tape holds the individual orifice plates in their original configuration. Fig. 1 shows orifice plates being removed from the low-tack mounting tape in the orifice attach machine. The sheet is positioned to allow for the automatic picking of each orifice plate by a vacuum chuck as shown. The camera is used to determine the location of the

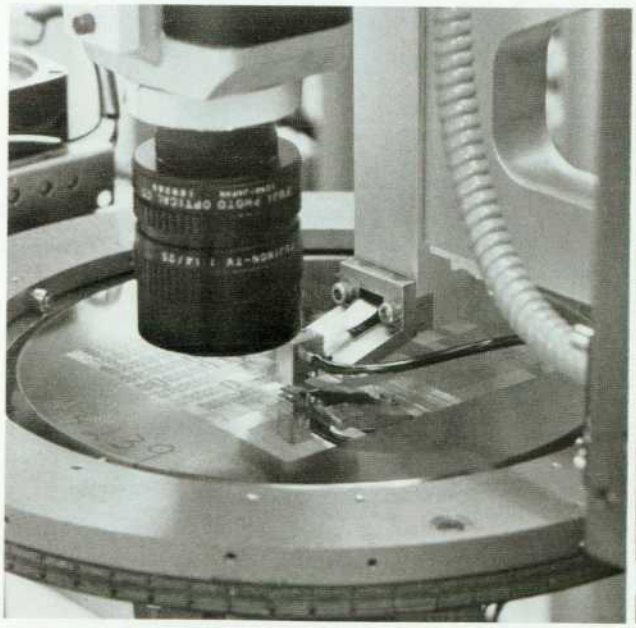


**Fig. 1.** *Orifice plates being removed from the low-tack mounting tape in the orifice attach machine.*

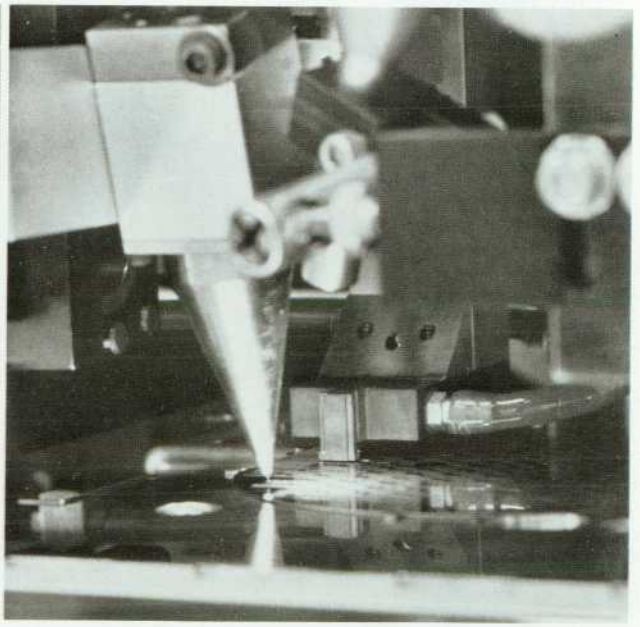


**Fig. 2.** *An orifice plate being optically aligned to a whole wafer in the orifice attach machine.*

orifice sheet. Fig. 2 shows an orifice plate held above the wafer and being aligned to a substrate of the wafer. In this position, the vision system can determine the relative misalignment of the two parts and make adjustments to bring them into the proper alignment for bonding.

The individual substrates of the wafer and the orifice plates of the sheet come to the system having been previously inspected and mapped. The material maps tell the system which parts are good and may be selected for assembly. Another advantage of the whole wafer, whole sheet process is the ease of loading material into the systems. Since it is crucial to know where all the individual parts are located for assembly, it is an advantage to handle these parts in their whole states. The wafer's substrates are still in whole wafer form, so their locations are known to a fraction of a micrometer. The orifice plate locations are also known. This feature allows the system to know where all of the individual parts are located within tolerances sufficient for the system to run automatically.

The basic operation of the orifice attach machine is to pick a good orifice plate off the mounting tape, hold it over a good substrate of the wafer, inspect the relative offsets of the two parts optically, adjust for this misalignment, and bond the plate to the substrate. Operation continues in this fashion until either the orifice plates have been depleted or the substrate wafer has been fully populated with plates. After the wafer is fully populated with orifice plates, the wafer is mounted to low-tack tape and a metal frame and then sawn into individual head assemblies.

The film frame with all of the head assemblies still intact is then loaded into the head attach machine. Fig. 3 shows the sawn head assemblies loaded into the head attach machine waiting to be picked off by a vacuum chuck in the same manner as the orifice plates were picked off by the orifice attach machine. The picked head assemblies are subsequently aligned and bonded to an inkjet pen body. With the head assemblies loaded into the system in this fashion, the head attach machine also realizes all of the advantages of the whole wafer process, including easy material handling, part cleanliness, mapping capabilities, and so on.

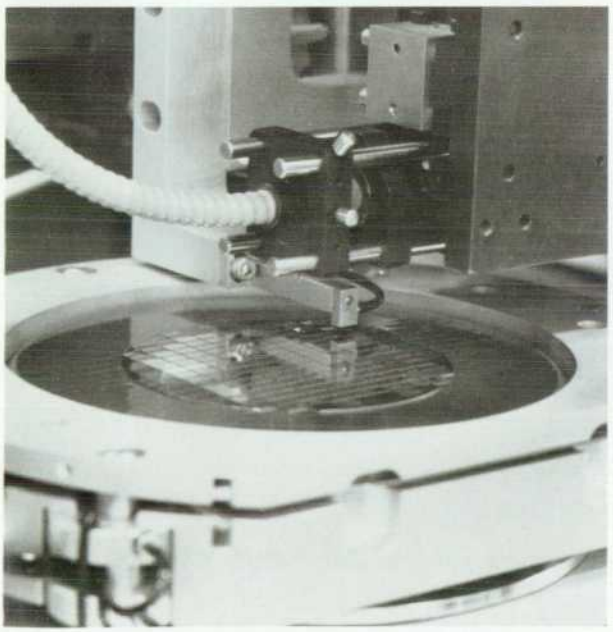


**Fig. 3.** *Inkjet printheads being removed from the low-tack mounting tape in the head attach machine.*

### Vision System

To meet the assembly tolerances and volumes required, the human operator needed to be taken off the assembly line. The human eye, even if aided with a microscope, cannot reliably align the orifice plates to the substrates. Also, because of processing limitations, the assembly tolerances cannot be met by mechanical alignment means. The demand for high-resolution alignment, fast cycle times, and high reliability, required the use of a vision system. The product design includes optical targets processed onto the orifice plates and the substrates. The vision system hardware and algorithms determine the offsets between the targets of the two parts and report these offsets to the assembly equipment. The assembly equipment makes the appropriate part adjustments to align the two parts and then bonds them together. After the parts are bonded together, the vision system inspects the final alignment and updates the wafer map with pass/fail data which is used in subsequent processes.

An additional benefit of the vision system is its multirole capability. It is also used as a process monitor for adhesive dispensing. The adhesive must be dispensed in precise locations and volumes onto the substrate. The vision system monitors the location and volume of the dispensed adhesive during the printhead assembly process.

### Computer Integrated Manufacturing

A significant advantage of whole wafer assembly is its compatibility with computer integrated manufacturing (CIM). All materials destined for inkjet printhead assembly are inspected during their respective fabrication steps and this data is attached to individual parts to generate a map of good and bad parts. CIM provides the assembly systems with these maps, and carries updated map files to the next assembly systems. Since all substrate and orifice plate locations are mapped, this information remains intact as long as the wafer or orifice sheet remains unseparated. The chance for errors is reduced because no further inspections, separate sorting, or paperwork are required. In addition, since the individual inkjet printheads are not removed from the film frames, the map data is updated from the orifice attach system files and the head attach system uses the updated map.

### Conclusions

The new assembly technique of attaching mechanical parts to a whole wafer before dicing has solved many manufacturing problems. It has been possible to use some off-the-shelf hardware because of the similarities of this process to those found in the semiconductor industry. Material handling problems have been kept to a minimum by handling only whole wafers and sheets of orifice plates. Part quality and cleanliness have improved over manual alignment because of the simplified parts handling technique. Cycle times have also been reduced because parts handling is kept to a minimum. Finally, a major advantage of whole wafer assembly is the ability to use CIM as an assembly tool and avoid all of the problems associated with manual inspection, sorting, and accounting. The whole wafer assembly technique, when coupled with precise positioning tables, CIM, and a vision system, has proven to be a very important factor in the successful manufacturing of the DeskJet printhead.

*Bob Aman*
Manufacturing Engineer
Inkjet Components Operation

need to sample every 20 μm. The area to be inspected is 7700 by 7100 μm. The part can be presented to the camera within a ±250-μm tolerance. Because the lens aberrations are greater at the periphery of the lens, twenty percent is added to the area to be inspected. The area is then 9000 by 8400 μm. This means that the camera resolution should be 450 by 420 pixels. At the time of this project, cameras such as this were very expensive, so an alternative was required.

The area to be inspected is two strips on the top and bottom edges of the area described above. These areas measure 7700 by 1300 micrometers each. If two cameras were used, they would need a resolution of 450 by 75 pixels, still difficult to obtain. If the area were split once more, the area for one camera would be 3900 by 1300 μm, requiring a camera resolution of 225 by 75 pixels. At this resolution, a 256-by-240-pixel system with four cameras can be used. The fields of view for one and four cameras are shown in Fig. 2.

The last vision challenge was to put the four cameras into an area measuring 7700 by 7100 micrometers! One possible solution is to put the four cameras at four different places along the tape. The solution chosen was to use two cameras and move the tape a short distance to present the second half of the part to the cameras. The pictures are not "reassembled," but each picture is analyzed and only information about the status of that particular corner is passed along in a part tracking routine within the machine.

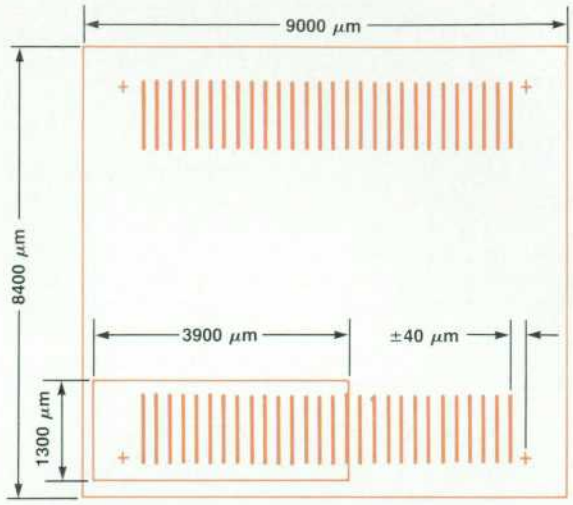



**Fig. 2.** *Overall and single-camera (lower left) fields of view for the TAB prep machine.*

### Postwrap TAB Inspection

The last process through which the TAB circuit goes is wrapping onto the back wall of the printhead body, where it is staked into place. The centers of the pads on the TAB circuit are required to be within 250 μm of a location relative to the printhead body fiducial marks. The problem comes when the part is presented to the camera with a tolerance of ±250 μm. Only a perfect part would pass any type of inspection! If the printhead body fiducial marks could be seen at the same time as a pad, then the location of the pad could be determined. This would be possible only through a complicated calibration procedure which could easily drift.

Through a mechanical fixture, the fiducial marks can be translated to a location near a pad. A translated fiducial mark can be located within ±50 μm of the ideal pad location, but still only within ±250 μm relative to the field of view of the camera. A hole in a plate much like a bombsight was developed. The hole was larger than the pad plus the tolerance. The idea was to find the center of the pad relative to the center of the bombsight. Unfortunately, insufficient light came through the bombsight to illuminate the pad, so the bombsight gave birth to the idea of a fiducial mark on a piece of glass. This was found to be the appropriate solution.

### Encapsulant Placement Inspection

To the right of the open-loop tester in Fig. 1 are shown the inspections for adhesive and encapsulant placement. These are tied a little more into the process control, since multiple failures shut down the line until an operator verifies and corrects the problem.

An encapsulant is used to cover the TAB circuit beams and exposed substrate. Conflicting requirements cause its placement to be critical. For a vision system these tolerances are not difficult to check.

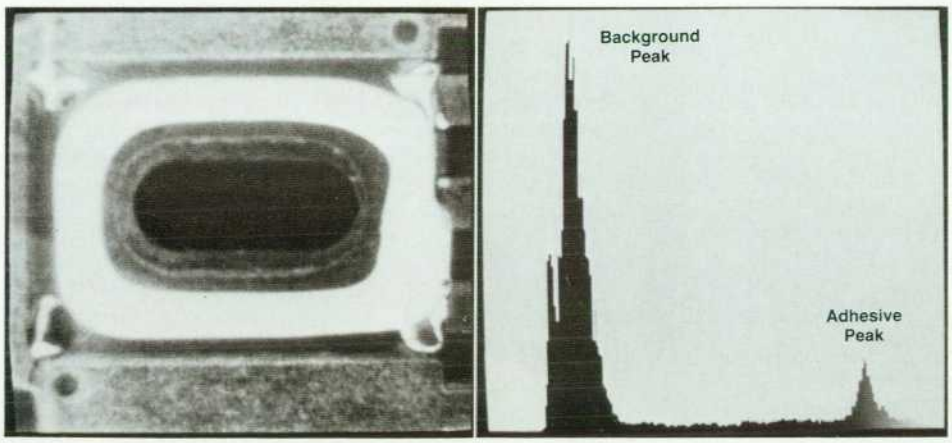



**Fig. 3.** *A machine view of adhesive on the pen body and the corresponding gray scale histogram.*

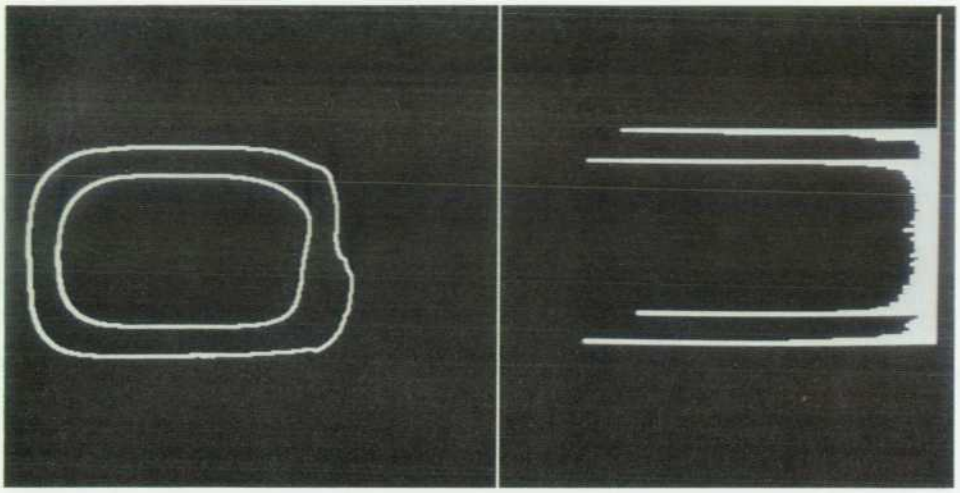


**Fig. 4.** *Edge-enhanced adhesive image and horizontal profile.*

The first encapsulant used was clear, having almost the refractive index of air. This meant that the encapsulant could not be seen by man or machine. Various dyes were added to the encapsulant to improve its contrast with the surrounding part. While the color contrast was sufficient for a human to see it, the black and white vision system required luminous contrast.

A fluorescing dye was found to be most successful. Fluorescence has the advantage of emitting light that is coming only from the object of interest. The ultraviolet light that causes the dye to fluoresce is not visible to the camera. The part is processed in a dark chamber to enhance the encapsulant even more.

Once the encapsulant was made to fluoresce, the vision algorithms became fairly straightforward. They follow very closely the algorithms used in the adhesive inspection system.

### Adhesive Inspection

Structural and tack adhesives are used on the printhead to bond the head assembly to the plastic body. A vision system is used on the assembly line to verify the position, shape, and quantity of the structural adhesive bead and the presence of the tack adhesive on the plastic body before a head assembly is attached.

Acquiring the image of the structural adhesive is not difficult. It is light-colored and opaque, and the contrast with the black printhead body is good. This good contrast allows use of an adaptive threshold technique, which has been found useful in many applications. This technique involves using a histogram, which is a distribution plot of the number of pixels at each discrete gray level within the image. The histogram for an image of the structural adhesive on the black plastic body has a major peak at the lower (dark) gray levels and a major peak at the upper (light) gray levels (see Fig. 3). Statistical methods are used to find the valley between these two major peaks, and this valley represents the optimal threshold for binarization. This threshold technique is a very powerful tool because of its ability to adapt to changing lighting conditions.

Once the image is segmented by binarization, the area of the structural bead is calculated and compared with predetermined limits. If the area is acceptable, completeness of the rectangular pattern is determined using run-length encoding and blob labeling techniques to assure that the pattern forms a closed loop.

The final test of the structural adhesive involves an interesting use of filtering to calculate the centerline of each of the four legs of the rectangular pattern. A gradient filter operation is performed on the binarized image, and in the

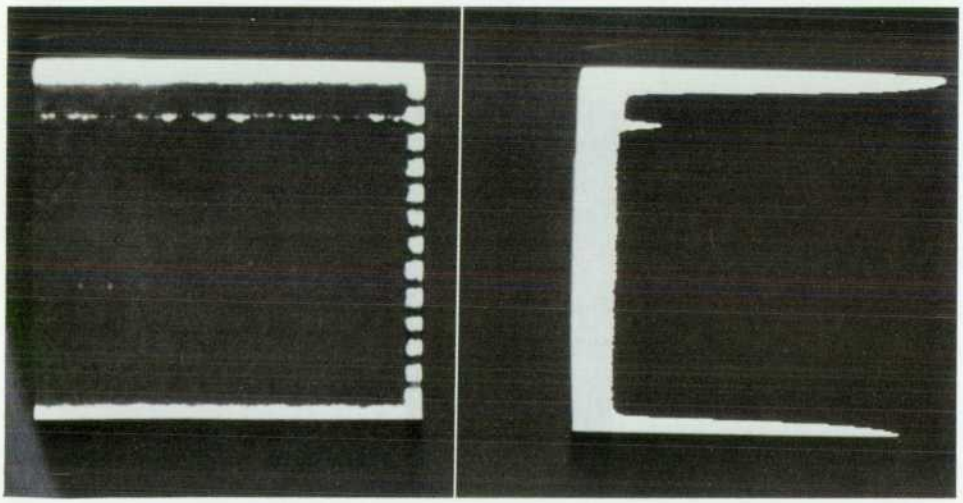


**Fig. 5.** *100% area fill test with a weak nozzle and the computed vertical profile showing the weak nozzle as a small peak.*

resulting image, the edges of the adhesive pattern are highlighted (Fig. 4). A profile, which is a distribution plot of the sum of the gray levels of all pixels in any row or column, is computed for the edge-enhanced image, resulting in distributions containing peaks that correspond to the locations of the adhesive boundaries. These peak locations are used to calculate the centerline and width of each leg of the adhesive bead.

The adhesive inspection process goes one step beyond simple part inspection by incorporating some of the process control into the station. A running yield is calculated from the results of the last ten parts inspected, and if that yield drops below an acceptable level, the line is stopped and an operator is alerted. The number of occurrences of each failure mode is displayed at the inspection station so that appropriate action can be taken by the operator to bring the dispensing process back into control.

### Final Print Quality Inspection

The last open-loop inspection to be discussed is perhaps the most important application of machine vision on the production line. The quality of the printhead is judged by objective measures that can be traced back to attributes defined in the original specifications of the printhead design. However, by the very nature of this inspection, it can only tell how well the printhead performs after the fact. The loop was closed by the installation of a data collection scheme that is used to track the effect of a change in system parameters. For more information see "Production Print Quality Evaluation of the DeskJet Printhead," below.

The primary objective of the print quality inspection station is to verify the health of every nozzle of the printhead. This means sufficient drop volume to sustain complete area fill and accurate dot placement for sharp edge acuity.

The most easily detectable print quality defect is unevenness or banding over a character. Its causes range from slightly varying drop volumes to completely missing nozzles. To detect the full range of banding defects, the vision engine makes extensive use of profiling techniques combined with peak detection algorithms to locate defects relative to the gray scale level of the image. This reduces the

## Production Print Quality Evaluation of the DeskJet Printhead

The DeskJet printhead's drop ejection capability is evaluated at the end of the printhead assembly process. This evaluation process has two main goals: first, to screen out defects, and second, to provide process feedback.

### Production Screen

Printheads shipped from the production line, when installed in the printer, must meet or exceed customers' expectations for print quality. A finely honed print quality evaluation process has been developed to ensure that printheads meet their minimum acceptance levels. Print quality can easily and repeatably be evaluated by trained operators and/or machine vision.

A wealth of experience was gained from the ThinkJet print quality evaluation effort which evolved into a machine vision print quality tester on the ThinkJet printhead assembly machine.[1] Three major lessons learned from this endeavor were: first, the obvious, that orifice presence had to be verified, second, that area fill (blackout) had to be evaluated, and third, that special test patterns were needed to highlight specific printhead process fail conditions.

R&D had established a printhead external reference specification (ERS), which outlined the expected operating window of the printhead. Early print quality evaluation techniques concentrated on specific metrics to verify the ERS. Parameters such as drop volume and velocity, dot diameter and placement, and spray all had their own test procedures which gave quantitative measures for experiments designed to tune the printhead fabrication process. However, for the production environment, R&D's efforts to measure print quality needed to be generalized to achieve production cycle times and produce measures that were less removed from customer expectations.

The first production print quality testers were modified HP 7550A Plotters with special print engines. These simulated printers allowed us to develop test characters and exhaustively exercise the printhead. Our first tests involved five pages of stress printing at low and high frequencies and duty cycles. Each page had three sets of test character patterns which were evaluated and recorded. All defective orifices were noted and compiled for the printhead lot.

### Quickie Pattern Development

The Quickie pattern of test characters (Fig. 1) was developed to evaluate a printhead in an open-loop production environment. It consists of an initial 25% area fill pattern that serves as a startup burst to clear any viscous plugging, followed by a series of test characters: 10 by 5 stairstep, blackout, vertical lines, miscellaneous patterns for development, 25% area fill, 50% area fill, and finally odd-even stairstep. The original R&D print quality metrics evolved into the present audit tests which run on a sample of printheads from each lot.

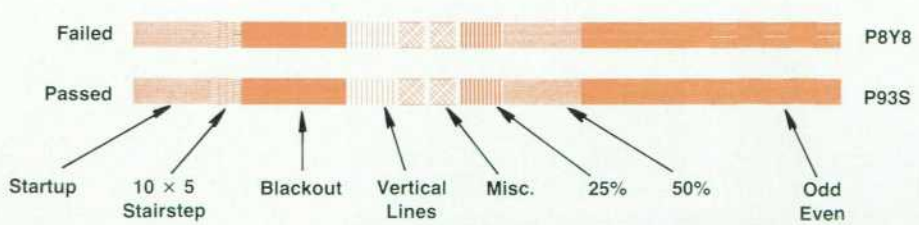



**Fig. 1.** *Sample Quickie patterns for DeskJet printhead print quality evaluation.*

The order of the test characters turned out to be important, especially when evaluating printheads that had marginal drop volume.

### Evaluation Hierarchy

The Quickie pattern provides the framework for developing an evaluation hierarchy. This hierarchy provides both a standard terminology and common evaluation criteria to assign one fail mode per printhead. Table I lists the hierarchy of test character evaluation and the associated fail modes. Two additional benefits from a hierarchical approach are: first, that a fail mode not only describes itself but also implies that all previous modes were passed, and second, that the consistent terminology allows tracking of failures through any possible yield improvement rework or reclaim processes.

**Table I**
**Test Character and Fail Mode Hierarchy**

| Hierarchy | Test Character | Fail Mode |
|---|---|---|
| 1 | Blackout | Deprime |
| 2 | Blackout | Vertical registration |
| 3 | Blackout | White bar |
| 4 | 10 × 5 Stairstep | Not present |
| 5 | 10 × 5 Stairstep | Extra orifice |
| 6 | 10 × 5 Stairstep | Misaimed |
| 7 | 10 × 5 Stairstep | Weak orifice |
| 8 | 10 × 5 Stairstep | Jack frost South |
| 9 | 10 × 5 Stairstep | Jack frost North |
| 10 | 10 × 5 Stairstep | Starvation |
| 11 | Vertical Lines | Trajectory |
| 12 | Vertical Lines | Satellites |
| 13 | Vertical Lines | Spray |

### Machine Vision Print Quality Testers

The Quickie pattern evaluation hierarchy was extended to the machine vision print quality testers. These testers employ an X-Y table to move test media under a printing printhead and then register the test characters under a camera for image capture and analysis. Strict adherence to the evaluation hierarchy allows simpler correlation and acceptance limit setting. The machine vision algorithms are discussed in the accompanying article.

### Process Feedback

The functionality of a printhead is monitored to allow feedback to the fabrication and assembly process. To maintain assembly process control, fail mode data for each printhead (identified by bar code) is collected by the machine vision print quality evaluator.

### Process Monitoring

Certain fail modes are tracked. When an out-of-control situation is detected, the operator is notified. Presently monitored are: electrical opens (this provides feedback on the TAB bond process or the quality of the tester interconnect), deprimes (feedback on the foam insertion and ink fill), and vertical registration (feedback on the printhead head alignment to the plastic body).

### Reference

1. R. Conder, "Three Cameras Look at Inkjet Pen Production," Manufacturing Systems, August 1987.

*Timothy S. Hubley*
Manufacturing Engineer
Inkjet Components Operation

sensitivity to variations in lighting and contrast across the field of view.

If profiling is performed parallel to any lighting trends in the image, the trends are removed. Once the profile (either horizontal or vertical) of the image has been determined, peaks can easily be located which represent sharp and significant changes in the gray scale range over the row or column. For example, given a 100% area fill character in the field of view as shown on the left in Fig. 5, the image is analyzed. Since the printhead moves from left to right, any banding will appear as a horizontal white gap in the blackout. The vertical profile of the image seen on the right in Fig. 5 is the summation over all columns for each row. Using this image, it is easily observed that the nozzle that is not firing creates a very large peak that can be detected as a failure. The height of this peak compared to the highest edges of the profile also reveals the whiteness of the band relative to the white paper background.

A status code for every printhead is placed in the CIM data base. Additional process information can be provided for monitoring and calibration.

### Alignment Machines

In the development of the DeskJet printhead manufacturing processes, three critical alignment processes were identified as being best accomplished by integrating machine vision into a closed-loop alignment process. The orifice attach machine, head attach machine, and TAB attach machine, contain a system controller that uses the vision processor as a tool to gauge the relative alignment of two parts. Then the parts are moved into better alignment before being tacked together. The design of the machines also allows for an inspection of the parts after they are assembled. The inspection does not affect the assembly cycle time because the vision inspection processing occurs in parallel with the transfer of new parts into the alignment area. For a more detailed report of the mechanics of these machines see "Whole Wafer Assembly of Thermal Inkjet Printheads," page 92.

### Orifice Attach Machine

The function of the orifice attach machine is to align and tack a singulated orifice plate onto a substrate that is still in wafer form. The process involves picking an orifice plate and transferring it to an assembly position directly above the substrate. The vision system then determines the misalignment between the two parts and the tables beneath the wafer move the substrate into better alignment with the orifice plate.

A major challenge in the development of the orifice attach processes was the design of the optics system to acquire the images of the alignment targets. The desire to resolve the location of the targets within several micrometers with a 256-by-256-pixel vision system dictated the use

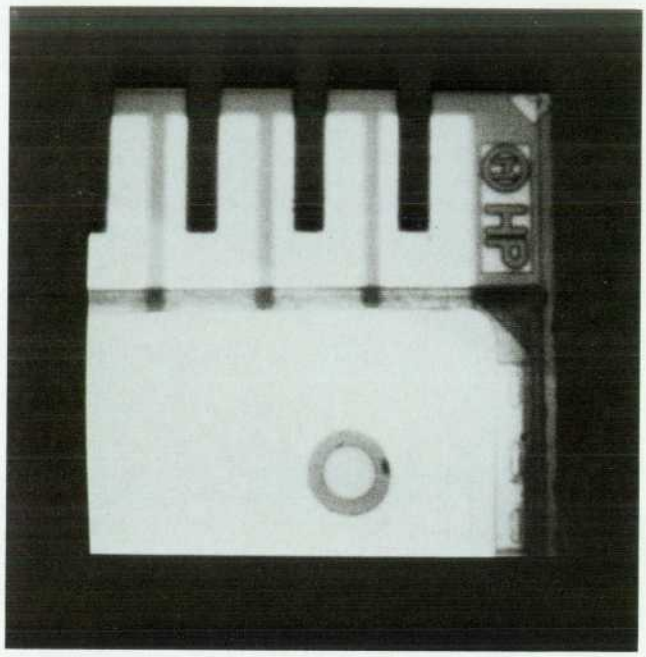


**Fig. 6.** *TAB attach machine field of view.*

of a unique field of view for each set of alignment targets. Locating the optical axis of one field of view within one quarter of an inch of another (the part target spacing) was a design challenge in itself. The conflicting requirements of high resolution and relatively large depth of field, combined with the need for axial lighting and a large front focal length to satisfy mechanical mounting constraints, required a custom lens set. The same lens design and mounting configuration are used at different magnifications for the head and TAB attach machines.

The vision process for this application involves an interesting variation of the adaptive thresholding techniques described in the adhesive inspection application. A histogram of the gray level image is again used as a tool to help determine the best threshold for binarization, but there are now targets with a specific geometry. The targets form an image that looks like a dark ring on a light background. Since the area of the ring is constant, the number of pixels in each column of the histogram starting with the dark end can be summed until the nominal area value is reached. The gray level value at this point is the optimal threshold for binarization.

The feature extraction portion of the process is relatively straightforward and is accomplished by calculating the centroid of the nearly concentric targets. Run length encoding and blob labeling techniques are used to segment and sort the targets after binarization.

### Head Attach Machine

The function of the head attach machine is to align and tack a substrate/orifice-plate assembly onto the plastic body. The process involves picking a head assembly off a film frame and transferring it to an assembly position directly above the plastic body. This process is unusual because the vision system is used to gauge the position of the head assembly only. Alignment targets molded into the plastic body can be discerned by human operators during a manual alignment process, but do not provide enough contrast to be reliably located using a vision system. Instead, a mechanical sensing method was chosen to determine the location of the plastic body. The vision algorithms for finding the position of the head assembly are identical to those used in the orifice attach machine. The machine controller calculates the relative offset and moves the parts into alignment.

### TAB Attach Machine

The TAB attach machine is an automated alignment system that aligns and tacks the TAB circuit to a head/body assembly. The system is integrated into the assembly line, handling two printhead bodies per pallet and receiving tested TAB circuits directly from the TAB prep machine.

The vision process involves determining the relative alignment between the substrate targets and the TAB circuit lead tips adjacent to these targets as shown in Fig. 6. The first step is to build a window around the nominal location of the substrate target. The algorithm uses adaptive threshold, binarization, and segmentation to determine the centroid. Based on this centroid location, a second window is built over the original image in the pad area, and a second binary threshold is determined using a histogram, which optimizes the contrast between the TAB leads and the pads. Binarization, blob sorting, and moments calculations within this window are used to determine the location of the desired TAB circuit lead tip.

Once both the lead tip position and the targets are found, the table moves to correct any misalignments. Just before the part moves out of the station, a final picture is snapped. This takes 1/30 second. As the part leaves the station the vision system calculates the final true alignment and stores the information for future reference. In this instance, a closed-loop inspection is used to get the part into alignment and the CIM connection is made from the final inspection data.

### Conclusions

The use of vision systems in the assembly of the DeskJet printhead has had many benefits. Verification of the quality level of the printheads produced is probably the most understandable. In addition, the process verification and part inspections have improved the quality of the printhead. Improvements in the process are only possible with the data to show where the problems are or how well a process has been performed. The TAB prep, TAB placement, encapsulant, and adhesive inspection systems satisfy the first level of an open-loop system.

Print quality inspection is on a level that begins to close the feedback loop by connecting data paths in the CIM format. The three attachment machines are tied even tighter in the CIM architecture using immediate feedback to direct the placement of parts. All levels of machine vision applications have shown their worth in helping to produce a high-quality printhead.

### Acknowledgments

Thanks to Larry Hubby of HP Laboratories, who served as optics consultant and designer.

# Economical, High-Performance Optical Encoders

*These high-resolution optical encoders are inexpensive and easy to install, making closed-loop motion control feasible in high-volume, extremely cost-sensitive applications.*

**by Howard C. Epstein, Mark G. Leonard, and Robert Nicol**

A HIGH-RESOLUTION, HIGH-SPEED printer like the HP DeskJet printer[1] needs an electronic ruler to know where it is on the page. For this function, the DeskJet printer uses a separately available HP product, the HEDS-9000 Shaft Encoder Module (Fig. 1). The version of this two-channel position sensor in the DeskJet printer provides 2000 marks (500 cycles) per revolution and gives direction information. The module facilitates a large reduction in the cost of servo motion control and is one reason the DeskJet printer is able to provide high-quality, speedy printing at a low price. Special geometries in the emitter/detector system and a "designed for manufacturability" architecture are key to the high performance and low cost of this transmissive optical encoder.

This article describes key elements of the module's design, manufacturing strategies, and performance. The box on page 100 presents the "Basics of Optical Incremental Encoders." The box on page 105 discusses how the module was made into a dust-resistant enclosed encoder with a self-contained code wheel, the HEDS-5500. A convenient gap-setting system allows quick assembly of the HEDS-5500 on a customer's motor with no special tools.

### Background

An earlier HP shaft encoder, the HEDS-5000, was reported on in October 1981.[2] The emitter/detector modules discussed in that article formed the basis for a family of encoding products that included a 1000-cycle-per-revolution encoder with index pulse and a panel-mounted digital potentiometer.

Building on the base of the earlier technology was a significant advantage in the HEDS-9000 design. Several veterans of the previous development led the product definition stage for the new module. The team also learned from a disciplined process of talking to users and potential users of encoders. Most engineers at the end of a design project have said to themselves, "I know what I would like to try next time." This team had their chance.

### Market Research

Customers were generally satisfied with the performance and reliability of the HEDS-5000 encoder kits, but several producers of office automation products said they needed a revolutionary change in price and assembly methods. The HP ThinkJet printer, for example, uses an open-loop step motor for driving the print carriage. A closed-loop system would have offered increased speed and graphics quality, but the added cost of an encoder (about the cost of the drive motor) was too great for the target market.

The purchase price of an encoder is not the only concern. Encoder characteristics can determine motor selection criteria. An encoder that can operate on a hot motor with large shaft play allows the use of significantly lower-cost motors. Economics often favor small motors that can reach temperatures up to 100°C. As one customer put it, "If I'm not running a motor hot, I'm using too big a motor."

HEDS-5000 encoders require adhesives in the assembly process and an alignment that takes about 3 minutes. Customers told the design team that such assembly processes are not compatible with high-volume automated manufacturing. They need an encoder that can be located and fastened by ordinary mechanical methods with no follow-up adjustments.

Logistics are also important in high-volume manufacturing. Some customers want high-volume capacity with quickly adjustable running rates to avoid costly inventories. Another issue is that count density and code wheel size vary with the application, so a variety of standard parts

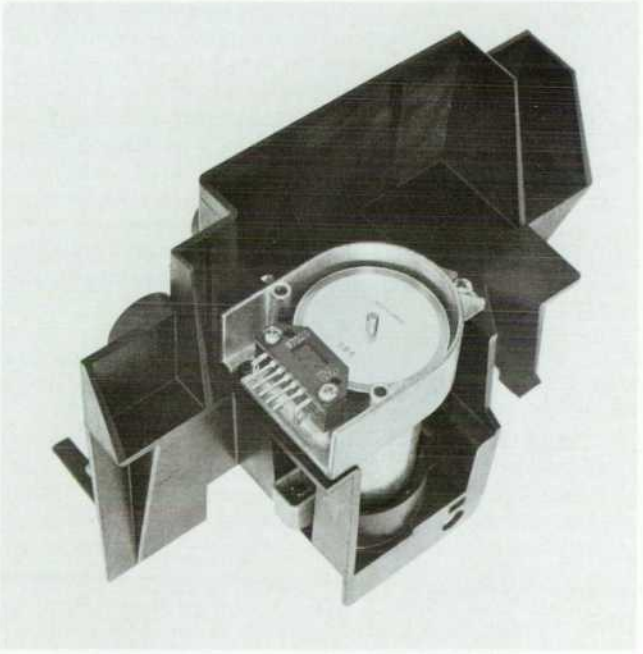


**Fig. 1.** *The HEDS-9000 encoder with code wheel mounted on the DeskJet printer carriage drive motor.*

# Basics of Optical Incremental Encoders

An encoder provides information about the position of some movable part. For example, in a plotter the controller must know where the pen is at all times and issue appropriate commands to the motors to make the desired plots. This discussion will focus on rotary encoders, but the principles apply equally well to linear encoders.

HP produces transmissive optical incremental quadrature encoders. In these encoders, light passes through a patterned wheel and a detector senses the resulting shadows. An incremental encoder only reports relative motion. The controller must use other means to find its reference position, and then count pulses to keep track of the instantaneous position of the moving part.

A quadrature encoder has two output channels which together indicate both the distance and the direction of motion. The signal from each channel is a square wave, one cycle for each spoke of the slotted code wheel. Typical encoder output waveforms and performance definitions are shown in Fig. 1. Channel A leads channel B by ¼ cycle for motion in one direction and lags by ¼ cycle in the other direction.

There are four possible states of the two channels, occurring in the sequence ..., s1, s2, s3, s4, s1, ... in one direction and ..., s1, s4, s3, s2, s1, ... in the other. From any given state, only the two adjacent states are valid. An out-of-sequence state signals an error. Even in unidirectional systems, a quadrature encoder is often used for error checking and/or to double the resolution (four transitions per cycle instead of two).

A digital encoder gives no information about motion until a signal transition occurs. Digitizing discards some information from the analog optical signals. Some applications, such as stopping at a point (with no deadband), might benefit from using the missing analog information. Normally, however, the resolution (states per revolution) of the code wheel is made high enough so the uncertainty within a single state is acceptable.

### Errors in the Output Signals

There is sometimes confusion about the terms accuracy, resolution, and repeatability. All three have something to do with knowing the position, but there are important differences. Accuracy is the relation between the reported position and the actual position. Resolution defines how small a motion can be detected. Repeatability concerns how closely the system can return again to a particular location. For example, imagine an encoder with a very high-count code wheel mounted on a bent shaft. Accuracy will not be very good, because the center of the wheel isn't where it should be. But the system can move and return to the same spot quite precisely (repeatability) or detect a small motion (resolution). On the other hand, a well-made encoder with a small number of counts per revolution would have good accuracy because its output pulses occur at the right places, but low resolution because the shaft has to turn a long way before the next transition.

Since the output signals are periodic, it is convenient to discuss their timing and errors relative to one cycle of those signals. Each cycle is defined as 360 electrical degrees. For a 500-count encoder, 360 electrical degrees equal 1/500 of a full turn of the shaft.

Cycle uniformity (position of corresponding transitions on adjacent cycles) is usually quite accurate, because the various sources of error affect corresponding transitions in the same way. A bent spoke in a metal code wheel does cause a cycle uniformity error, but even that error is decreased because the encoder averages the positions of several spokes. An eccentric code wheel will cause cycle uniformity errors that accumulate into position errors equal to the total eccentricity.

Ideally, the signals for the A and B channels differ in phase by 90 electrical degrees. Errors in this phase relationship can be caused by mechanical misalignment between the code wheel and the detector. In Fig. 2, the dark lines represent code wheel slots, and the cross hairs represent the detectors for channels A and B. Suppose the center of the code wheel is moved a distance e. The code wheel slots that should be over the detectors are no longer in the correct positions. Rotation of the code wheel can still align the slots with the detectors, but it must rotate counterclockwise for A and clockwise for B. The difference between those two positions is the phase error between the A and B channel signals. Reference 3 listed on page 106 calculates this error to be

$$\Delta\phi = (360/2\pi)eSn/r^2,$$

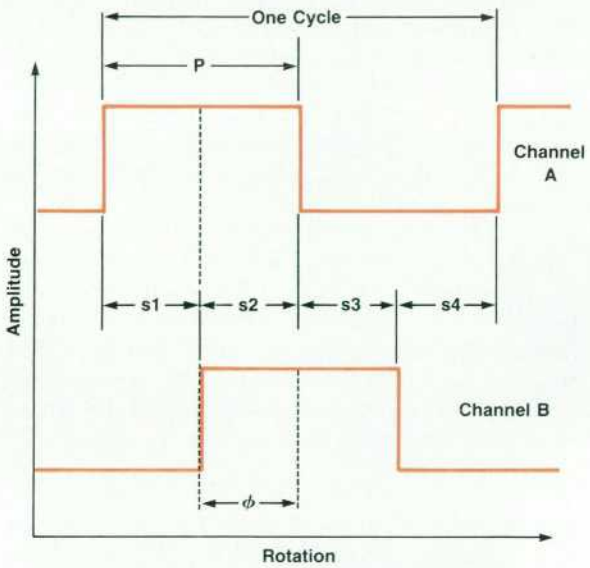



**Fig. 1.** *Typical output waveforms for a two-channel quadrature encoder. One shaft rotation equals 360 mechanical degrees or n cycles, where n is the number of counts per revolution. Each count (cycle) corresponds to one code wheel spoke and window pair. One cycle equals 360 electrical degrees.*
Pulse width (P): *The number of electrical degrees that an output is high during one cycle, nominally 180.*
Pulse width error (ΔP): *The deviation, in electrical degrees, of the pulse width from its ideal value of 180.*
State width (s): *The number of electrical degrees between a transition in the output of channel A and the neighboring transition in the output of channel B. There are four states per cycle, each nominally 90 electrical degrees wide.*
State width error (Δs): *The deviation, in electrical degrees, of each state width from its ideal value of 90.*
Phase (φ): *The number of electrical degrees between the center of the high state of channel A and the center of the high state of channel B, nominally 90 for quadrature output.*
Phase error (Δφ): *The deviation of the phase from its ideal value of 90 electrical degrees.*

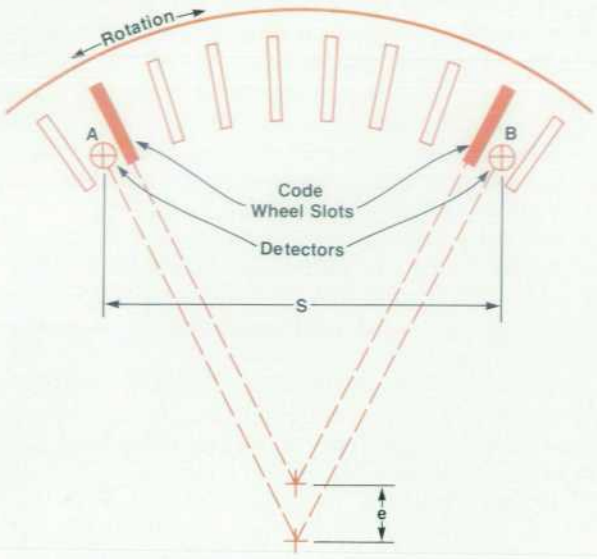



**Fig. 2.** *Lines on a code wheel radically displaced by a distance e no longer overlap targets on the detector. This leads to a change in timing for events seen by the two detectors, A and B. This change in timing is a phase error.*

where $\Delta\phi$ is the phase error in electrical degrees, e is the misalignment, S is the separation between the A and B detectors (near zero in the HEDS-9000), n is the cycles per revolution, and r is the radius of the code wheel.

Errors in pulse width are typically caused by threshhold or balance errors in the detector, or by a nonuniform light source. Pulse width and phase errors combine to cause errors additively in at least one of the four states.

If the code wheel happens to stop right at the point of a transition of the output signals, the output should not chatter or dither between the two adjacent states. Such dithering should not in theory cause the controller to lose count, but it would be a burden. The remedy is to put hysteresis in the logic circuit. This requires the code wheel to travel slightly beyond the ideal switching point, then to switch abruptly. To return to the previous state, the wheel has to go back across the transition point a finite distance before the output will snap to that other state. The amount of hysteresis should be large enough to avoid dithering yet small enough not to add significantly to position uncertainty. In the HEDS-5000 and HEDS-9000 encoders, a built-in circuit provides the required hysteresis.

and a reasonable lead time for specials are important. The HEDS-9000 is designed to accommodate a large variety of formats, including use of linear scales instead of rotating code wheels.

In summary, market research told us that the new encoder had to be dramatically less expensive, both to buy and to use.

### Overall Results

Out of the user feedback came ambitious goals, leading to a product that matches well the original customer requests. Compared with the HEDS-5000 family, HEDS-9000 manufacturing costs are reduced by a factor of 4 and labor content is reduced by a factor of 10. Assembly tolerance (detector to code wheel) is increased by a factor of 50. Customer assembly time is reduced by a factor of 10. The maximum recommended operating temperature has been extended from 85°C to 100°C, and the module can operate to 100 kHz over its full temperature range with substantially constant encoding accuracy.

In most two-channel encoders, phase errors between the two signals result from misalignment between the code wheel and the detector. In the HEDS-9000, a 50-to-1 reduction in phase sensitivity to alignment is of key importance. Some of the increased tolerance budget is used in manufacturing to decrease cost, while most of it is given the customer in the form of increased assembly allowances. Encoding errors are discussed later in this article and in the box on page 100. A more detailed analysis of encoder errors can be found in reference 3.

In mounting a HEDS-9000 module, the customer is allowed a mounting position error of ±400 μm in any direction. One version has about 30 μm resolution (2000 transitions per revolution on an 11-mm-radius code wheel) and

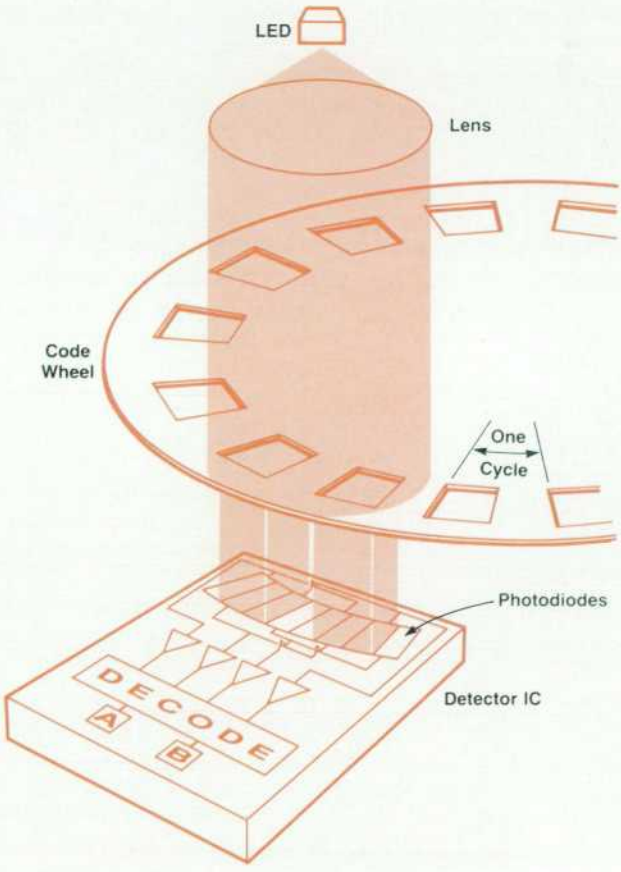



**Fig. 2.** *Optical arrangement of the HEDS-9000 encoder. There are four photodiodes per code wheel cycle (one spoke and window pair). All four contribute to both output channels (A and B).*

is accurate within about 10 μm. Thus the allowed mounting position error is about 40 times the accuracy of the resultant encoder. The repeatability of the resultant encoder is even tighter, about 4 μm or 1/100 the allowed assembly error. This repeatability is maintained over a recommended temperature range of −40°C to 100°C, a frequency range of 0 to 100 kHz, and an operating life greater than 50,000 hours.

### Design Approach

The design approach is to create a beam of light, pass it through the customer's code wheel, and report the locations of the shadows. Like the HEDS-5000 encoder, the HEDS-9000 design takes advantage of in-house expertise and capabilities in integrated photodetector circuits, LED technology, plastic optics, and high-volume manufacturing. A major strategy was to use geometric intimacy between channels A and B to reduce phase sensitivity to mounting position. This led to a new IC design and elimination of a lens and an aperture plate—the plate was used for phasing the two channels. Optical balance is maintained by design throughout the fabrication process, eliminating the need for adjustments at final test. Precise collimation allows wide gaps between the code wheel and the detector. Packaging integration reduces the number of parts and increases reliability. Manufacturing technologies are adapted from the semiconductor and hybrid industries to take advantage of proven, inexpensive, and reliable processes.

The detector IC contains an array of photodiodes and circuitry to process the resulting signals. The array is custom-made to work with a specific code wheel design. All the signal processing is done by comparing light levels received at the various detector regions. This is differential or push-pull detection, which avoids the need to calibrate the brightness of the light source.

### Geometric Intimacy in the Detector Design

Fig. 2 shows the HEDS-9000 optical arrangement. There are four photodiodes per code wheel cycle (one spoke and window pair), and each of the four photodiodes contributes to both output signals. The shared function merges the effective locations of the detectors for channels A and B. This geometric intimacy reduces phase errors between the channels in response to alignment errors, as discussed on page 100. In the earlier HEDS-5000 encoder, separate detectors for channels A and B were located about 2 mm apart.

The HEDS-9000 detector combines light detection and signal processing for both output channels on one chip. With minor process adjustments, a normal IC manufacturing step makes the photodiodes.

Each point on the photodiode array can only measure light intensity, but the integrated detector as a whole is sensitive to the pattern of light and shadow. Different counts-per-revolution geometries are provided by changing a single IC mask layer.

In use, the light source and code wheel combine to project a pattern of light and shadow on the detector IC. The light levels falling on the various photodiodes behind the code wheel are compared with each other to determine the shadow locations.

The earlier HEDS-5000 design has a metal aperture plate to shutter the light passing through the code wheel, and lenses to focus the light on separate detector areas. Not only do these parts cost money, but the aperture (phase) plate also blocks half the light passing through the code wheel at the signal transitions. In the new encoder, eliminating the aperture plate and interdigitating the detector areas puts to good use light that would otherwise have been lost.

The signal processing portion of the IC includes nonlinear operations to make it respond to the ratio of the light intensities in the shadows and illuminated areas. Absolute brightness of the light source is not a significant concern. At very low light levels, performance is limited by noise and leakage currents. At very high light levels, the amplifier stages overload. Between those extremes there is a range of about 200:1 of satisfactory operation.

At the logic switch points, hysteresis is provided to avoid oscillation. The hysteresis circuit was developed for the previous generation of encoders,[4] and is used here as well.

Most other pieces of the previous circuit were also reused to reduce the design effort and maximize the probability that the new design would work the first time, which it did. The output circuitry was modified to improve frequency and temperature response.

Included in the borrowed circuitry is the block that compares the photocurrents from the alternating pattern of light and shadow to make the digital outputs. It also creates an analog signal that follows the original triangular waveshape of the photocurrents, but has an amplitude independent of light level. The signal is not available to the customer,

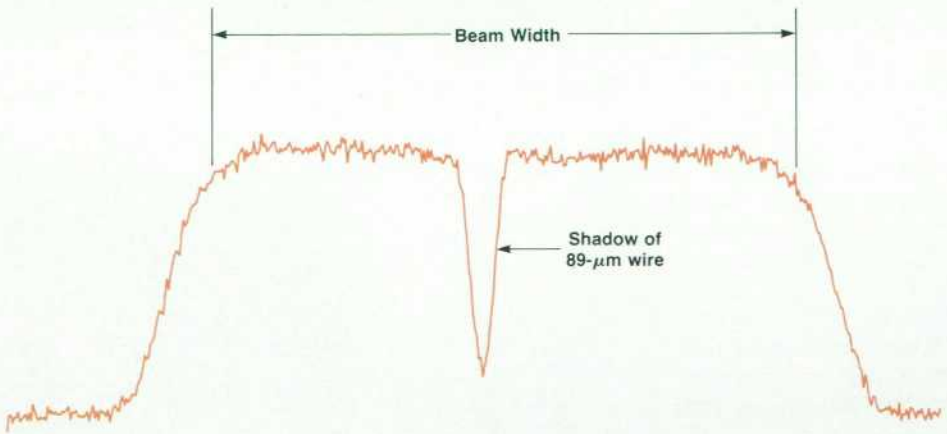



**Fig. 3.** *A scan plot showing the light energy uniformity of the LED source and the shadow of a wire located 45 wire diameters from the source.*

but is a useful in-process test point for monitoring the optical system.

### Optical Balance in the Emitter/Lens System

Differential detection accommodates large variations in light level, but requires matching between the two halves of the system. Imbalance in the optical path or circuitry creates pulse width errors. Although it might have been possible to adjust pulse width at final test, unit-by-unit adjustments clutter the path to manufacturing simplicity. The design stategy for the HEDS-9000 was to make the optics and detector inherently balanced. This was not particularly difficult in the detector, but in the emitter system this was technically challenging.

The detector design takes advantage of matching components that come naturally in an IC. IC resistors, for example, may have 20% variation from batch to batch, and a significant temperature drift, but adjacent resistors on the same IC usually match each other within 1%. The circuit leans heavily on characteristics that come free and avoids unrealistic expectations on process control.

One figure of merit for encoders is their tolerance for uncertainty in the distance from the code wheel to the detector. Such tolerance allows the system designer to use cheaper code wheels (less flat), cheaper bearings (more end play), or less careful assembly. The code wheel should not rub on the detector at its closest position, yet must cast a clear shadow at its farthest position. This leads to another restriction on the light source: it must produce a collimated beam of parallel light rays. The HEDS-9000 emitter lens is a precision-molded polycarbonate lens with an f-number of 0.7 for high efficiency and two aspheric surfaces for control of collimation and uniformity. The emitter, a light-emitting diode (LED), produces more intense light on-axis than at wide angles. The lens funnels more of the weaker wide-angle light to each unit area of exit surface. It does this compensation while causing more than 50% of the total light from the emitter to emerge in a parallel, collimated beam.

Tight collimation requires a small light source as well as precise lens surfaces. The gallium arsenide phosphide emitter radiates from a 60-$\mu$m-wide active area on its top surface.

Emitter system characteristics are monitored in the manufacturing process to ensure consistent collimation and unformity. Fig. 3 shows the intensity profile across the diameter of the light beam. The dip in the middle of the graph is the shadow of a test wire at a distance of about 45 wire diameters. The contrast between the shadowed and nonshadowed areas remains good.

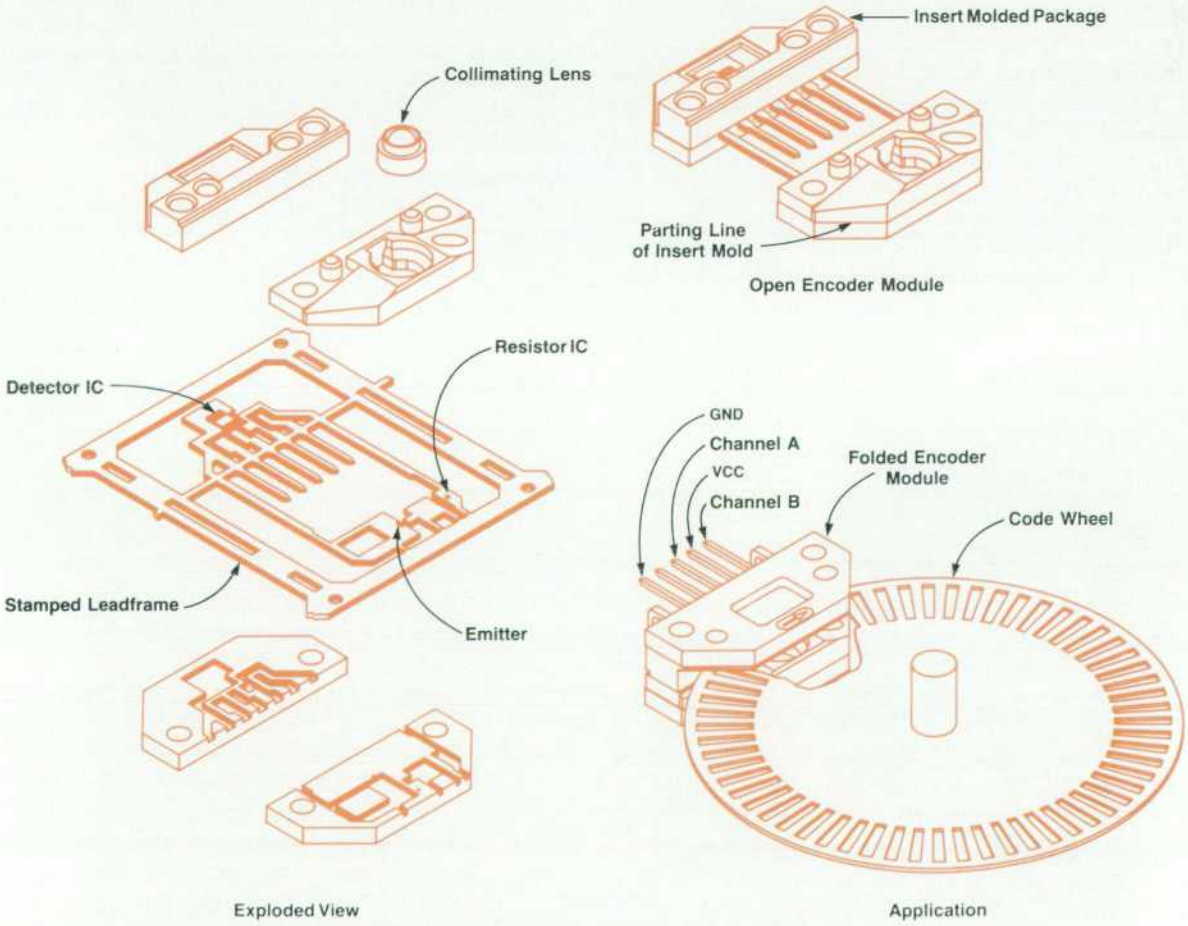



**Fig. 4.** *Package design of the HEDS-9000 encoder showing the fold-over insert-molded leadframe.*

### Package Design

The package design is illustrated in Fig. 4. The package carries the IC, the LED, the lens, and a current limiting resistor. It provides electrical interconnections between these elements and includes connecting pins to the outside world. The package also aligns the emitter system to the detector and provides mounting location features. A plastic body molded around a stamped metal leadframe was developed for the HEDS-9000 because it performs all of these functions, replacing as many as 10 separate parts (see Fig. 5), saving labor, and improving reliability.

The molded leadframe is flat for die attachment and wire bonding of the detector IC, the emitter, and the current-limiting resistor. The detector is given a protective transparent encapsulant. The collimating lens is inserted over the emitter cavity and sealed. The package is then folded into its final C shape and tested. Five 0.025-inch-square posts on the leadframe become the connecting pins. Details on the molded plastic align the emitter system to the detector. Holes provide for screw mounting and registration of the module by the customer.

### Manufacturing Technology

The assembly process for the HEDS-9000 is similar to that used for a standard IC package. Since IC manufacturing is a well-developed technology, off-the-shelf machines are widely available for high-volume production.

The LED, the IC, and a resistor die are automatically positioned and wire-bonded using standard equipment. IC encapsulation, lens insertion and sealing, package folding, and testing operations are done on equipment specifically developed for the HEDS-9000.

Because the HEDS-9000 is not dependent on special alignment by the customer, it can be 100% tested in an as-used configuration for both electrical characteristics and encoding parameters. Testing is performed at over 700 units per hour with a tube-to-tube parts handler. To test a 1000-cycle-per-revolution encoder at the required resolution of 1/360 of a cycle, the tester must resolve about 4 seconds of arc (0.000017 radians). A code wheel turning at constant speed allows the tester to make time measurements rather than angle measurements. Each output channel is monitored at a 50-MHz sample rate.

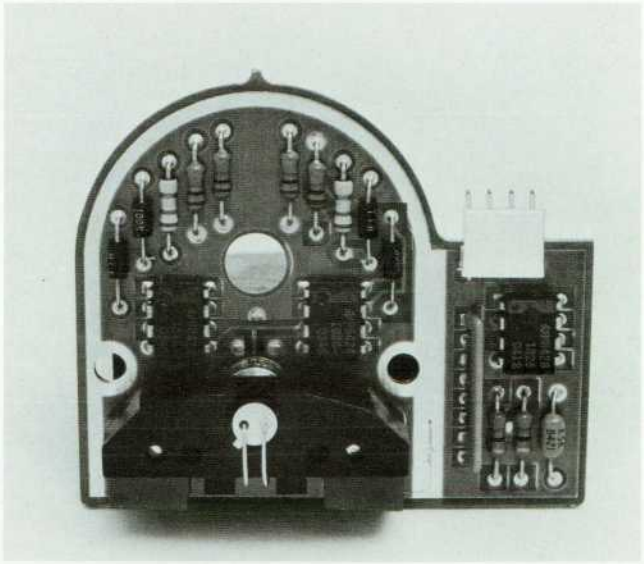


**Fig. 5.** *An encoder previously used in many HP plotters. It had 28 parts. The HEDS-9000 module that replaced it has five.*

### Production System

Work in progress (WIP) flows through the HEDS-9000 production line under a pull system, that is, WIP is processed only as required to satisfy the needs of the next operation. Rework, hot lots, and fill jobs are not allowed. The result is short lead times without huge inventories. 1,000-piece orders are delivered in less than four weeks. Pull systems and JIT (just-in-time) manufacturing are common throughout HP and other manufacturers, but there are some difficulties in applying a JIT philosophy to component manufacturing. Because of special process steps, such as oven cures, components are best made in batches to allow better use of equipment and labor. HEDS-9000 production machines are loaded using magazines holding 48 units each. This accommodates batch processing while allowing the benefits of a pull system.

Fig. 6 is a simplified diagram of the HEDS-9000 production line with workstations and WIP holding areas. A workstation is an operation or combination of operations under the control of one operator. The workstations are balanced, that is, all stations run at approximately the same speed. WIP flows from a cabinet through a workstation and into the next cabinet on a first-in, first-out basis. WIP cabinet contents are not allowed to exceed a preset maximum. Limits are set to allow a certain amount of random speed variation and machine setup time without stopping the line. For example, if station 3 shuts down long enough, cabinet B fills to its maximum and station 2 cannot work. This propagates backward, so one person's problem becomes everyone's problem. A cabinet that is always full indicates that the next station is a bottleneck and it receives engineering attention. If the speed at a bottleneck is increased by 5%, the output of the entire line increases by 5%.

The pull system works best if WIP has few reasons to stop before completion. No rework of defective units is allowed. Effort is spent learning to make things right rather than learning to rework. Yield is high. Quality problems are painful and get fixed in a hurry. Cycle time has been reduced by eliminating time-consuming steps. For example, conventional adhesives and coatings would require a

**Table I**
HEDS-9000 Reliability Test Results (Abridged)

| Test | Devices | Failures |
|---|---|---|
| High-Temperature Operating Life (+100°C), 1000 Hours | 105 | 0 |
| 85°C, 85% Relative Humidity, Biased, 1000 Hours | 90 | 0 |
| Temperature Cycle (−40°C to +100°C), 800 Cycles | 55 | 0 |
| Shock, 1500g, 5 ms, 6 Planes, 5 Blows | 5 | 0 |

total of about 14 hours of oven cure time. To eliminate one oven cure, an adhesive was developed that cures in less than 5 seconds when exposed to ultraviolet radiation. Cumulative cure time was reduced to 2 hours.

### Reliability

Table I summarizes the reliability test results for the HEDS-9000. Reliability is inherently good because of the low number of parts and interconnections. Here again, the benefits of integrated design are reaped. Judicious choice of materials, stress-test monitoring of production parts, and statistical quality control result in consistent performance.

### Encoder Selection

There is no one best encoder for all applications. Assum-


**Fig. 6.** *Simplified diagram of the HEDS-9000 production line, showing work-in-progress (WIP) holding areas and workstations. The line operates under a pull system.*

## A Complete Encoder Based on the HEDS-9000 Encoder Module

Many customers want a complete, easy-to-use encoder without the complications of procuring a code wheel and a housing. The HEDS-5500 Quick Assembly Optical Encoder, Fig. 1, is the response to this need. The goals for this encoder were for customer assembly time to be less than 30 seconds, to have a mounting tolerance of ±0.25 mm, and to require no special assembly tools. It also had to be dust resistant, be able to accommodate a through shaft, and be easily removed.

An important step in encoder assembly is the setting of the gap, or the distance from the code wheel to the photodetectors. The HEDS-5500 encoder's integral, multifunction twist cap design solves the problem of setting the gap quickly without special tools.

The HEDS-5500 encoder main housing contains the code wheel, the twist cap, and the HEDS-9000 encoder module. The encoder is shipped with the twist cap preset to provide the proper gap and comes supplied with a preinserted hex wrench and a baseplate.

To mount the HEDS-5500 encoder to a motor, the baseplate is first attached with screws. The main housing is then snapped onto the baseplate. Applying a downward force on the end of the supplied-in-position hex wrench levers the code wheel hub upwards against the twist cap. A set screw is tightened using the same hex wrench to secure the code wheel to the motor shaft, and the wrench is removed. Turning the cap lifts it away from the hub, allowing the code wheel to turn freely, and also moves a wiper arm over the hex wrench access hole to keep out dust. This finishes the 15-second assembly sequence.

The cam-actuated twist cap plays a key role in setting the gap between the code wheel and the detector. It provides a reference surface to the code wheel hub in one position and moves up out of the way in its other position. A blade screwdriver can be used to turn the twist cap through a slot on the top surface or through exposed slots on the cap's wiper arm. In the case of a through-shaft option, only the wiper arm slots are available.

The twist cap also limits radial travel of the code wheel hub to prevent damage to the code wheel during shipping.

*Chris Togami*
Development Engineer
Optoelectronics Division

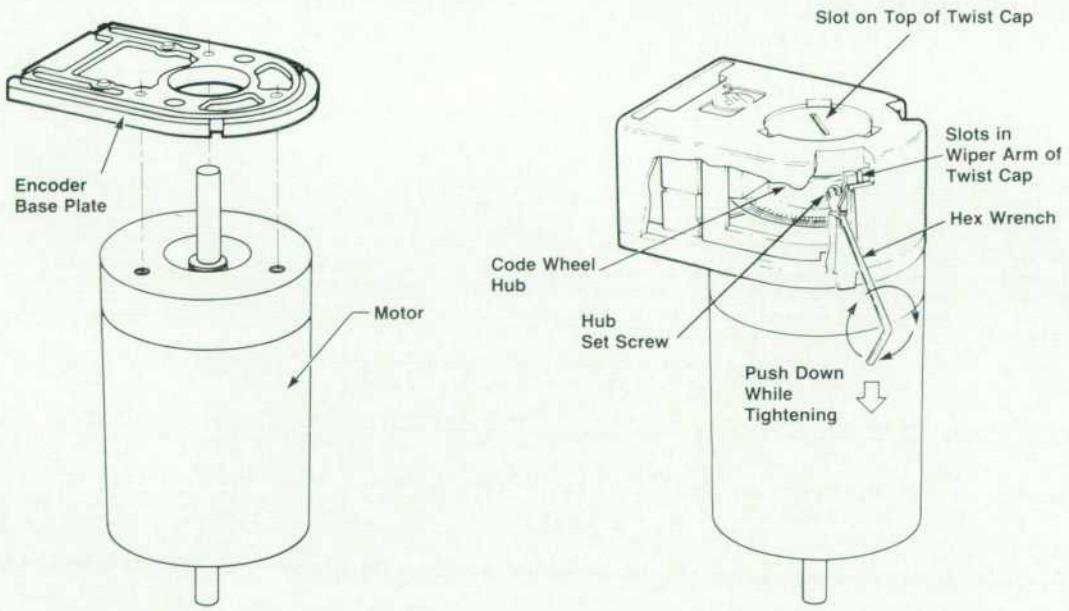



**Fig. 1.** *To attach the HEDS-5500 encoder to a motor, a baseplate is attached to the motor with screws and the encoder is snap mounted to the baseplate. The encoder housing contains a code wheel and an HEDS-9000 encoder module.*

ing that an optical digital incremental encoder like the HEDS-9000 is appropriate for the application, some remaining important choices involve code wheel diameters and counts.

Larger code wheel diameters give better accuracy, but take up more room and have greater inertia.

Higher code wheel counts give better resolution, but at a higher signal frequency, which could approach the limits of the detector or the controller. Count and diameter combine to determine feature size. Cost can be largely dependent on area, a fact that favors small diameters until a point is reached where the feature size gets too small. Code wheels with very small features are hard to make, therefore expensive.

All encoders in the HEDS-9000 family share the same temperature and frequency capabilities. Different versions are available for various code wheel sizes and resolutions. The HEDS-9100 module is used with 28-mm-diameter code wheels. The HEDS-9200 is for linear applications using a code strip rather than a rotating wheel. The part number HEDS-9000 specifically refers to modules used with 56-mm-diameter code wheels but is also used as a family part number. Many standard resolutions are available and specials between 1.2 and 8 lines per millimeter can be provided by tooling a new photodiode layout on the IC.


### Acknowledgments

Mark Bullock conceived much of the manufacturing architecture. Chris Togami designed the leadframe and plastic parts. Joe Dody and Donald Lapray contributed in the areas of processes and tooling. Art Wilson did prototype tooling. Thomas Lugaresi provided IC layout direction. Akhtar Khan's analysis provided the insight that led to the detector IC's ability to operate smoothly over its wide temperature range. Thomas and Akhtar brought the insight to fruition with simulation and physical modeling. Craig Sue, whose leadership was essential for keeping the whole IC task on schedule, was responsible for IC product engineering, including test development and IC characterization. Curt Wilson did product characterization and ensured the integrity of the data sheet. Richard Ruh managed the project in its development phase with unrelenting dedication. He took over from the able leadership of Debbie Haferburns, who left to manage her family, also in development. Larry McColloch gets credit for the pull system and much ingenious tooling design. Victor Loewen and Bill Bilobran supplied expertise in many areas including tester design. Dave Oshima, Ray Tam, Yoshi Tatsumi, and Maria Costa made successful production possible. Bill Loesch and Bill Beecher provided unwavering support and sage advice from the inception of this project to product release. Bob Steward brought in the right people to make it all happen. Lui Kok Chwee, George Lim, and many others in Singapore established high-volume production capabilities. Karen Owyeung and her predecessor Lisa Wade made certain that the customer viewpoint predominated.


### References


1. *Hewlett-Packard Journal*, this issue, pp. 6-52.
2. H.C. Epstein, M.G. Leonard, and J.J. Uebbing, "An Incremental Optical Shaft Encoder Kit with Integrated Electronics," *Hewlett-Packard Journal*, Vol. 32, no. 10, October 1981, pp. 10-15.
3. H. Epstein, "Optical and Mechanical Design Trade-offs in Incremental Encoders," *Proceedings of the Tenth Annual Symposium on Incremental Motion Control Systems and Devices*, Chicago, Illinois, June 1981, p. 57.
4. M. Leonard, "Push-Pull Optical Detector Integrated Circuit," *IEEE Journal of Solid-State Circuits*, Vol. SC-15, no. 6, December 1980, p. 1087.

# Authors

October 1988

### 6 Discless HP-UX

**Scott W. Wang**

Scott Wang served as project manager for a variety of calculator software projects before he joined HP's UNIX development team. Successively, he was a project manager and R&D section manager involved in HP-UX development for the HP 9000 Series 300. Scott now is R&D manager at the Information Software Division of HP and continues to be responsible for HP-UX software. He came to HP in 1972, when he joined the Calculator Products Division in Loveland, Colorado. His BSEE degree is from the Massachusetts Institute of Technology (1971) and his MSEE degree is from the University of Michigan (1972). Scott is a member of the IEEE. He has contributed two previous articles to the HP Journal. He was born in Taipei, is married and has two children. He is an afficionado of high-fidelity audio, video, and photography.

### 10 Discless File System

**Debra S. Bartlett**

Debbie Bartlett was responsible for the HP-UX 6.0 file system, particularly the I/O and FIFO scheme. She has since become a project manager for the file system and discless testing. Debbie attended Purdue University, where she obtained a BS degree in mathematics in 1977. Her MS degree in computer science is from Colorado State University (1982). She was born in Indianapolis, Indiana, is married, and has two small daughters. Debbie's husband is a project manager at HP's Colorado IC Division. She resides in Ft. Collins, Colorado, and enjoys outdoor activities with her family.

**Joel D. Tesler**

As the lead engineer of the team that conceived and built the original distributed HP-UX environment, Joel Tesler designed most of the initial file system code and the message interface. In a previous project, he designed a softkey package and other environments for the HP 64000-UX system. Joel came to HP in 1980, when he joined the Logic Systems Division. He is the coauthor of a paper on HP-UX operating systems presented at Uniforum 87, and he has previously contributed to the HP Journal. He attended the University of California at Davis, where he received his BS degree in computer science in 1980. Joel was born in Los Angeles and lives in Cupertino, California. His favorite pastime is orienteering, a cross-country race in which contestants navigate through unfamiliar territory using only a compass and a map.

### 15 Discless Program Execution

**Ching-Fa Hwang**

Ching Hwang initiated and managed the DUX project at HP Laboratories. He continued to manage kernel and integrated systems at HP's Information Software Division and led their integration into HP-UX products. He coauthored a paper on the subject at Uniforum 87, and a patent application describing the DUX network protocol includes his ideas. Previously, Ching led two projects aimed at developing distributed data bases. Before joining HP Laboratories in 1979, Ching's professional activities included real-time process control, computer architecture and processors, and multiple-processor operating systems. His BSEE degree is from the National Taiwan University (1971), and his MS degree in computer science is from the University of Utah (1974). Ching and his wife, who also works for HP, have two sons and live in Cupertino, California. He is building a koi pond with waterfalls and enjoys playing the piano.

**William T. McMahon**

The remote swapping scheme for the discless workstations was Bill McMahon's primary project. His previous development assignments include the graphics ROM for the HP 9826 BASIC Release 1.0, and the linker and assembler for Release 2.0 of the HP 9000 Series 200 HP-UX system. He holds a BA degree in philosophy from Ohio University (1971) and an MS degree in computer science from Colorado State University (1979). He came to HP in 1979. Bill is married and has two children. Among his favorite recreational activities are Tai-Chi, cross-country skiing, hiking, and backpacking.

### 20 Discless Network Functions

**David O. Gutierrez**

As a member of the team developing the HP-UX 6.0 software, David Gutierrez' responsibilities included the network functions and transport, protocol and buffer management, and configurability. Before joining HP in 1985, he worked for Digital Equipment Corporation, Western Electric Company, and Bell Laboratories. David's main professional interests are the UNIX operating system, special-purpose networking protocols, and distributed operating systems. He attended the University of New Mexico, where he received his BS degree in 1980, and did graduate work in computer engineering at the Illinois Institute of Technology. He has taught a project business class in a middle school and serves as treasurer of the Parent/Child Education Center in Windsor, Colorado, where he lives. He was born in Pueblo, Colorado, is married, and has two daughters. He has designed his own house and enjoys woodworking, golf, skiing, camping, and "anything that doesn't deal with computers."

**Chyuan-Shiun Lin**

Distributed operating systems, data base systems, and communications are Chyuan-Shiun Lin's focal professional interests. Before working on the distributed HP-UX system, his responsibilities included data base research at HP Laboratories and work on the HP-UX Release 2.0 for the HP 9000 Series 800. Before coming to HP in 1981, he worked in the data processing field. He has published a number of papers on a variety of computer subjects and has contributed to a protocol design for which a patent is pending. Chyuan-Shiun is a member of ACM. His BSEE degree is from the National Taiwan University (1970), and his master's degree in computer science (1976) is from the University of Utah. Born in Taipei, he is married and has three children. He lives in Cupertino, California.

### 27 Discless Crash Recovery

**Annette Randel**

Crash detection and recovery, system reboot, self-test, and the file system cnode maps were among Anny Randel's projects for the HP-UX 6.0 system. Previous responsibilities include work on the boot ROMs for the HP 9000 Series 200/300 Computers and the assembler and commands for the Series 200. She first joined HP in a summer-student position in 1981, and two years later joined full-time.

Anny's BS degree in computer science and computer engineering is from Graceland College (1981) and her MS degree in computer science is from Colorado State University (1983). Born in Roseville, California, she is married and lives in Ft. Collins, Colorado. She sings for a pop/jazz group and plays both clarinet and baritone saxophone. Her other hobbies include bicycling, triathlons, running, skiing, softball, and water skiing.

## 33 Discless Boot Mechanism

**John S. Marvin**

A software engineer on the HP-UX Release 6.0 project, John Marvin developed a number of commands for discless operation. Before he came to HP in 1984, John worked as a programmer analyst for his alma mater, the University of Virginia. Both his BS and his MS degrees are in computer science (1981 and 1983, respectively). He was born in Brooklyn, New York, is married, and lives in Ft. Collins, Colorado. Among his favorite off-hours activities are skiing and ballroom dancing.

**Perry E. Scott**

Perry Scott's primary responsibilities for HP-UX Release 6.0 were the secondary loader, discless kernel initialization, discless context for CDF, and system clock synchronization. He had previously worked on Releases 5.0, 5.1, and 5.2. He has been with HP since he earned his bachelor's degree in electrical engineering from North Dakota State University in 1980. Perry has served in the Air National Guard for six years. He was born in Fargo, North Dakota, is married, and lives in Ft. Collins, Colorado. Gardening and bicycling are among his favorite leisure activities.

**Robert D. Quist**

Among the HP projects Robert Quist has worked on since he joined HP in 1971 are a Lisp workstation, third-party support, a Pascal workstation and a boot ROM for the HP 9000 Series 200 system. To the HP-UX Release 6.0 system he contributed boot ROM revisions B and C. Specialty boot ROMs, Pascal workstations, low-level drivers, and human interfaces are Robert's special interests. His BE degree in computer science is from Brigham Young University (1971). Born in Lethbridge, Alberta, Robert is married and has eight children. He lives with his family in Loveland, Colorado. He is active in the Cub Scouts and teaches Pascal programming. Robert enjoys birdwatching, camping, and reading science fiction.

## 37 Discless System Configuration

**Kimberly S. Wagner**

Kim Wagner's responsibilities on the HP-UX Release 6.0 project included the discless administration tools and system software integration. She came to HP in 1986. She holds a BS degree in computer science and mathematics from the University of California at Davis (1983) and an MS degree from Colorado State University at Ft. Collins (1986). Kim is a member of ACM and SIGGRAPH. She was born in Redwood City, California, and now lives in Ft. Collins, Colorado.

## 39 SCSI

**Paul Q. Perlmutter**

Paul Perlmutter's responsibilities in the HP-UX Release 6.0 project included the mass storage software, the driver support, and backup strategies. High-performance mass storage devices for UNIX systems were the main focus of the positions Paul held before joining HP in 1985. His PhD and MS degrees in mathematics are from the University of Colorado (1975 and 1971). He has held a position as a college professor of mathematics and has published several articles on the subject. Paul serves as president in his synagogue in Ft. Collins, Colorado, where he lives. He was born in New York and has two daughters. His favorite pastimes are bicycling, photography, and hiking, but he spends most of his spare time with his children.

## 46 X Window System

**Frank E. Hall**

Frank Hall is a project manager for user interface productivity tools and has held a similar position in the development of the Xrlib and HP X Widget user interface libraries for HP-UX. Before coming to HP in 1979, he worked as a system analyst for the Computer Sciences Corporation, where he helped develop a worldwide computer network for communications with the space shuttle and geosynchronous satellites. At HP, Frank has worked as a software engineer on operating system firmware for the HP 71B Handheld Computer and application software for the HP Portable PLUS laptop computer. He has published three articles in the proceedings of the HP Software Engineering Productivity Conference. Frank holds a BA degree in mathematics from Florida State University (1972) and an MA degree in anthropology from the University of Texas at Austin (1979). He is a member of the ACM. He was born in Ft. Myers, Florida, is married, and lives in Corvallis, Oregon. His leisure interests include bicycle touring, birdwatching, folk music, skiing, white water rafting, and fishing.

**James B. Byers**

With marketing user interface technology his focal interest, Jim Byers is the product marketing engineer involved with HP's release of the X Window System Version 11. He joined the Indianapolis sales office of HP in 1982 and has served as the marketing representative for the HP 9000 and HP 1000 Computers. Jim's BSEE degree is from Purdue University (1980) and his MBA degree in marketing is from Indiana University (1982). He was born in South Bend, Indiana. He's married, has a small daughter, and lives in Corvallis, Oregon. In his off-hours, he likes skiing, camping, and exploring the outdoors with his family.

## 51 Managing DeskJet Development

**John D. Rhodes**

When he joined the Microwave Division of HP in 1966, John Rhodes had just received his MBA degree from Stanford University. He also holds a bachelor's degree in industrial technology from Long Beach State College (1964). In his varied career at HP, John served in many different engineering and managerial positions. As project manager of the mechanical team working on the DeskJet project, he originated several patents. John was born in San Jose, California. He is married, has a daughter and a son, and lives in Vancouver, Washington. For seven years, he served on the board of directors for a puppet theater which tours extensively throughout the United States. His hobbies include astronomy, photography, and piloting light aircraft.

## 55 High-Resolution Printhead

**Kenneth E. Trueba**

The printhead firing chamber, the ink feed process, and a method for bubble observation were among Ken Trueba's design assignments for the DeskJet pen. Earlier design work included the thermal printheads for the HP 2621P Terminal, the HP 2671 Thermal Printer, and the HP 85A/B Personal Computers. On the DeskJet development team, Ken was responsible for process, assembly, and architectural design of the printhead. He came to HP after receiving his BS (1974) and MSEE (1976) degrees from the South Dakota School of Mines. Ken has presented papers and published a journal article on the subjects of thin-film techniques and thermal inkjet devices. Three patents based on his designs are pending. Ken was born in Boise, Idaho, is married, and lives with his wife and two daughters in Corvallis, Oregon. He enjoys photography, skiing, playing the guitar, and psychology.

**Richard R. Van de Poll**

A process engineer in the team developing the DeskJet printer, Rich Van de Poll worked on printhead-related assignments. He is now a project leader at the Inkjet Components Operation. His BS degree in chemical engineering is from the University of Colorado. Before he joined HP's Logic Systems Division in 1986, he had been a process engineer at Eastman Kodak, where he participated in developing photographic emulsions. Born in Surabaja, Indonesia, Rich served three years in the U.S. Army. He is married, has two children, and serves as a cubmaster in the Boy Scouts. He lives in North Albany, Oregon, and spends his leisure time with photography and scuba diving.

**Paula H. Kanarek**

A statistician and R&D project manager for the DeskJet product, Paula Kanarek came to HP in 1982. Her bachelor's degree is from the University of Michigan (1967), and her ScM (1969) and ScD (1973) degrees in biostatistics are from Harvard University. Before joining HP, Paula held positions as an assistant professor in statistics and biostatistics, respectively, at the University of Washington and at Oregon State University. Statistical applications and reliability in engineering are her focal interests and provide the subjects of some 20 of her publications. She is a member of the American Statistical Association, the American Society of Quality Control, and the Society of Women Engineers. Paula is active in local school advisory committees at Salem, Oregon, where she lives. She is married and has two children. For recreation, she likes hiking, bicycling, and cross-country skiing.

**Robert N. Low**

Starting as a temporary hire during summer recess, Bob joined HP permanently in 1977, when he received his BS degree in metallurgical engineering from Purdue University. His first assignment involved the hybrid development for the HP-41C Calculator. He went on to other projects, participated in development of the ThinkJet printer, and eventually became project manager for pen assembly and manufacturing technology for the DeskJet printer. Bob's work has resulted in four patents for DeskJet-related devices. His professional interests focus on high-precision, high-volume manufacturing technology, and he has previously contributed to the HP Journal. He was born in South Bend, Indiana. Bob is married, has two small daughters, and lives in Corvallis, Oregon. His leisure interests include sailing, scuba diving, skiing, hiking, and sports cars.

**William A. Buskirk**

Bill Buskirk came to HP in 1977 with a BSEE degree from the University of Colorado at Boulder. Before he became a project manager for the DeskJet printhead, he handled a variety of assignments as a production engineer and R&D engineer, including work on the HP 82161A Digital Tape Drive. The DeskJet development provided the subject of two papers Bill published in conference proceedings. He recently was appointed R&D section manager at the Inkjet Components Operation. Bill was born in Bloomington, Indiana, is married, and now lives in Albany, Oregon. He has three young children. His favorite recreational activities are hiking, windsurfing, alpine skiing, and sailing.

**Stanley T. Hall**

After joining HP in 1976 at the Corvallis Component Operation, Stan Hall spent over five years as a manufacturing engineer working on tooling for HP 33, 10, and 75 Series calculators and an HP-IL printer. He moved to InkJet development in 1983 and became a project manager in production engineering. Stan's BS degree in industrial technology is from California Polytechnic University. His previous professional experience includes positions as a supplier quality engineer for ISS-SperryUnivac and Intel. Stan is a member of the Society of Manufacturing Engineers. He was born in Oakland, California, and now resides in Corvallis, Oregon. He is married and has two daughters. His leisure activities include woodworking in the winter, sailing and gardening in the summer.

**David E. Hackleman**

As R&D project manager at HP's Inkjet Components Operation, the inks and media used in ThinkJet, PaintJet, and DeskJet printers have been the focus of David Hackleman's interests in recent years. He came to HP with a BSEE degree from Oregon State University (1979) and a PhD in analytical electrochemistry from the University of North Carolina at Chapel Hill (1978). His previous design work at HP includes a variety of integrated circuit processes at the InkJet Component Operation. His work in IC processing, thermal inkjet inks, and multiplexing has resulted in six patents, with several more in application. David is a member of the American Chemical Society and the Electrochemical Society. He is a National Youth Science Camp lecturer and serves as a control station operator of the amateur radio emergency service. David was born in Coos Bay, Oregon, is married, and lives in Monmoth, Oregon, where in his off-hours he operates a tree farm "on forty acres way out of town."

## 62 Printer/Printhead Integration

**John A. Widder**

Author's biography appears elsewhere in this section.

**J. Paul Harmon**

As a development engineer on the DeskJet project, Paul Harmon was responsible for carriage, service station, and interconnect design. A patent is pending for the DeskJet interconnect support structure Paul developed. He came to HP after receiving his BMSE degree from the University of Washington in 1981 and has since earned his MSME degree from Stanford University (1988). Paul has previously coauthored an article for the HP Journal (May 1987). Born in Hermiston, Oregon, Paul is married and has a child. He now lives in Washougal, Washington. Paul's spare time is taken up by motorcycles, sports cars, and church activities.

## 67 Chassis and Mechanism Design

**David W. Pinkernell**

Mechanical design of the servo control for paper feed and carriage drive was Dave Pinkernell's focal contribution to the DeskJet development. With the project since he joined HP in 1981, he worked as a design engineer until the conclusion of the design phase, when he joined a team of manufacturing engineers in the task of setting up production facilities. He is coinventor of a pending patent for the printer mechanism. Dave attended the California Polytechnic State University at San Luis Obispo, where he received his BSME degree in 1981. His MSME degree is from Stanford University. He was born in Santa Barbara, California, and makes his home in Pullman, Washington. His recreational interests include photography and traveling, both of which he recently combined in a trip to East Africa.

**John A. Widder**

The design of the carriage servo system, linefeed motor control, and printhead driver printed circuit boards are among John Widder's contributions to the DeskJet printer. As a development engineer at the Vancouver Division, he also worked with the suppliers of the power supply. Now a manufacturing engineer, John has shifted his attention to the

DeskJet production line. In the past, he has worked on the design of thermal printers such as the HP 2675A, HP 2671/3A, and HP 2674A. John came to HP's Boise Division in 1978, after receiving his BSEE degree from the University of Portland. He is a member of the IEEE and the American Association for the Advancement of Science. He's also active in the City Club of Portland and the World Affairs Council of Oregon. John was born in Bethesda, Maryland, and now makes his home in Brush Prairie, Washington. His favorite pastimes include backpacking and cross-country skiing.

**Kieran B. Kelly**

The DeskJet printer was not the first product to use a paper path design by Kieran Kelly. With mechanical design his main professional interest, he has previously worked on the QuietJet printer, the tilt/swivel base for the HP 150 Computer, and the HP 9826 and HP 9836 Controller/Computers. His work on the QuietJet paper drive resulted in a patent application. Kieran joined HP in 1979, the year he received his BS degree from the University of Virginia. He also holds an MS degree from Stanford University (1984). He lives in Vancouver, Washington, where he is presently renovating a turn-of-the-century Victorian house. Other hobbies include sailing, skiing, bicycling, and hiking.

**Steve O. Rasmussen**

The paper path, carriage, and transmission of the DeskJet printer were the focal points of Steve Rasmussen's design work. He has contributed to designs for the DeskJet printer that resulted in a patent and five patent applications. He came to the Vancouver Division in 1982, after receiving his BSME degree from Iowa State University. Steve also holds an MSME degree from Stanford University (1987). He was born in Fort Dodge, Iowa. He is married, has an infant son, and lives in Vancouver, Washington. Steve likes to spend his off-hours with church activities, working on his house, bicycling, and woodworking.

**Larry A. Jackson**

Larry Jackson coordinated the design of DeskJet paper handling parts, such as the chassis, carriage guide, pinch rollers, pinch springs, and gear trains. An R&D engineer at the HP Vancouver Division, his past product involvement includes the ThinkJet printer, the HP-01 Watch, the HP-41C Calculator, an IC tester, a laser interferometer, and a multichannel analyzer. He has managed the hybrid laboratory at Santa Clara Division. A patent and four patent applications are based on Larry's designs. He attended Utah State University from which he received BS (1965) and MS (1966) degrees in mechanical engineering. He was born in Ogden, Utah, is married and has four children. He resides in Vancouver, Washington. Larry's spare time interests include camping, boardsailing, downhill skiing, and church activities.

## 76 Data to Dots

**Donna J. May**

As a development engineer at the Vancouver Division, Donna May worked on the firmware for the DeskJet printer. On other assignments, she has worked on the HP 2934A Business Printer and a landscape upgrade cartridge for the DeskJet printer. Donna came to the Vancouver Division in 1983, after receiving her BS degree in computer engineering from Iowa State University. Born in Cedar Rapids, Iowa, she is married and resides in Vancouver, Washington. She plays piano and bassoon and likes backpacking and bicycling.

**Claude W. Nichols**

After joining HP in 1980 as a development engineer, Claude Nichols worked on a variety of printers, including the HP 2675A, the HP 2671A/G, the HP 2674A, and the HP 2932A. In the design of the DeskJet printer, he was involved in various aspects of the firmware. Claude's BS degree in computer science is from Brigham Young University, earned in 1979. He was born in Reno, Nevada, but grew up in Cheney, Washington. He is married and has three children. Claude is active in the Boy Scouts and in his church in Vancouver, Washington, where he lives. He enjoys cross-country skiing, bicycling, and hiking.

**Mark D. Lund**

One of the patents pending for the DeskJet printer resulted from Mark Lund's design work. An R&D engineer at the Vancouver Division, he joined HP in 1977 after receiving his BSEE degree from the University of California at Irvine. Among the projects Mark has worked on are the electronics architectures of the HP 2621A Terminal and the HP 2671A and HP 2673A Thermal Printers. He was born in Santa Monica, California, and now lives in Vancouver, Washington. He is married and has a daughter and triplet sons. He enjoys camping, backpacking, scuba diving, and woodworking. He also likes fishing, especially salmon and steelhead.

**Thomas B. Pritchard**

As a development engineer for the DeskJet printer, Tom Pritchard's responsibilities included design of a custom IC and testing and correction of electrostatic discharge conditions. His past assignments include both firmware and electronics design on HP 2934A, HP 2932A, and HP 2671A Printers and work as a production engineer for the HP 3000 Business Computer System. Tom is the primary author of an article describing a microprocessor-based signal processing system for biomedical measurements and also has previously contributed to the HP Journal. A patent is pending for a character generator he developed. Born in Ann Abor, Michigan, Tom is married and has a three-year-old child. He now lives in Vancouver, Washington. He enjoys hiking and playing tennis.

## 81 DeskJet Firmware

**Mark J. DiVittorio**

Mark DiVittorio is a project manager at the Vancouver Division, where his responsibilities included development of the DeskJet firmware. In the past, he has served as development engineer, production engineer, and R&D engineer. His EE and MS degrees in computer science (1974 and 1978, respectively) are from the University of Santa Clara. Born in Chicago, Illinois, Mark is married, has a seven-year-old son, and lives in Vancouver, Washington. In his off-hours, he likes to go fishing.

**Claude W. Nichols**

Author's biography appears elsewhere in this section.

**Michael S. Ard**

As a project manager at the Vancouver Division, Mike Ard directed the development of the Epson FX-80 printer emulation firmware for the DeskJet. His past responsibilities as a development engineer include work on the HP 300 Business Computer System and the HP 2675A, HP 2673A, and HP 2934A printer systems. He also managed the printer systems group, focusing on printer solutions to system and application support. He holds a BS degree (1975) and an MS degree (1978) in computer science, both from Brigham Young University. Mike is active in his church and in youth sports. Born in St. Anthony, Idaho, he is married and has six children. He lives in Vancouver, Washington. Mike enjoys outdoor activities and has been busy designing, building, and landscaping his new home.

**Kevin R. Hudson**

Development of the Epson FX-80 printer emulation firmware, specifically the graphics and parser, was Hud Hudson's focal interest on the DeskJet project. In previous years, he has worked on printhead and character set development for the ThinkJet printer and hardware for the QuietJet. Hud earned his BS degree at Iowa State University in 1981 and shortly thereafter joined HP, where he now is a development engineer at the Vancouver Division. He was born in Vinton, Iowa, is married, and lives in Vancouver, Washington. His recreational interests include softball, golf, and science fiction.

**Brian Cripe**

Development of the DeskJet formatter was among Brian Cripe's most recent projects. Presently, he is working on the X Window System at the Corvallis Workstation Operation, and past assignments include the mechanism controller code for the ThinkJet printer. Brian has originated a text scaling system for which a patent is pending. His BSCE and BACS degrees are from Rice University (1982). Brian was born in Anapolis, Brazil, is married, and lives in Corvallis, Oregon. His favorite pastimes are bicycling, telemark skiing, and tending his prize-winning roses.

**David J. Neff**

David Neff's responsibilities for the DeskJet printer included development of the Epson FX-80 emulation firmware. In the past, he has worked on the RTE-L and RTE-XL operating systems and landscape cartridge firmware. He also developed CAD/CAM software links used internally in HP's Vancouver Division. David attended Harvey Mudd College, where in 1979 he earned his BS degree in mathematics. He was born in Portland, Oregon, is married, and has two children. He now resides in Vancouver, Washington.

## 87 Robotic Assembly

**P. David Gast**

As a manufacturing engineer at the Vancouver Division, Dave Gast developed an automated high-volume assembly line for mixed-mode production of DeskJet and RuggedWriter 480 printed circuit boards. He also designed the mechanical hardware for the robotic workcell that builds the boards. In a previous position, Dave worked for the Research Center of Weyerhaeuser Company. He holds a BSME degree from Texas A&M University (1982) and an MBA degree from Oregon State University (1984). He was born in Minneapolis, Minnesota, and now lives in Vancouver, Washington. Windsurfing, bicycling, telemark skiing, and photography are Dave's favorite leisure activities.

## 91 CIM and Machine Vision

**Robert F. Aman**

As a production engineer and later as a procurement engineer, Bob Aman has shared responsibility for production, procurement, and materials selection for the HP-85 Series personal computers and other portable computers. More recently, at HP's Inkjet Components Operation, he served as a project leader for design and fabrication of the equipment used for wafer assembly and its integration in the manufacturing process of the DeskJet pen. Bob came to HP in 1980, after working for some three years as a manufacturing engineer at Boeing Commercial Airplane Company. He earned a BSME degree from Oregon State University in 1977 and was awarded a professional engineering license in 1981. Bob was born in Silverton, Oregon. He is married, has two sons, and lives in Albany, Oregon. His hobbies include radio-controlled model airplanes, canoeing, fishing, and bow hunting.

**Brian L. Helterline**

Just after receiving his BSEE degree from Montana State University in 1987, Brian Helterline joined the Inkjet Components Operation of HP. As a product engineer, he was responsible for print quality testing of DeskJet print cartridges, ownership considerations for the print quality tester, and machine vision algorithms used for testing. Vision software and computer-control applications are his main professional interests. Born in Plains, Montana, Brian now makes his home in Salem, Oregon. He is married and enjoys playing basketball and tennis.

**Gregg P. Ferry**

An engineer at HP's Inkjet Components Operation, Gregg Ferry participated in the design of vision applications for the DeskJet printer. The project continues to be central to his design activities as he concentrates on electronic tools and computer-integrated manufacturing for the product. Both his BSEE (1973) and his master's (1976) degrees are from the California Polytechnic Institute at San Luis Obispo. Gregg was born in Minneapolis, Minnesota, and now lives in Corvallis, Oregon. He serves as a volunteer instructor for the Saturday Academy, an organization offering extracurricular instruction for high school students. Gregg likes traveling and bicycling, two avocations he once combined in a two-year biycycle trip around the world.

**Timothy S. Hubley**

A vision applications engineer and project leader at the HP Inkjet Components Operation, Tim Hubley has focused on evaluation of DeskJet print quality. In previous projects, he has worked as a product engineer on the chips for the HP 71B Handheld Computer and as an electrical tooling engineer for test equipment. He earned BS and ECE degrees from the University of Massachusetts in 1981 and joined HP the same year. He is a member of the Society of Mechanical Engineers and the Machine Vision Association. Tim was born in St. Charles, Illinois. He is married, has two children, and lives in Corvallis, Oregon. Among his favorite activities, Tim lists volleyball, tennis, and fatherhood.

**Mark C. Huth**

In the over seven years since he joined HP, Mark Huth has worked on manufacturing development and tool design for the HP 85 Computer, the HP 75C and HP 71B Handheld Computers, and inkjet print cartridges. On the DeskJet team, Mark helped design the optics system and develop the machine vision software. He received his BS degree in mechanical engineering from Virginia Tech in 1981. He was born in Boston, Massachusetts, and now lives with his wife and two sons in Corvallis, Oregon. Bicycling, volleyball, and rock climbing are Mark's favorite sporting activities, but he also likes potluck parties and enjoys the "small-town atmosphere of Corvallis."

**Robert A. Conder**

Vision applications and computer-integrated manufacturing are Bob Conder's focal professional interests. Since he joined HP in 1975, his product involvements have included a wide variety of HP calculators, handheld computers, and the ThinkJet and DeskJet printers. Bob is now a project manager and has been responsible for the coordination of vision projects and control systems associated with the DeskJet product. He is the author of an article about inkjet cartridges, and three patents are

pending on optics and alignment procedures he developed. Bob's BSEE degree is from the University of Utah (1975). He was born in Salt Lake City, Utah, and served four years as a sergeant in the U.S. Air Force. He is married, has a son and a daughter and lives in Corvallis, Oregon. His recreational interests include dirt-bike riding, sailing, and fishing.



**Robert Nicol**

As a manufacturing engineer, Rob Nicol developed a number of production methods for the HEDS-9000 encoder. Prior to that, he contributed to the development of the HEDS-9000 as an R&D engineer. He is now working on new optical-encoder applications. Before joining HP as an R&D engineer in 1983, Rob worked for the Santa Barbara Research Center, a subsidiary of GM/Hughes. His BSME degree is from the University of California at Santa Barbara (1983). He was born in Walnut Creek, California, is married and has two sons. He resides in Fremont, California. Rob is a member of the Society for Automotive Engineers and, in his spare time, has converted a conventional automobile to electric operation. He also enjoys serious music, sailing, and making beer.

**Mark G. Leonard**

Mark Leonard was one of the designers of the original HP encoder, and his collaboration with other development engineers led to product definition and design of the HEDS-9000. His professional interests are interdisciplinary and encompass electronics, computer science, and reliability physics. In the past, Mark has been involved in the design of a fiber optic receiver and high-voltage optocouplers. Before coming to HP in 1975, his responsibilities included failure analysis and computer programming. He is a senior member of the IEEE and a registered professional engineer and has published articles about optical encoders. Four U.S. patents are based on his designs. Mark attended Macalester College at St. Paul, Minnesota, where he received his BA degree in 1965. He was born in San Jose, Costa Rica, is married, and has three sons. He lives in Los Altos, California. His hobbies include woodworking and photography.

**Howard C. Epstein**

Since coming to HP in 1976, Howard Epstein has been responsible for the development of shaft encoder technologies. On the HEDS-9000 encoder, he led the definition and prototype stages and was the architect of the emitter/detector/lens system. Howard has since concentrated on extensions of the HEDS-9000 product. Before joining HP, he developed piezoelectric and piezoresistive transducers. Sensors and transducers are the focus of his professional interests, and he has published four papers about electromechanical and optical sensors. He is a named inventor on five patents. A registered professional engineer, Howard holds BA and MS degrees in physics (1965 and 1975) from the California State University at Los Angeles. Born in Los Angeles, he is married, has three teenage daughters, and lives in Los Altos, California. Howard serves on the board of the American Association for Ethiopian Jewry, an organization dedicated to the rescue of the ancient Ethiopian Jewish community. He enjoys playing handball, roller-skating, and surf fishing.

Hewlett-Packard Company, 3200 Hillview Avenue, Palo Alto, California 94304

HEWLETT-PACKARD JOURNAL

October 1988 Volume 39 • Number 5

Technical Information from the Laboratories of Hewlett-Packard Company
Hewlett-Packard Company, 3200 Hillview Avenue
Palo Alto, California 94304 U.S.A.
Hewlett-Packard Central Mailing Department
P.O. Box 529, Startbaan 16
1180 AM Amstelveen, The Netherlands
Yokogawa-Hewlett-Packard Ltd., Suginami-Ku Tokyo 168 Japan
Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive, Mississauga, Ontario L4V 1M8 Canada

Bulk Rate
U.S. Postage
Paid
Hewlett-Packard
Company



CHANGE OF ADDRESS: To subscribe, change your address, or delete your name from our mailing list, send your request to Hewlett-Packard Journal, 3200 Hillview Avenue, Palo Alto, CA 94304 U.S.A. Include your old address label, if any. Allow 60 days.

5953-8571