HEWLETT-PACKARD
JOURNAL

DECEMBER 1990

HEWLETT
PACKARD

HEWLETT-PACKARD JOURNAL

December 1990 Volume 41 • Number 6

Articles

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 presented 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.

In this Issue

Optical data storage technology offers several advantages over magnetic storage. Optical disks are more durable and reliable, have greater storage capacity, and cost far less per megabyte than magnetic disks. On the other hand, optical access time is currently much longer than magnetic, so don't look for optical disks to replace magnetic disks completely any time soon.

There are three optical storage technologies: compact disk read-only memory (CD-ROM), write-once-read-many (WORM), and rewritable optical. CD-ROM is basically the technology that is used for compact disk audio recording. The data is represented by pits on the disk surface, which are read by a laser system. WORM technology also stores the data as pits, but the pits are created by a high-power laser instead of by duplicating a master disk. Rewritable optical technology also comes in three types, but only one—magnetooptical—is currently practical. Data is stored magnetically on the disk. It is written and read optically using a laser system and can be erased and rewritten repeatedly without wear or degradation.

HP offers both CD-ROM drives and rewritable optical disk drives and autochangers. The article on page 6 describes the autochanger concept and architecture. Officially called the HP Series 6300 Model 20GB/A rewritable optical disk library system, the autochanger is the key element in a concept called direct access secondary storage, or DASS. The idea is that if you could have all of your data on-line, even archival data, you'd prefer that to off-line archival storage on reels of magnetic tape. The Model 20GB/A stores 20.8 gigabytes of data on 32 magnetooptical disks. Access time ranges from a fraction of a second if the required disk is in one of the two built-in magnetooptical drives to 15 seconds or less if a disk needs to be exchanged. Besides archival storage, the autochanger is designed for backing up large systems without operator intervention and for data-intensive applications such as electronic image management. The major autochanger design issues were the mechanical design (page 14), the servomechanism design (page 24), qualifying the vendor-supplied drives (page 35), and integrating the autochanger with the HP-UX operating system (page 11). Reliability was the overriding concern, since the autochanger is intended to serve as a vital link in a company's computer operations. It's designed for a lifetime of a million exchanges.

The HP Series 6100 Model 600/A CD-ROM drive (page 38) is designed for providing such things as software manuals for computer systems, large reference documents, training packages, and large-scale software distribution. Each disk can store over 500 megabytes of data. Design issues included the implementation of error correction (page 42) and software protection (page 49), integration of the drive with the HP-UX operating system (page 54), and the design of the controller board for the vendor-supplied drive mechanism (page 38). The ISO 9660/High Sierra Group standard format is used for recording data, so the Model 600/A can read non-HP CD-ROM disks and audio CDs recorded using that format.

In HP's PA-RISC architecture, printers, terminals, and personal computers connect with HP 3000 PA-RISC computers through remote multiplexers called data communications and terminal controllers, or DTCs. The host computer and the DTCs are connected by a local area network. For wide area communications, packet switched networks, both public and private, are now common, and the DTCs can connect directly to these networks, which are defined by an international standard, CCITT recommendation X.25. Asynchronous devices such as terminals and printers can connect to X.25 networks through hardware or software elements called packet assembler/disassemblers, or PADs. PAD support software allows the DTCs to communicate with PADs over the network. In the HP architecture, the PAD support software is assigned to the DTC rather than to the host HP 3000 system. This is done to maximize performance. The article on page 63 describes the design, development methodology, and testing of the PAD support software for the HP 2345A DTC. One of the test tools was a message machine that simulates the environment of the software under test. Designed using object-oriented concepts, the message machine is described in the article on page 74.

Copper beryllium (CuBe) alloy is widely used for spring contacts in the electronic industry. On page 88, Nguyen Hung of HP Singapore reports on an investigation of the dimensional changes of cold-drawn CuBe during aging at elevated temperatures. The results show that the changes are anisotropic and agree well with theoretical predictions.

December is our annual index issue. The 1990 index begins on page 81.

R.P. Dolan
Editor

Cover

Magnetooptical disk cartridges are shown with various mechanical parts designed for the HP Series 6300 Model 20GB/A 20-gigabyte rewritable optical disk library system.

What's Ahead

Lightwave component analysis with modulation rates to 20 GHz is the major topic in the February issue. The design of the HP 8703A lightwave component analyzer, the HP 83420A lightwave test set, and related products will be presented. Also featured will be the design of the HP 8153A lightwave multimeter, a modular instrument for optical power measurements and other basic fiber optic mesurements. HP VUE, a visual user interface for the Domain and HP-UX operating systems, will be the topic of the lone software article.

A Rewritable Optical Disk Library System for Direct Access Secondary Storage

This autochanger system can store up to 20.8 Gbytes of data on-line. Applications include archival storage, automated backup and recovery, and document storage and retrieval.

by Donald J. Stavely, Mark E. Wanger, and Kraig A. Proehl

HEWLETT-PACKARD MANUFACTURES a wide range of computer peripherals. Customers for these peripherals include not only users of HP systems, but also OEM customers and others who use HP peripherals with non-HP host systems.

Supplying peripherals to OEM customers has been a major initiative for Hewlett-Packard and has had a large impact on how we plan and evolve our business strategies. To be successful in the OEM business has required that we develop a broader and more timely understanding of the market than we had in the past. We feel that our experience as a system company gives us valuable insights into how our peripherals work in systems and applications to solve real customer needs.

HP's Greeley Storage Division is responsible for high-end secondary storage devices that are used for backup and archival storage on computers, mainframes, and networks of workstations. Our current product offering is a family of low-cost, autoloading, streaming, 1/2-inch GCR tape drives.[1]

As we looked to the future, we naturally focused our attention on advances in tape technology. Emerging products were using air bearings for media reliability, a thin-film 18-track head for very high transfer rate, and a compact tape cartridge for ease of handling. Initially, this technology seemed a good match to what our current HP and OEM customers needed. Customers were asking for faster backup to reduce planned system downtime—or more accurately, to keep from increasing their downtime as their disk storage requirements grew. They also need ever higher levels of reliability to minimize unplanned system downtime.

Unfortunately, simplistic market research—asking customers what they want—often yields only predictable and simplistic answers. They want what they have now, only faster, cheaper, more reliable, and so on. In other words, customers may be too close to their problems to see them from a new perspective.

We evolved a much more powerful market research process that consists of three steps. The first step is to gain a thorough knowledge of how customers do business. What applications do they run? How much disk space do they have? How do they do backup today? What else do they use tape drives for?

The second step of the process is to try to solve customers' problems in the abstract—matching available technologies to a high-level model of each customer's business. The last step is to present a coherent vision of the future back to our customers. In essence, we are trying to help them look past the limitations of today's solutions and help them architect the solutions of tomorrow. We call this developing an "imaginative understanding of user needs."

Fig. 1. *The HP Series 6300 Model 20GB/A rewritable optical disk library system stores up to 20.8 Gbytes of data on 32 optical disks. An autochanger automatically selects the correct disk and inserts it into one of two internal drives.*

In probing more deeply with both HP system customers and OEM customers, we found that their responses to our simplistic market research were indeed conditioned by the properties and limitations of tape technology. The truth is that customers don't like tapes. Tapes are inherently off-line devices requiring sequential access to data and operator intervention to handle the media. What customers really want is direct access to all of their archival data, without special utilities and without operator intervention. We call this concept direct access secondary storage, or DASS. When we clearly articulated this concept and fed it back to customers, we received consistently positive responses.

Rewritable optical disk technology, configured in an automated library, has exactly the right properties to meet this customer need for direct access to archival data. Optical disks are removable, rugged, reliable, and fairly inexpensive on a cost-per-megabyte basis. Transfer rates are competitive with many current tape products. And because it is a disk drive, an optical disk drive attaches to the host system using standard disk drivers and file systems. This can give direct, transparent access from current applications without modification. The connection between optical disks and secondary storage makes perfect sense, but it was not obvious to either customers or the optical drive vendors themselves.

The optical disk autochanger plays the other key role in the DASS concept. With many gigabytes of on-line Winchester-disk storage, a typical host system requires tens or even hundreds of gigabytes of secondary storage for backup and archival information. In the DASS concept, this secondary storage must be on-line—accessible without operator intervention or special recovery utilities. Rewritable optical drives in an autochanger configuration provide a cost-effective answer to the customer need for direct access to huge amounts of historical data.

Reliability is the single most important attribute of an autochanger. The customer perception is that autochangers are "mechanical nightmares" that are fascinating to watch at trade shows but frightening to consider as a vital link in a company's computer operations. It was for this reason that Hewlett-Packard chose to design and build its own autochanger mechanism.

The philosophy used to guide the development was that reliability should not be tested into a product, nor even designed in—it must be architected in. An architecture that minimizes complexity, followed by careful design and rigorous testing, is the only way to achieve a quantum leap in reliability.

Optical Disc Library System

The result of these considerations is the HP Series 6300 Model 20GB/A rewritable optical disk library system, Fig. 1. The Model 20GB/A combines the convenience and low storage cost of optical-disk technology with the massive capacity of a library system to provide on-line access to vast amounts of infrequently accessed information. The Model 20GB/A is a direct access secondary storage (DASS) device that fills the price/performance gap between high-performance hard disks and low-cost tape storage (Fig. 2). Because of its huge, 20.8-Gbyte storage capacity and low cost per megabyte (Fig. 3), the product makes it feasible to store information on-line that has traditionally been stored off-line, and to automate labor-intensive backup and recovery processes. It also greatly reduces the floor space required for archiving (Fig. 4).

The Model 20GB/A uses magnetooptical technology (see box, page 8). Data is stored on removable 5¼-inch disks. Optical disks are not susceptible to head crashes and are much more tolerant of magnetic interference than magnetic media. Fingerprints and small scratches have no effect on the data. Data can last over ten years without the retensioning or reconditioning that tapes require.

The Model 20GB/A consists of an autochanger, two magnetooptical disk drives, and 32 5¼-inch, 650-Mbyte optical disk cartridges in a deskside cabinet. A mailslot is provided for loading or removing disks. The autochanger automatically selects the appropriate cartridge and inserts it into

(continued on page 10)

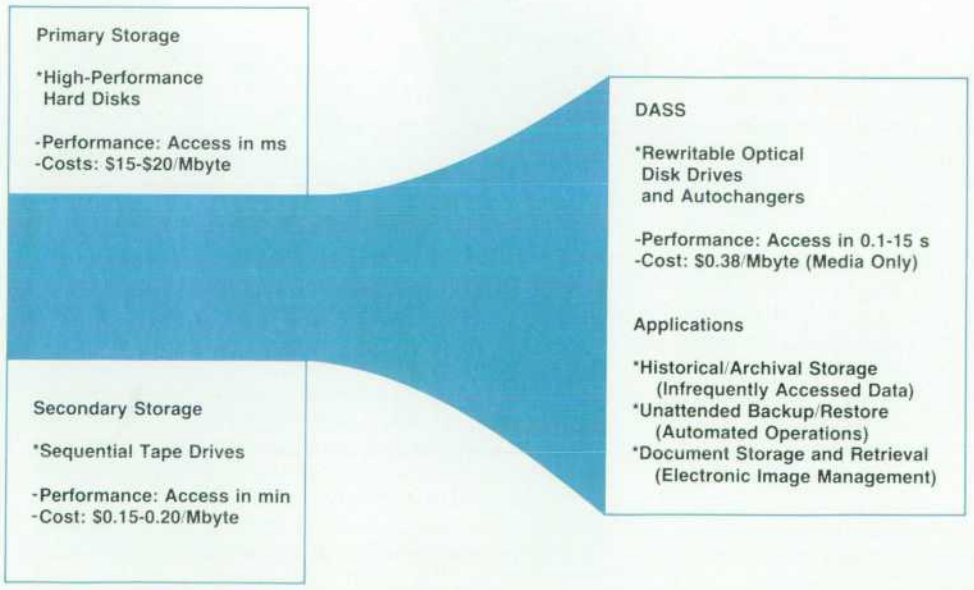


Fig. 2. *The optical disk library system is a direct access secondary storage (DASS) device, lower in cost than high-performance disk drives, but higher in performance than low-cost tape.*

Magnetooptical Recording Technology

Rewritable optical technology today encompasses three different methods:

- Magnetooptical
- Dye-Polymer
- Phase-Change.

The most durable and predominant technique in the market today is magnetooptical. This discussion will be limited to this technique, which is the method that Hewlett-Packard has chosen for introducing rewritable optical technology for direct access secondary storage.

Magnetooptical technology relies on the storage of information on a thin film of magnetic material. Like conventional magnetic recording, the information is stored on the media in the form of magnetic domains. The domains are aligned vertically, in contrast to most magnetic recordings today, which are based upon longitudinal magnetization. The important and significant difference comes from the fact that the processes of writing, erasing, and reading are performed with a light beam derived from a solid-state laser and associated optics, not by mechanical heads that come into contact or near contact with the recording surface. This attribute allows optical recording to have longer life and higher reliability than tapes and flexible disks. Optical disks are immune to the typical wearout modes that occur with contact or close proximity recording.

Recording

Thermomagnetic writing is the term used to describe the process of writing information on a thin magnetooptical film. The laser beam heat-modulates the magnetic film about its Curie temperature. The Curie temperature of a magnetic material is the temperature at which the material loses its coercive magnetic field. This occurs between 150°C and 200°C for typical magnetooptical thin films. When this occurs, the material loses all memory of its prior magnetization and can acquire a new magnetization as it cools in the presence of an external magnetic field.

The writing process is shown schematically in Fig. 1. The recorded information is stored on the magnetic medium by reversing a magnetic domain to store a one and by not reversing a domain to store a zero. Thus the precondition for writing information is for all domains to be initialized to the zero state. This means that, to overwrite data, an erase pass must be performed before the write pass to set up this initial condition of all-zero domain alignment. During the erase pass, the laser is turned on to heat the magnetic domains and an external magnetic field is applied in the proper orientation to change all of the domains to the zero state.

Data can be written on the erased track during a subsequent disk rotation. With the polarity of the external magnetic field reversed, the laser is turned on and off to heat only those domains that are to be changed to the one state. The external magnetic field required to erase or write data is supplied by a bias magnet which is typically positioned on the opposite side of the film surface from the optical head. This external bias magnet must have the ability to change magnetic polarity; therefore, it is typically an electromagnet or a permanent magnet that can be mechanically rotated to accomplish polarity changes.

When magnetooptical films are at room temperature, they typically exhibit coercivities of several thousands of Oersteds. This means that in the absence of laser heating, the magnetic field required to affect their state of magnetization is extremely large. Because of these high coercivities at operating and storage temperatures, magnetooptical records are less susceptible to damage from external fields than records on conventional magnetic storage materials such as those on flexible and rigid magnetic disks.

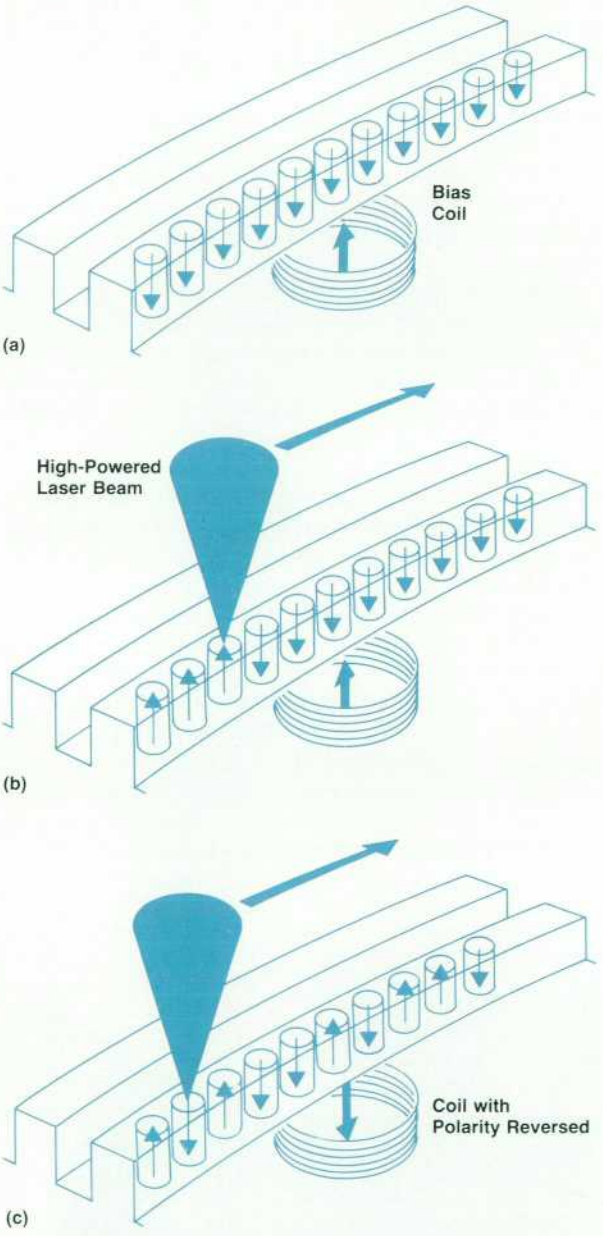


Fig. 1. *The magnetooptical write process. (a) All of the magnetic domains are magnetized north-pole-down. This all-zeros state is the precondition for writing. (b) The laser beam turns on for each domain that is to store a one. Heating the domain above the Curie temperature causes it to lose its previous magnetization and orient itself with the external magnetic field of the bias coil. (c) To erase the data, the polarity of the external field is reversed and the laser is turned on, returning all of the magnetic domains to the condition shown in (a).*

Readback

For data readout, information is extracted from the magnetooptical film by reflecting a polarized light beam off the magnetic film surface and detecting a change in the angle of polarization of the reflected beam. This physical phenomenon, upon which the magnetooptical rewritable technology is based, is known as the Kerr effect. It is manifested as a change in the state of polarization of light upon interaction with a magnetized medium. The amount of polarization rotation is small (less than one degree) but techniques used in film manufacturing can enhance the effect. In addition, a variety of detection and readout techniques have been developed to enhance the magnetooptical signal. As a result, good signal-to-noise ratios of 60 dB or more can be achieved.

Another magnetooptical readout alternative is based upon the Faraday effect. This effect is similar to the Kerr effect but relies on light transmitted through magnetic films. The interaction of the light with the film causes polarization state changes. This technique is not employed in the magnetooptical rewritable process primarily because of the low transmissibility of magnetooptical films and the difficulty of placing interactive optics on both sides of the media.

Magnetooptical Materials

Magnetooptical materials are composed of a rare earth element and a transition metal. Typical rare earth elements used in magnetooptical recording include gadolinium (Gd, $z = 64$) and terbium (Tb, $z = 65$). These rare earth elements are also called lanthanides. These elements are soft, gray metals that have good conductivity. As a group, the lanthanides are not very abundant. The most common lanthanide is cerium, which makes up only 3×10^{-4} percent of the mass of the earth's crust. The transition metals commonly used in magnetooptical recording include iron (Fe, $z = 26$) and cobalt (Co, $z = 27$). These elements contribute characteristics such as high melting temperature, good conductivity, and fairly high hardness. Alloys of rare earths and transition metals are amorphous and have been processed to achieve a high level of chemical stability. The transition metal provides the dominant magnetooptical interaction (Kerr effect) while the rare earth element helps to provide high vertical magnetic anisotropy.

Curie and Compensation Temperatures

The important parameters in processing magnetooptical films are the Curie temperature of the alloy (mentioned earlier) and the compensation temperature. The compensation temperature is the temperature at which the magnetization component of the transition element is equal and opposite to that of the rare earth element, so that the net magnetization is zero. The compensation point can be either above or below the ambient temperature. At the compensation temperature, since there is no net magnetization, the material cannot interact with external fields. Therefore, the coercivity is extremely high and the magnetic domains are very stable. For practical magnetooptical recording films in use today, the compensation temperature is kept well below the Curie temperature and the lowest operating temperature the film will see. The reason is that the interaction of the compensation point magnetic behavior can affect the requirements for Curie temperature recording. If the compensation temperature is in the region of operation, the magnetic properties change dramatically and can interfere with the designed magnetooptical recording process.

The compensation point for magnetooptical films is determined by the percentage of the rare earth element in the film. Typical percentages, for example, are for terbium to be below 19 to 20 atomic percent to keep the compensation temperature below ambient. If the percentage of terbium is higher, say above 22 to 23 atomic percent, then the compensation temperature can exceed ambient. At percentages greater than 27 or 28 atomic percent, there is no compensation temperature because the compensation temperature exceeds the Curie temperature, and magnetic properties above the Curie temperature dominate the film behavior.

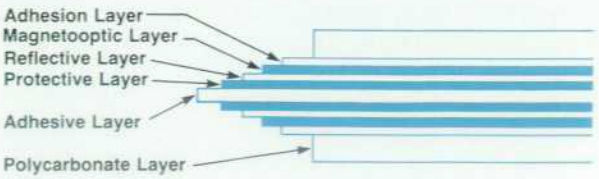


Fig. 2. *Magnetooptical disk construction.*

The Curie temperature is selected so that the laser light source can easily raise the magnetooptical film material to this temperature without exceeding the design limits on the laser power. A film with a lower Curie temperature requires less heat and therefore is more sensitive. Hence the Curie temperature controls the media sensitivity.

The Curie temperature for magnetooptical films is determined by the selection of the transition metal component. One way to control the Curie temperature is to adjust the ratio of cobalt to iron in the transition metal. As the ratio of cobalt to iron is increased, the Curie temperature is increased and the film sensitivity is decreased—more power is required to reach the Curie temperature.

Manufacturing

The manufacturing processes for magnetooptical disks have to take into account a wide variety of parameters. Important considerations include the following:

- Mechanical stability of the substrate, which is typically plastic (polycarbonate). Glass and aluminum have also been used. Some of the parameters of concern are warp, tilt, axial and radial runouts, and accelerations.
- Birefringence of the substrate, a condition in which the index of refraction is dependent on the polarization of the light.
- Dust protection. This is provided by a transparent layer that keeps dust, scratches, or other optical disturbances away from the focal plane of the recording surface.
- Surface reflectance control. This requires control of layer thicknesses and refractive indexes.
- Thermal characteristics, including thermal properties of films and surrounding structures and materials.
- Magnetic properties, which are determined by film composition and thicknesses.
- Protective coatings, such as dielectric barrier films for corrosion protection.

A typical cross section of a magnetooptical disk is shown in Fig. 2. The disk consists of two ten-nanometer-thick layers of magnetooptical film—one for each side of the disk—sandwiched between two polycarbonate disks. Dielectric material and adhesives separate and bond the layers.

Ed Sponheimer
Project Manager
Greeley Storage Division

one of the two internal drives. Operation is transparent to users, who see only a slightly slower response time when accessing optically stored data—approximately 100 milliseconds if the disk is already in a drive, or about 10 to 15 seconds if disks need to be exchanged. The HP-UX operating system recognizes each disk side as a 325-Mbyte mountable file system, so data access and software compatibility are the same as if the library system were a (slower) hard disk.

The Model 20GB/A conforms to ANSI and ISO specifications for continuous composite format 5¼-inch rewritable optical disks. This ensures compatibility with the HP Series 6300 Model 650/A stand-alone rewritable optical disk drive and the drives and media of other manufacturers. The system implements the Small Computer System Interface (SCSI) in asynchronous mode with separate IDs for both drives and the autochanger. The product is supported on HP 9000 Series 300 and 800 computer systems running the HP-UX 8.0 operating system.

Design Philosophy

The design criteria for the HP Series 6300 Model 20GB/A rewritable optical disk library system were focused on three points that we felt were essential to the successful launch of a peripheral with new functionality: minimizing time to market, making the product very reliable, and completing full system integration. The major design and architecture philosophies we used were high leverage of existing successful designs, design simplicity, and modular design with limited coupling between modules.

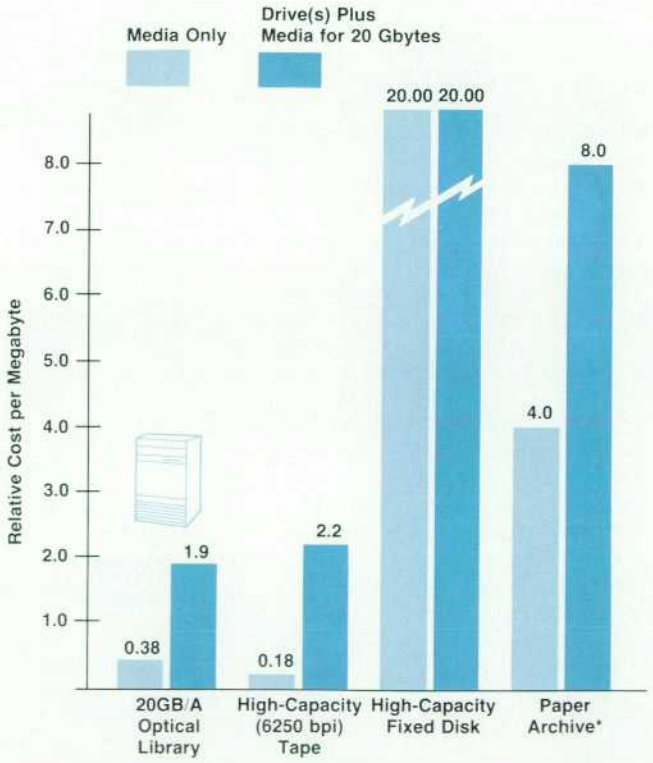
Leveraged designs allowed us to produce early breadboards rapidly with a high level of sophistication. In using leveraged designs, we also gained all the benefits of the engineer-years of effort that went into reliability engineering. We leveraged coupled servo architecture, motor/encoder design, a proprietary HP motor control IC, pulley and belt design practices and capabilities, and carriage and way design. We limited ourselves to off-the-shelf power supplies to decrease risk and tooling expense.

The design for reliability began with a study of failure rates on HP plotter products and the HP quarter-inch tape cartridge autochanger. We found that the predominant failures were associated with sensors, switches, motors, and solenoids. We next generated graphs of annual failure rate as a function of the number of sensors and as a function of the number of actuators. The architecture that followed from this analysis called for a minimum number of sensors and motors, and for a "passive payload" design, which means that no sensors or actuators were allowed on any moving parts. This strategy decreased the number of high-failure-rate parts and supporting parts as well, such as cables, connectors, and flexible circuits. It also eliminated flexing wires, which have fatigue problems. We were well aware that as printers have evolved in reliability, the hardest remaining reliability problem is the flexible cable going to the printhead.

The mechanical design of the autochanger is described in the article on page 14.

Another feature of the design is the servomechanism "sense of touch," which is tied directly to the passive

(continued on page 12)



*Varies with application. Conservative estimates are 0.01 per page and 0.02 per page with storage and retrieval.

Fig. 3. *Storage cost per megabyte for various alternatves.*

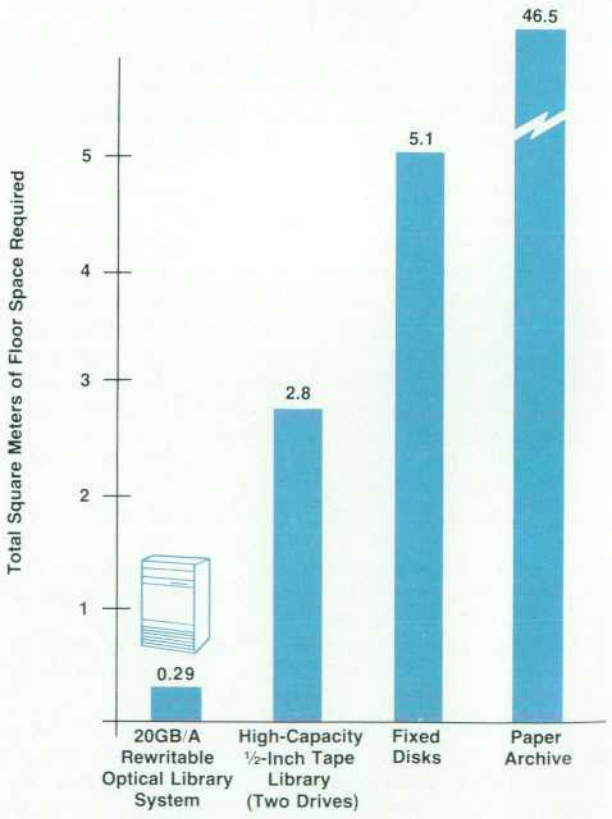


Fig. 4. *Storage space comparison for 20.8 gigabytes.*

Integrating the Optical Library Unit into the HP-UX Operating System

The HP Series 6300 Model 20GB/A rewritable optical disk library autochanger is unlike any other peripheral supported on the HP-UX operating system and therefore required a new approach to integration into the operating system. On one hand, its random-access attributes suggest a connection with the operating system that is disk-like. On the other hand, its need to share multiple disks with one or two drives hints at something that may need specific user or application support.

Although the design options were unrestricted, we wanted an integration method that would satisfy two overriding goals. First, the integration method should hide as much as possible the requirement to swap disks into and out of the drives. This transparency goal means that no special programs or utilities are required to access information on the autochanger. This holds for commands that rely on network services as well as commands that treat mass storage peripherals as raw devices. The second goal was that the integration method should have minimum impact (complexity, coupling, etc.) on the HP-UX operating system.

Design Choices

The accepted way of integrating WORM (write once, read many) autochangers relies heavily on the application. An application is provided with low-level control of the changer mechanism and low-level control of the drives. The application is responsible for swapping disks in and out of drives and tracking the location of disks. This method clearly does not allow transparency, but our solution needed to support this low-level control so that existing applications that already rely on it could be ported to HP-UX and so that other autochanger-specific utilities could be developed. The HP-UX ioctl system call is used to support these low-level commands.

A tempting way to model the integration is to view the entire autochanger as one large disk. This solution implies that the disk cartridges that make up this large disk travel as a group and remain in some logical order. Since disks in the autochanger are inherently removable, the administrative problems of keeping sets of disks together led us away from this solution.

The solution we settled on treats each side of a disk as an individual disk.[1] The system administrator is free to create file systems on these disks and mount them or access them in the raw mode using existing system calls such as read(), write(), and other utilities that use raw devices. A file system residing on an individual disk surface can be mounted anywhere in the directory structure of the HP-UX file system.

Neither existing commands nor application programs require modification to maintain their functionality. The file systems residing on cartridges maintain their NFS (Sun Microsystems' Network File System) functionality with other machines on the network. The file systems are also protected from power failure to avoid lengthy file system recovery processes.

By confining most of the changes to a driver, the goal of minimizing HP-UX changes was met. Fig. 1 illustrates the structure of the autochanger driver. The autochanger driver consists of two main parts: the surface driver and the changer driver. The existing disk driver is used to control the drives. This is the same driver that controls the stand-alone rewritable optical drive, the HP Series 6300 Model 650/A.

The changer driver provides low-level control of the autochanger mechanism. It accepts commands to move the disk in slot x to drive 2, to report whether there is a disk in slot 2, and similar tasks. The surface driver controls the swapping of disks and routes disk requests to the disk driver when the requested disk is in a drive.

The Swapping Algorithm

When requests for different surfaces occur, only one of those surfaces can be inserted into each drive. The other requests must be suspended. To avoid having these requests wait forever, we set a limit, called the hog time, on the time that a cartridge can be in a drive processing requests while other requests are waiting. Once this time expires, that cartridge is removed and the request waiting the longest is inserted. We have found that the hog time should be somewhat larger than the time required to exchange a cartridge. Twenty seconds has proved sufficient.

If a cartridge in a drive were to be replaced immediately when there are no additional requests for that surface, it is possible that shortly after the cartridge exchange is started, a request for the original surface could arrive. This could result in a cartridge swap for every request. To avoid this problem an additional limit called the wait time was added. This is the maximum time that a cartridge can reside in a drive without processing any requests while other requests are waiting to use the drive. Choosing the wait time too large increases the effective swap time of the autochanger. Making the wait time too small increases the chances of thrashing as a result of consecutive requests for the same surface. We found a wait time of one second to be sufficient to avoid the extra swapping in most instances.

Because we realize that certain configurations will require different hog and wait times, these values are configurable (via drive ioctl calls) while the system is running. For example, in a backup application, the hog time should probably be high, so that other processes won't seriously degrade the throughput. If an application program reads blocks of data and then processes that data for more than a second before it reads more data, the wait time should be increased to avoid swapping between reads.

Autochanger Driver Design

The design of the autochanger driver overcomes three problems:

- It avoids a file system check (the fsck command) of all the cartridges after a power loss.
- It avoids excessive swapping caused by the syncer.
- It supports the concept of asynchronous read and write operations.

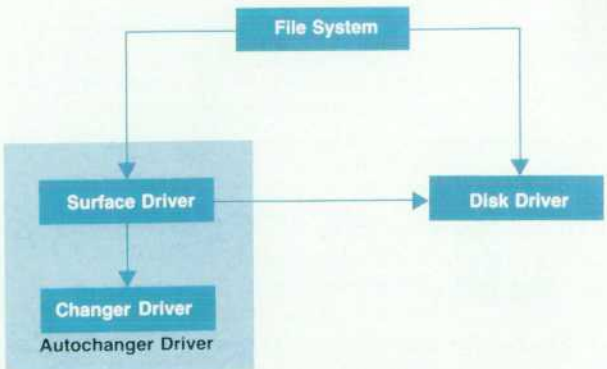


Fig. 1. *The autochanger driver consists of a surface driver and a changer driver. The existing disk driver is used to control the optical drives.*

Avoiding a Lengthy fsck. The superuser must execute the mount command for each surface to be accessed before users can perform file system operations to the autochanger. Normal Winchester disks require a file system check if the power cycles while the disk is mounted. Since we do not want to require the user to check every surface, which could take several hours, we make sure the file system on every disk is in a known, valid state before the disk is removed from a drive.

To do this, the autochanger driver uses the concept of virtually mounted devices. A virtually mounted device is not vulnerable to power failure corruption of its file system. When a device is virtually mounted, it is not directly connected to the rest of the file system. Only basic information about the file system and device is stored. The file system is available for use, but files cannot be accessed until the device has been physically mounted.

Cartridges in the autochanger that are waiting in their slots are only virtually mounted. When a file on a virtually mounted device is referenced by a user, the file system code uses the stored information to mount the device physically before the file is accessed. When the operations on that device are finished, the file system physically unmounts the device. This causes any buffers in the host that were not written to the device to be flushed. The file system is now in a state that would not require file system repair on a power failure. Cartridges only need to be virtually mounted when the system is first booted.

Avoiding Extra Syncer Swaps. To limit the number of modified buffers that haven't been written to the disk at the time a power failure occurs, there is a process that executes periodically that flushes these buffers. This process, called the syncer, schedules all the modified buffers to the drivers so they will be written to the disk. If a surface is removed from the drive without flushing all the modified buffers, the syncer will execute and require the cartridge to be reinserted. Having several cartridges mounted can cause excessive thrashing.

The solution is to write all the modified buffers for a surface before the cartridge is removed. This is done as part of the physical unmounting process.

Supporting Asynchronous Operations. Every block device driver in the HP-UX kernel must be able to support the concept of asynchronous requests. An asynchronous request essentially means that when the file system makes a request to the drive to perform say, a write to disk, the driver should queue the request and immediately return without actually doing the I/O. This tends to pose a problem in that, once the request is queued, some thread of execution must eventually complete the request. The normal method of doing this is through hardware-generated interrupts to the driver. However, this interrupt structure is absent in the autochanger driver.

There are two basic ways to solve this problem. One is to provide the extra thread of execution through the disk driver's interrupt calls. This adds extra complexity and coupling to both of the drivers. The method we have chosen is to create extra kernel daemons to provide the extra threads of execution necessary. Two daemons are used. A transport daemon is responsible for moving the cartridges, and a spinup daemon is responsible for spinning up the drive. The two daemons make it possible to overlap moves and spinups. For example, after a cartridge has been put into a drive and begins to spin up, the picker can be used to move another cartridge out of another drive.

The transport daemon flushes the asynchronous write operations to the disk before it removes a cartridge from a drive. The spinup daemon flushes all the waiting asynchronous requests to a new cartridge that has just been put into a drive. Thus, all asynchronous writes are eventually flushed out to a drive. If an asynchronous request arrives while a cartridge is currently in a drive, that request is passed to the disk driver immediately. This maintains the asynchronous function of the autochanger driver. One advantage of this daemon approach is that it is portable to other UNIX* architectures.

There are four processes in the autochanger driver: Accept Requests, Schedule Async Requests, and the transport and spinup daemons. Each process represents a separate thread of execution. Accept Requests executes any requests for surfaces already in drives. All other requests are queued. This process is in charge of determining if a swap is to occur as the result of receiving a request by checking the hog time and other conditions. The transport daemon only runs when Accept Requests determines that a swap should occur. It also performs the physical unmounting and makes sure the surface in the drive is put in a consistent state before it is removed from the drive. The spinup daemon is charge of spinning up the cartridge after it has been inserted. It also performs the physical mounting and processes any asynchronous requests waiting to use that drive. The Schedule Async Request process is executed when no synchronous requests are waiting for a surface. Its purpose is to satisfy the requirement that asynchronous requests return without waiting for any I/O to be performed. This process is not a daemon but is called by an interrupt from the wait timer in the spinup daemon. If the wait timer times out, Schedule Async Request is started if no synchronous requests are waiting to use the drive.

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

References

1. B. Thompson, D. Stolte, and D. Ellis, "A Transparent Integration Approach for Rewritable Optical Autochangers," *Proceedings of the USENIX Association Summer Conference*, 1990.

Daryl C. Stolte
Bruce A. Thompson
David Ellis
Development Engineers
Greeley Storage Division

(continued from page 10)

payload concept. The ability for the controller to sense forces and distance allowed us to remove physical sensors in the hardware. The architecture called for moving the sensors from the robot back into the information processing capabilities of the motor controller and encoder. Sense of touch algorithms allow the controller to sense whether there is a cartridge in a location, verify a move, and so on.

Another reliability enhancement is overforce sensing and error recovery. Using sense of touch, the robot stops motion before anything is broken. An error recovery algorithm is then invoked to clear the error condition and complete the move sequence. This design provides another chance to avoid a service call.

Details of the servomechanism design are in the article on page 24.

The design is highly modular with limited coupling of the modules. This effort paid off in many ways. The design areas could be assigned to engineers in a relatively un-

coupled way, as long as the interactions of the design modules were relatively well-defined. In these design modules, functionality was the key. Designers had the ability to simplify or redesign with measured effect on the rest of the project. This has been very fruitful as the project has continued in its life. As reliability concerns are raised, they can be addressed with minimum impact on other aspects of the design. This allows improvement with limited risk of introducing new problems in related designs.

Trying to achieve minimum system integration time and maximum flexibility led to the concept of mounting the autochanger in either a stand-alone HP rack or a standard 19-inch rack. This would allow the library system to fit into many different systems, both HP and non-HP. This design was adopted early in the project, and is now being born in a follow-on product.

In selecting a system as an initial host, we chose the HP-UX operating system because it provides enough power and flexibility to support this peripheral, and is almost an open architecture. This provided us with a number of options including developing a driver ourselves or developing a file level interface. Integration of the autochanger with the HP-UX operating system is the subject of the box on page 11.

Autochanger Architecture

One of the major goals in the design of the autochanger architecture was defining a growth path for future products. We did not want to lock ourselves into an architecture that would not allow us to meet the interface performance demands of the future. The optical drives that were going to be used in the autochanger communicated via the SCSI, and since there was a standard emerging for an SCSI autochanger command set, we decided that the primary interface of the autochanger would also be SCSI.

There are two drives in the standard configuration, and this means that the autochanger needs to use three SCSI bus IDs (two for the drives and one for the autochanger controller and mechanism). This posed the problem that if more than two autochangers were used by a host on a single SCSI bus, the available bus IDs (8) would soon be used up. We investigated some other architectures that would consume only one SCSI ID, with the autochanger and its two drives configured as logical unit numbers (LUNs) under that single ID. However, there were concerns about the performance degradation of doing the SCSI-to-SCSI command conversion for each LUN.

Even more ambitious than this architecture was the full file-level interface concept. This entailed defining an entirely new interface that interacted on a file level and totally hid the SCSI in the drives. With this concept, we felt we would have complete freedom to optimize the performance, throughput, and thrashing issues associated with an autochanger. On the other hand, this approach would also cause us to abandon the SCSI-II interface standard being adopted by most of the industry, and the use of a standard interface was seen as essential if this product was going to have a viable life as an OEM product.

Although future growth was a major concern, time to market was an even greater preoccupation. The autochanger was on an aggressive schedule, so we needed to be careful about not taking on too big a task for the time allotted. We were able to satisfy both goals by opting for the first architectural option described—the use of three SCSI IDs on the bus.

Autochanger Controller

The autochanger controller board is based on a 68000 microprocessor. The 68000 controls or oversees all processes in the autochanger. Because of the architecture chosen, the 68000 has no direct communication with the magnetooptical drives over the SCSI bus. The autochanger is meant to be an SCSI target device only, and at present does not support any initiator functions.

The 68000 operates with a time-sliced operating system whose primary function is to control the two servo loops of the Y and Z motors. Operation of these two loops can consume up to 70% of the processor's 12-MHz bandwidth. The remainder of the processor's time is involved with command interpretation and the overseeing of all other autochanger functions.

Commands to the autochanger can come from one of three different sources: the SCSI, an RS-232 port, or the front panel. The SCSI is the primary means for controlling the autochanger. The RS-232 port is primarily for diagnostic purposes, although it can also be used as the primary interface for the autochanger controller. The front panel is the personal, direct user interface. Commands can be entered through each of these interfaces, and although their formats differ, they are all processed through a common control flow in the 68000 firmware.

The hardware implementation of this architecture is highly integrated and uses either multifunction or intelligent off-the-shelf parts. The SCSI is managed by a proprietary controller IC, which frees the processor from all but the most necessary processing tasks of the SCSI bus. The RS-232 port is managed by a multifunction peripheral chip, which also handles all the interrupt vectoring and generates various timers used by the controller. The front panel is controlled by an 8051-type microcontroller, which manages all the key presses and is responsible for updating the vacuum fluorescent display. The low-level servo processing is serviced by another proprietary IC, which manages the duty cycle of each motor and monitors the position encoder information.

The controller board also contains 16K words of nonvolatile RAM. This RAM is used to store critical state information and positional parameters for use in autochanger error recovery and powerfail conditions. The nonvolatile RAM also contains certain configuration parameters and certain logging values that constantly reflect the age and health of the machine.

References

1. J.W. Dong, et al, "A Reliable, Autoloading, Streaming Half-Inch Tape Drive," *Hewlett-Packard Journal*, Vol. 39, no. 3, June 1988, pp. 36-42.

Mechanical Design of an Optical Disk Autochanger

The autochanger moves 32 disk cartridges between two magnetooptical drives and two stacks of storage positions using only two motors and three optical sensors.

by Daniel R. Dauner, Raymond C. Sherman, Michael L. Christensen, Jennifer L. Methlie, and Leslie G. Christie, Jr.

THE MECHANICAL DESIGN of the autochanger mechanism for the HP Series 6300 Model 20GB/A rewritable optical disk library system posed several technical challenges, including architecture, reliability, physical size, and schedule. The system holds 32 optical disk cartridges and has two magnetooptical disk drives. The magnetooptical disks are rewritable. Each cartridge holds 650 Mbytes of data; however, only 325 Mbytes is accessible at a time because the drives are single-sided. The total capacity of the library system is 20.8 Gbytes. The system runs on a single-ended SCSI asynchronous bus, which conforms to the SCSI II standard established for autochangers. The average access time to load a disk from a storage position to a drive is seven seconds.

The mechanical architecture of the autochanger excludes any electrical components, cables, or connectors on the moving parts of the mechanism. This "passive payload" concept was chosen to maximize product reliability. The design team set a goal at the onset of the project to have an absolute minimum of sensors, solenoids, and motors. The final design has only two motors, no solenoids, and three optical sensors. The sensors are used in the vertical calibration of the system and in the mailslot.

Physical size was determined early in the product design to allow use in two orientations. In the normal orientation, the autochanger fits into an HP rack. It can also be laid on its side and used in an industry-standard 19-inch rack. This two-orientation requirement established the height, width, and depth of the product. Meeting these space constraints was a persistent challenge in the design of the

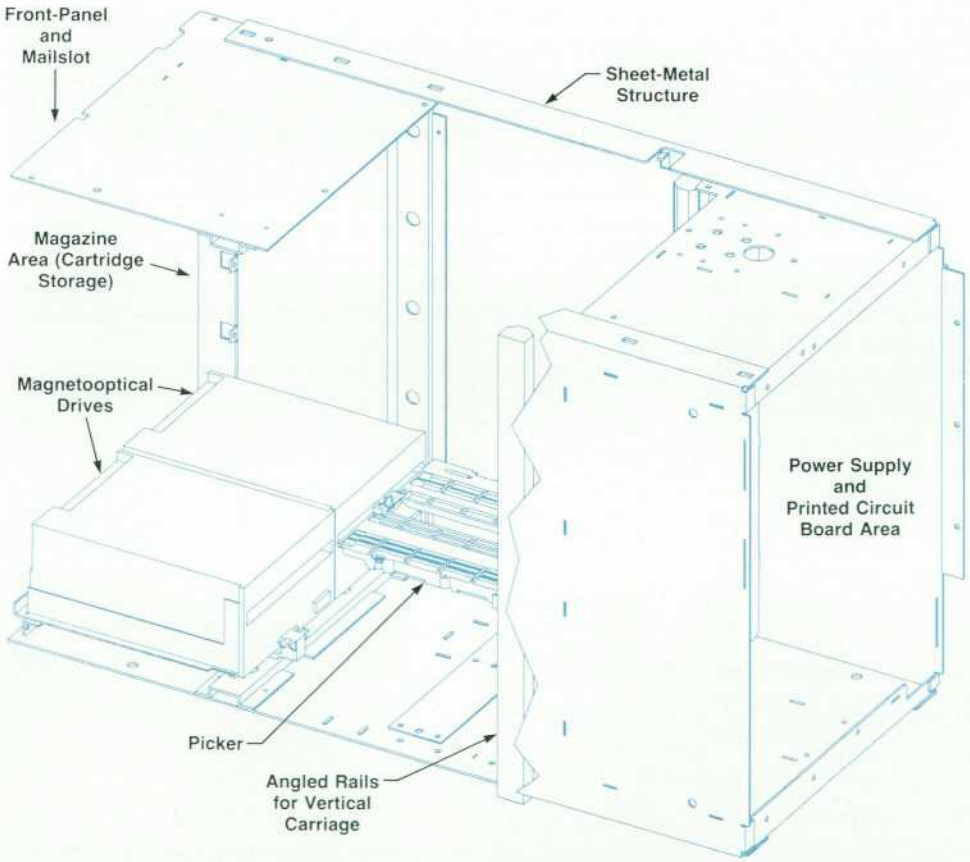


Fig. 1. *Mechanical layout of the HP Series 6300 Model 20GB/A rewritable optical disk library system.*

subsystems.

Adding to these design challenges were the need to ensure HP quality and the time constraints of an aggressive schedule.

Fig. 1 shows the mechanical layout of the autochanger.

Mechanical Functions

There are two basic mechanical functions of the product. First, the user can load or remove a cartridge via the mailslot. Second, the cartridges are moved from storage slots and the mailslot to drives and vice versa via the picker mechanism. These functions are implemented using two motors and associated subsystems. All movement in the product has been grouped into two types: Y or vertical motion and Z or horizontal motion.

Y Motion. All movement up and down along the vertical ways is labeled Y motion. It is driven by a dc servo motor and a vertically mounted leadscrew. A small toothed belt drives the vertical leadscrew through a reduction gear. The vertical carriage is attached to the leadscrew.

The vertical carriage is made up of the horizontal carriage, picker, and translate mechanisms. All of these mechanisms are powered by a toothed belt called the T belt.

Z Motion. The Z or horizontal plane is the plane in which the following motions occur:

- Plunge. The picker plunges to get a cartridge from a drive or slot or to put a cartridge into a drive or slot. The picker is the cartridge carrier, that is, the device that holds a cartridge that is being moved between a storage slot and a drive.
- Flip. The picker is caused to rotate 180 degrees.
- Translate. The system has two stacks of cartridges. In a translate move, the Y and Z systems are positioned to move the picker and the horizontal carriage—on which the picker is mounted—from stack to stack. Translate motions occur only at the lowest vertical position of the vertical carriage.
- Mailslot Actuation. This is a special plunge with picker side and vertical positioning.

All Z motion occurs within or as part of the vertical carriage. All Z motion is driven by the T belt, which is attached to the Z motor and oriented perpendicularly to the vertical carriage assembly.

The plunge occurs in the picker mechanism. This motion is driven by the T belt through a gear attached to the picker leadscrew. The flip is required because the disks are double-sided and the drives are single-sided. The translate occurs through a special combination of Y position and release mechanisms on the vertical carriage. When all of the proper conditions are met the Z motor will drive the horizontal carriage from one stack to the other. Mailslot actuation occurs at a particular vertical height, sensed by mating actuators on the picker mechanism and the mailslot. When all of the proper conditions are met a plunge motion of the picker actuates the mailslot.

Vertical Carriage

As described earlier, the cartridge holder (picker) must be able to move in the vertical direction to any magazine or drive slot. It must also be able to move to one of the two horizontal positions. The vertical carriage is the mechanism that constrains these motions. It is designed to hold the end of the cartridge holder in close tolerance despite variations of the sheet-metal structure that forms the enclosure for the product and to which all of the other parts are mounted and referenced. The vertical carriage is light in weight to reduce dynamic forces. It is designed to be installed and removed easily and to have a high degree of reliability. The biggest challenge in its design proved to be designing it to fit into a 375-mm box structure.

The vertical carriage is shown in Fig. 2. It is guided in its vertical motion by a set of angled rails, which attach to the structure, as shown in Fig. 1. The vertical carriage consists of:

- A set of bearing blocks with roller bearings, which roll on the rails
- Plastic carriage blocks, which hold the bearing blocks, the T belt pulleys, and the horizontal rod and way
- The horizontal rod and way, which provide the translate means for the horizontal carriage and picker.

In the breadboard design, a 0.75-inch-diameter rod and a linear bearing were used for the vertical transport. However, because of the volume limitations, this design proved difficult to implement. To solve this problem and meet manufacturing requirements, the rail and roller bearing approach was chosen. The main problem this design faced

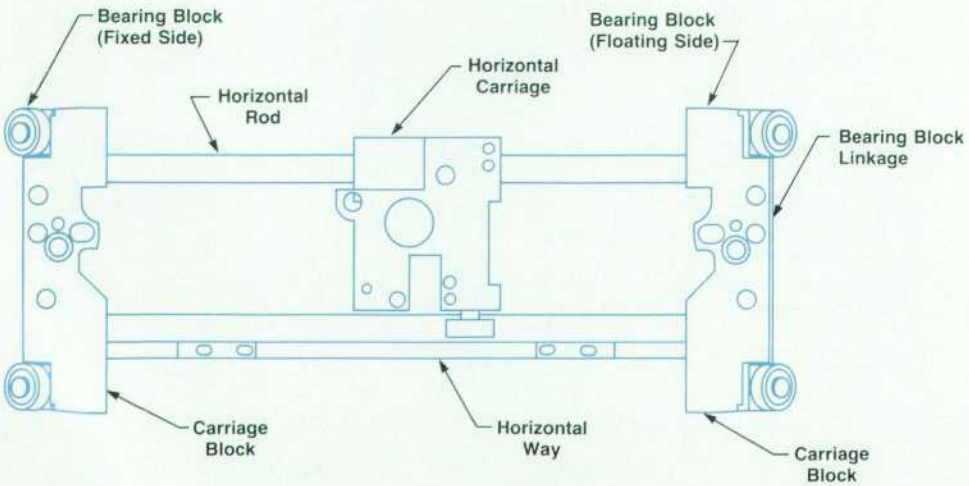


Fig. 2. *Vertical carriage structure.*

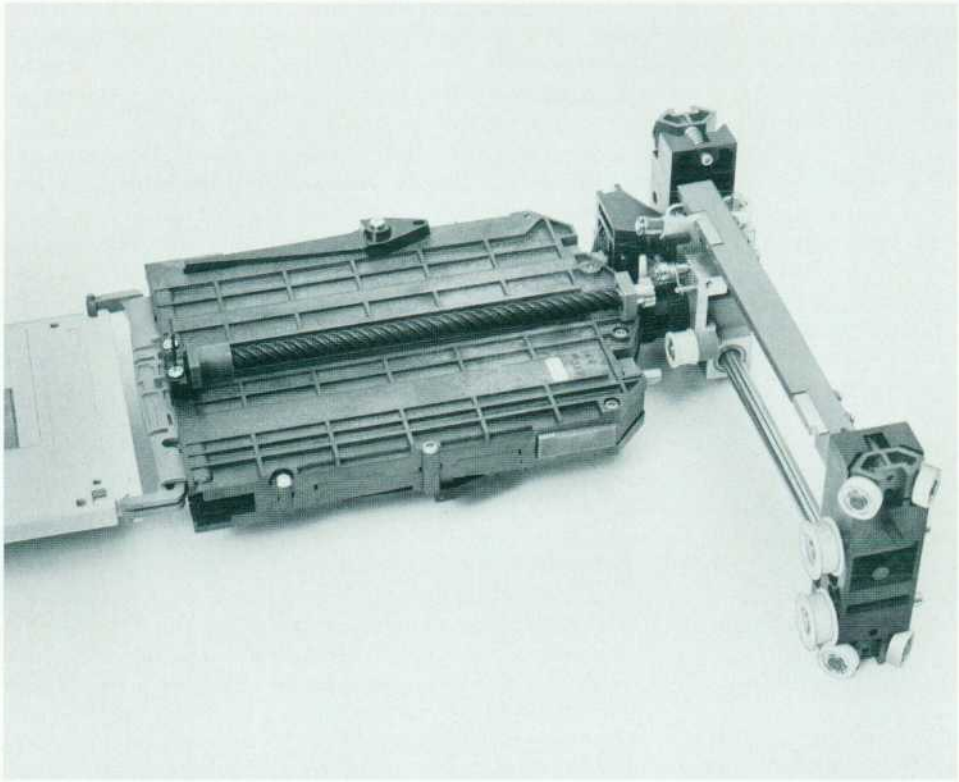


Fig. 3. *Horizontal carriage and picker mounted on the vertical carriage.*

was the variations of the sheet-metal structure. To account for these variations, the right-side bearing blocks are constrained in the carriage block but are spring-loaded (floating) between the vertical rail and the carriage block. This allows the assembly to correct for up to a millimeter of structure variation and still maintain the reference against the left rail, which always serves as the vertical reference surface.

A steel bearing block linkage is used to stiffen the vertical carriage assembly relative to the vertical rails. The bearing block linkage ties the two floating bearing blocks together to ensure that their motion always acts to tighten the vertical carriage within the vertical rails.

Because of the limited volume, the left-side rail has to perform several functions. Half-inch holes in the bottom of the extruded aluminum rail provide for mounting the bearings and shaft. This assembly has space for gearing in the back and the drive gear for the T belt in the front. At the top, a quarter-inch slotted hole in the rail and a plastic slider that fits around it provide a belt tensioner. The top T belt idler pulley is placed through the slider and rail, and is spring-loaded upwards to provide proper belt tension. Because the belt places a moment on the slider, it will lock up when momentary high forces are encountered.

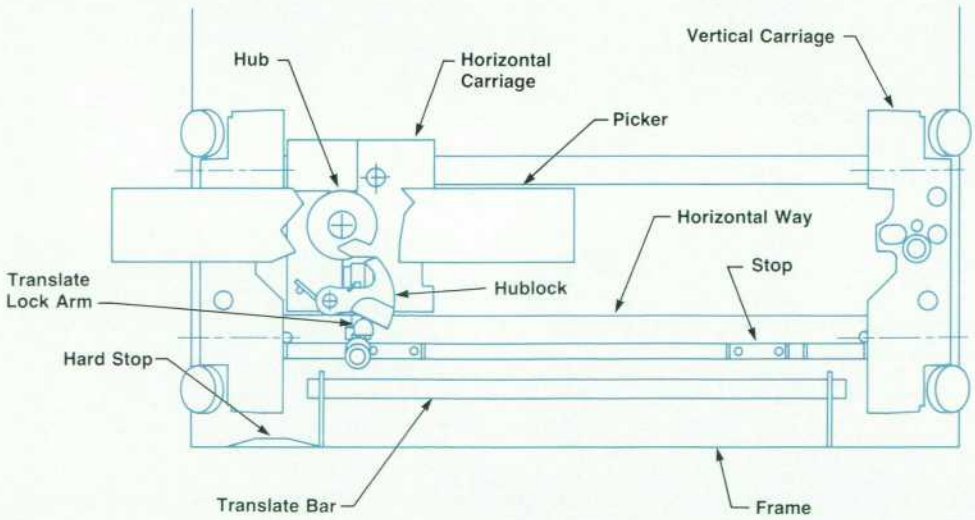


Fig. 4. *Translate mechanism.*

This keeps the belt from slipping when the drive system encounters the high force spikes sometimes seen during magazine and drive insertions.

The bearing blocks provided several design challenges. The first problem was running the bearings on the aluminum rails. This was very noisy and caused the aluminum to wear. Plastic tires were placed on the bearings, but developed flat spots in storage temperature testing. Plastic with better creep qualities was tried, but showed fatigue failures short of the required life.

For the final design, the plastic is Delrin and the tires are redesigned to increase their surface contact area. The bearing blocks were originally designed to be made of aluminum, but the aluminum tended to gall while sliding in the carriage blocks, and was much higher in cost than initially expected. Therefore, an all-plastic design was conceived, using bearing blocks made of polyphenylene sulfide with 40% glass. This design works much better. It results in lower friction, lower wear, better tolerances between mating parts, and a significant cost savings.

The carriage blocks tie the horizontal rod and way together structurally, hold the T belt pulleys, and provide the sliding constraints for the bearing blocks. Glass-filled polycarbonate helps reduce the carriage blocks' weight and cost. Because the carriage blocks are the stops for translate motions of the horizontal carriage, the distance between the blocks is set by machined features in the horizontal rod and way, which are held securely in place by crush bumps in the plastic blocks and then bonded for added rigidity. Once the vertical carriage is installed between the side rails, the carriage blocks cannot separate.

The horizontal rod is a three-eighths-inch hardened stainless-steel rod. The horizontal way at the bottom of the vertical carriage is a machined aluminum L-section bar. The top of the L section is a track for roller bearings on the horizontal carriage and the bottom of the L section provides a latch that holds the horizontal carriage in the appropriate translate positions. The aluminum way also provides a mounting surface for a portion of the translate lock assembly. This way was originally to be extruded, but the added machining made this less cost-effective than a completely machined part.

Horizontal Carriage

The horizontal carriage supports the picker and translates it from one cartridge stack to the other. The support structure for the horizontal carriage allows linear motion in one axis while excluding linear motion in two axes and rotational motion in three axes. A rigid structure is necessary to ensure proper alignment of the picker for cartridge exchanges.

The horizontal carriage is supported by the following parts of the vertical carriage: the horizontal rod, the horizontal way, the two carriage blocks, the four bearing blocks, and the bearing block linkage on the spring-loaded (right) side. Fig. 3 shows the horizontal carriage and picker mounted on the vertical carriage.

A machined aluminum casting controls picker alignment and serves as a mounting surface for the flip support, the translation lock assembly, the hub lock assembly, two idler shafts, two bearings for the main picker shaft, two tire shafts, and two linear bearings for translation capabilities. Rigidity and spatial concerns top the list of design requirements.

The hardened, ground, steel horizontal rod supports the horizontal carriage linear bearings. Two tires on roller bearings, mounted beneath the horizontal carriage, ride on the front and rear surfaces of the aluminum horizontal way (L bar) to control rotation about the rod.

Translate Mechanism

The translate motion—moving a cartridge from one side of the autochanger to the other—requires a combination of the vertical and horizontal motions. There were six design goals for the translate mechanism. First, in line with the passive payload concept, the movement must only use the two servo motors and require no additional sensors or solenoids to clutter a clean and reliable design. Second, the mechanism must be reliable to one million cartridge exchanges. This requires a translate mechanism life of 1.5 million translates because, on the average, there are 1.5 translates per exchange. Third, no more than 6 mm of Y motion can be used to actuate the translate mechanism. Fourth, the design must minimize wear through proper selection of materials and geometry. Fifth, the design must be fault tolerant. This fault tolerance must include the ability to recover from power failures at all times. Sixth, the design must allow translates at both the top and bottom vertical positions to provide architectural flexibility, although in the current product, translates only occur at the bottom.

The translate mechanism (see Fig. 4) consists of the vertical carriage including the horizontal carriage and the horizontal way with adjustable stops, and the translate lock mechanism. The translate lock mechanism (Fig. 5) consists of:

- A translate lock arm, which pivots in and out of a slot in the vertical carriage's horizontal way
- A hublock, which can engage a notch in the picker hub to prevent the hub from rotating
- A translate bar fixed in the structure, which pushes the

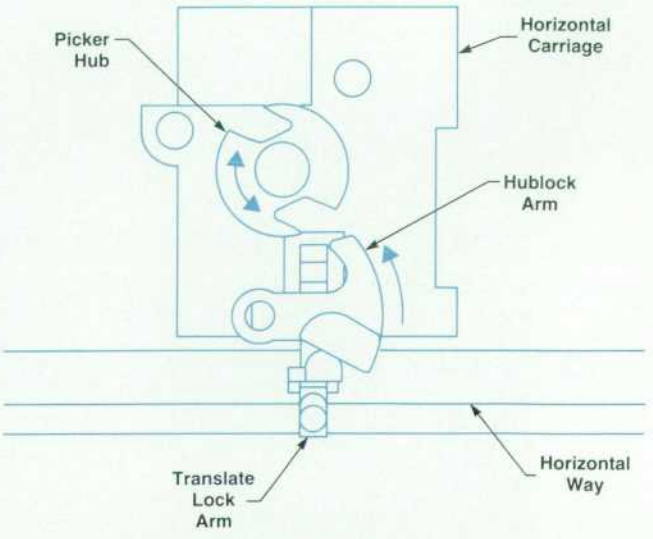


Fig. 5. *Translate lock mechanism.*

lock open.

Both the translate lock arm and the hublock are mounted on the horizontal carriage. They are independently spring-loaded downward for the normal case of the horizontal carriage being locked in place and the plunge motion allowed.

The translate lock mechanism has to hold the horizontal carriage rigidly fixed normally, and ensure that only one motion at a time can occur, either translate or plunge. Ensuring only singular motion is necessary because there are no sensors to tell the system what the picker and horizontal carriage are actually doing, so the servo could conceivably get "lost" without this stipulation. Fig. 6 shows the two basic locked positions: plunge allowed and translate prevented or translate allowed and plunge prevented.

The translate lock arm pivots on a horizontal axis contained within the horizontal carriage. The translate lock arm is spring-loaded to pivot downward, which forces the arm into a slot in the horizontal way and locks the horizontal carriage, and therefore the picker, to one side of the vertical carriage or the other. To minimize wear from the sliding motions, the translate lock arm has a needle bearing on its far end for contacting the translate bar and a molded wear pad closer to the center that contacts the hublock. The translate bar is rigidly attached to the sheet-metal structure and is the contact surface that actuates the translate lock arm. The hublock is another downwardly spring-loaded, pivoting arm that is responsible for locking the hub when actuated by contact with the translate lock arm. The hub is part of the plunge mechanism so that locking the hub results in locking the plunge mechanism. Locking the hub locks the horizontal carriage to the T belt so that the Z servo can move the entire horizontal carriage instead of just actuating the plunge motion.

The translate movement can best be described if split into four phases. In phase 1, the Z motor lines up a slot in the hub with the hublock, allowing for its eventual insertion. The Y servo lowers the entire vertical carriage assembly, causing the translate lock arm to contact and be pivoted upward by the translate bar. The translate lock arm in turn contacts the hublock, moving the hublock into the hub slot and locking the hub and the plunger. At the end of phase 1, both the plunge and translate motions are locked. This overlapping of the two locks is a reliability feature that prevents the Z servo from freewheeling and losing track of where the picker is. The servo is always in positive control.

In phase 2, when the hub is securely locked, the translate lock arm is lifted free and clear of the stops on the horizontal way to allow the Z servo to translate the horizontal carriage across the vertical carriage. Phase 2 is completed when the Y servo saturates* as a result of the vertical carriage's hitting the hard stop of the translate bar at the bottom of the structure, thus ending the vertical movement. No sensor is required to end the vertical movement, thus contributing to reliability and simplicity.

In phase 3, the Y servo is stationary while the Z servo drives the horizontal carriage from one side to the other. The needle bearing on the translate lock arm rides on the translate bar. This allows the tab on the translate lock arm to clear the horizontal way along the entire path. The design has to be tolerant of a possible power failure, since it would be very easy to get into an unrecoverable position at this time. To this end, the translate lock arm geometry is such that if the vertical carriage rises, as it would after a power failure, the Z servo can still pull the horizontal carriage to one side and have the tab on the translate lock arm lock into the stop. This is another major reason why the hublock and the translate lock can never physically be unlocked at the same time, even in the worst-case tolerance conditions. Phase 3 ends with the Z servo saturating with the horizontal carriage against the opposite side of the vertical carriage, thereby finishing the translation part of the move.

The Z servo continues to saturate throughout phase 4, while the Y servo moves the vertical carriage upward. This movement allows the translate lock arm to drop into the beveled stop in the horizontal way, which locks the horizontal carriage. The hublock does not follow the translate lock arm downward because it is held in place through friction by the saturating Z servo. This ensures that the horizontal carriage is in direct contact with the side, allowing reliable locking of the translate lock arm. A bevel in the stop eliminates the possibility that any burr in the lock arm will cause it to hang up. The stop is adjusted during assembly to ensure that the horizontal carriage does not float sideways in the locked position. When the vertical

*Servo saturation means that the torque output of the servo motor, which is calculated from the motor voltage and the motor torque constant, has exceeded a threshold. Servo saturation is also called force sense of touch.

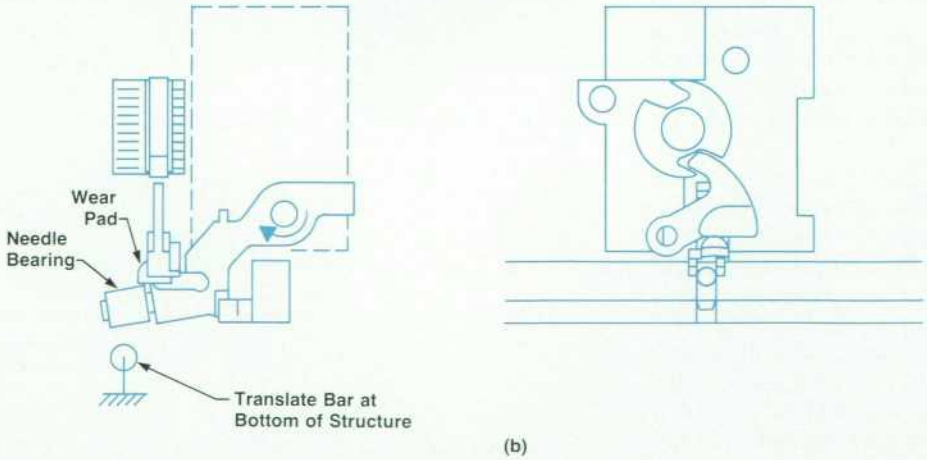


Fig. 6. *(a) Translate lock arm locked into horizontal way. Plunge motions allowed. (b) Hublock arm locked into picker hub. Translate motions allowed—no plunge motions.*

carriage has moved vertically far enough to allow the needle bearing on the translate lock arm to separate from the translate bar, the Z servo comes out of saturation and allows the hublock to fall back down onto the translate lock arm. The translate move is now complete, freeing the autochanger to perform whatever other moves are requested.

Picker

Like the rest of the autochanger, the cartridge retrieval mechanism is designed with simplicity in mind to help meet both reliability and cost goals. The mechanism requires four degrees of motion within a small form factor: in/out plunge, grasp/release cartridge, side-to-side translate, and 180-degree flip. In spite of all the motions, it was felt that the picker had to have a passive payload, that is, no motors, solenoids, or sensors on the moving platform, for the highest possible reliability.

Fig. 7 shows the basic layout of the mechanism. It consists of six main components or subassemblies:

- The vertical carriage, which connects to the vertical leadscrew, providing the up/down motion
- The horizontal carriage, which holds the cartridge picker mechanism, flip latch, and translate lock arm, and translates from side to side
- The picker hub, which converts the belt motion to picker plunge motion
- The picker itself, which can grab and hold a cartridge
- The translate lock mechanism, which either holds the horizontal carriage fixed or allows it to translate from side to side
- The flip latch mechanism, which holds the picker flat during plunges but can allow the picker to be flipped 180 degrees.

The picker hub is a single plastic molded piece with three functional sections. The back portion is a pulley, which the belt from the motor engages to provide all the power to the horizontal carriage and picker. The front is a gear, which mates with a smaller gear on the end of the picker's leadscrew to convert the belt motion to picker plunge motion. Finally, the center section has two cutouts 180 degrees apart which are used by the translation latch to lock the hub. The belt is the only power source for the entire picker mechanism, and the encoder on the motor moving the belt is the only method of sensing anything on the horizontal carriage.

The picker mechanism design was heavily influenced by space constraints. The width of the sheet-metal structure limited both picker width and picker height (because of flips). Short structure-length constraints and long minimum plunge depth requirements necessitated a compact cartridge grasping mechanism. In addition, the front-panel openings in the magnetooptical drives limited where the picker could grab the cartridge and required that whatever inserted the disk had to go well past the drive front panel. Another concern was how to handle misalignments, particularly as the cartridge was loaded into the drive.

With these concerns in mind, the picker was designed to mimic a hand pushing a cartridge into a drive. "Fingers" grab the cartridge from either the drive or a magazine, and the "thumb" pushes the disk back into place. The leadscrew's nut floats within the thumb, providing the muscle. (The nut is not rigidly attached to the thumb to allow for leadscrew runout and general part variations.) The finger mounts into the thumb, which in turn rides in a plastic shell that has tracks to guide the finger to grab or release the cartridge. Identical thumbs, fingers, and shell halves are used on both sides to form the whole grasping mechanism. Fig. 8 shows the two paths the fingers can take depending on their initial position. The fingers are spring-loaded to a normally rotated-in position. When the fingers

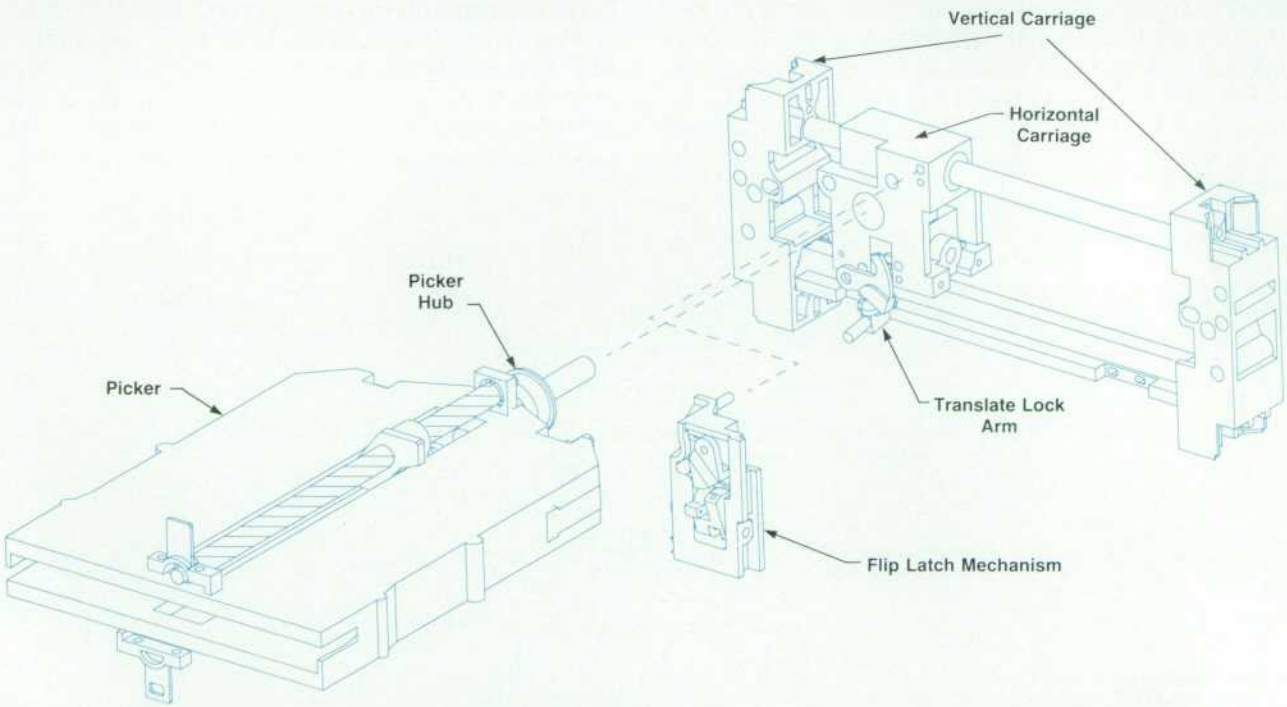


Fig. 7. *Cartridge retrieval mechanism.*

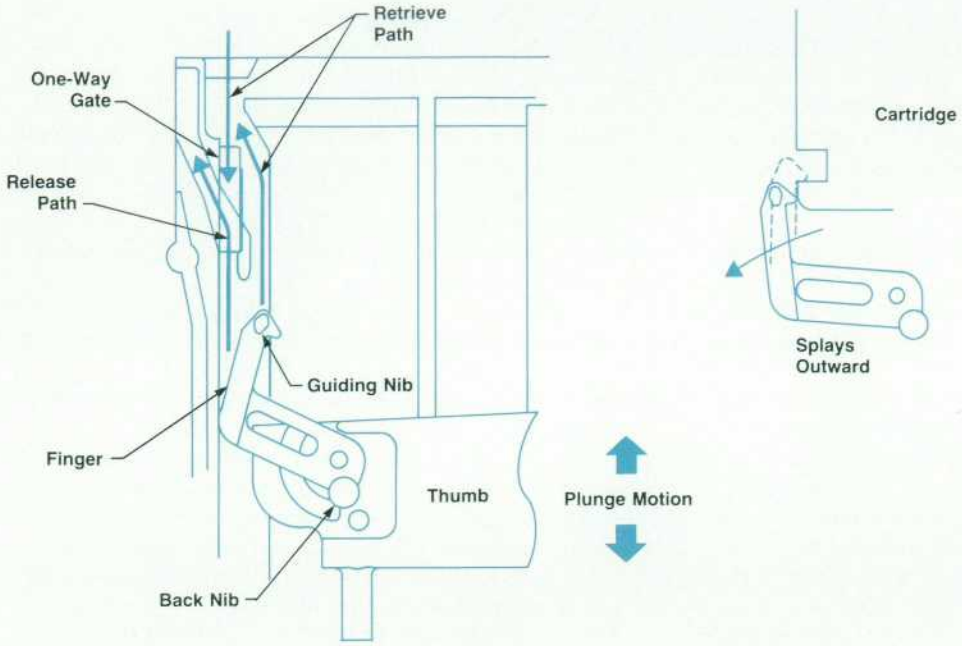


Fig. 8. *Picker finger and thumb mechanism.*

move out and contact the cartridge, they are splayed outward and grab the disk. In the track, a one-way gate turns the system into a type of mechanical flip-flop. This gate, a molded spring, is forced downward as the finger travels over it from the drive side, but will not move down when the fingers come from the picker side. Thus, with the next plunge, the fingers must follow the release path and are rotated out to release the cartridge. The center of the thumb then pushes the cartridge to its final position. As the thumb retracts, the fingers spring around again to their normally closed position, ready to grab a cartridge on the next plunge.

Materials were an interesting challenge in the picker design. The one-way gate is made of Ultem, which has high strength, good wear qualities, and low creep—important characteristics for a preloaded, highly cycled spring. The fingers, thumbs, and sleeves all require low-wearing materials, and since these parts slide against each other, each has to be of a different material. The fingers also require high strength, so they are made of 35% long glass, 15% Teflon polycarbonate. The thumbs have the most parts sliding against them (leadscrew nuts, sleeves, and fingers) and require good flatness, so they are molded of 30% glass, 15% Teflon Nylon 6/10. The sleeves, which are part of the electrostatic discharge path for any charge that might build up on the picker and affect the cartridge, are molded of 10% carbon, 15% Teflon polycarbonate.

Flip Mechanism

The objectives in designing the flip latch mechanism were to use already existing motion and to limit the flip to 180 degrees. Using the rotation of the picker hub without moving the leadscrew satisfies the first objective, and smart hard stops satisfy the second. Fig. 9 shows the main parts of the latch: the pivot arm, the release arm, and the cam. In normal picker plunge motions, a nib on the rear of the picker shell is trapped between the pivot arm on the bottom

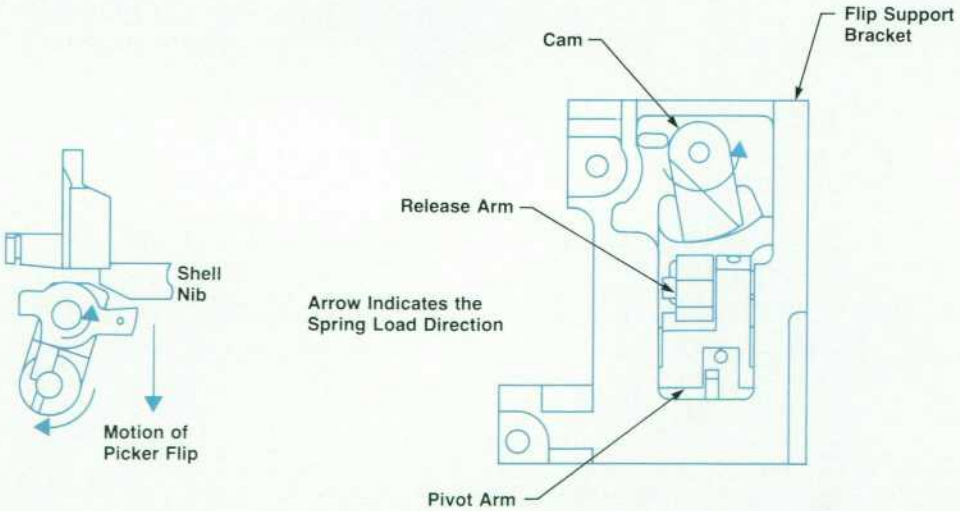


Fig. 9. *Flip latch mechanism.*

and the cam on the top. This keeps the picker flat with respect to the drives and magazines. Fig. 10 shows the steps involved in opening and closing the lock during a flip. To flip, the thumb plunges backwards, pushing the release arm, which is mounted on the pivot arm. Although the release arm can rotate downward on the pivot arm, the thumb, pushing backward on the release arm, forces the pivot arm to rotate backward also, allowing the nib to fall down through. As the belt continues to rotate the hub, the picker's leadscrew bottoms out when the thumb can move back no farther. Thus the entire picker rotates with the hub. As soon as the nib has cleared the pivot arm, the arm springs back to its normal position. At the end of the flip, the nib from the other side comes around, opens the cam, and is stopped by the pivot arm. The thumb, which originally pushed the release arm backward, now pushes it down. With the release arm down, the picker cannot flip again until the lock is rearmed, that is, until the thumb plunges out far enough to allow the release arm to spring back up again.

Stress loads, space, tolerance build-up, and fail-safe once-only actuation were the major concerns in the design of the flip mechanism. Long fiberglass material in both the cam and the pivot arm gives the parts superior wear and impact strength. The orientation of the latch parts not only economizes space but puts the impact loading down onto the pivot shaft for minimal bending stresses. The cam allows for variations in parts and any wear in use while maintaining the picker fixed during plunge operations. The back side of the release arm is designed to help prevent accidental double flips. When the release arm is rotated just slightly, the back no longer aligns with a through hole in the lock's support bracket. This prevents the pivot arm from rotating and eliminates the possibility that the impact at the end of the flip might cause the flip latch to open again.

Mailslot

The mailslot is a mechanism that allows the user to install a cartridge in the autochanger just as if it were being put into a drive. For the picker to grab the disk cartridge, the mailslot mechanism must rotate the cartridge 180 degrees.

The part of the mailslot that accepts and delivers cartridges is called the carrier. In the out position, the carrier extends the cartridge approximately 20 mm out from the front panel for ease of removal by the user. When a cartridge is installed and pushed flush with the front panel, a spring mechanism catches, giving the user a stop position. As this position is reached, a sensor trips, indicating that the mailslot has a cartridge installed. The picker then moves to the correct height to activate the mechanism. Using the picker to activate the mailslot eliminates the need for another motor in the system.

The mailslot rotates the cartridge using forces applied through the picker and an actuator (see Fig. 11). The actuator is approximately at the center of mass of the cartridge and the carrier, thereby keeping the forces in a straight line. The leadscrew nut on the picker drives the actuator. The actuator and the carrier run in tracks molded into the top and bottom pieces of the mailslot. The tracks are designed so that as the actuator drives the the carrier from front to back in the mailslot, the carrier is rotated 180 degrees.

The cartridge load sequence consists of the following moves:

1. User pushes load button
2. Move picker to mailslot actuate height
3. Plunge to maximum get position
4. Check for sensor sensing cartridge in mailslot
5. Rotate mail in
6. Move to get-mail height
7. Plunge to get cartridge
8. Pull cartridge into picker
9. (Sequence of moves to put cartridge into drive or magazine)
10. Move to mailslot actuate height
11. Rotate mail out.

When the carrier is facing the inside of the autochanger, it looks like another magazine slot to the picker. When the carrier is facing the outside it looks like a drive to the user. A special catch mechanism between the actuator and the mailslot top keeps the carrier from moving when either the user or the picker is pushing or pulling on the cartridge, but allows the carrier to be rotated and moved when the picker is pushing or pulling on the actuator. In other words, the carrier has limit positions on both sides of the mailslot, and the actuator must be the moving device to move the carrier past these limit positions.

Because the carrier slides in the top and bottom pieces of the mailslot, the wear of the carrier against these parts required special materials considerations. The carrier and actuator needed to be different from the top and bottom, but all parts were to be molded. For regulatory reasons, all

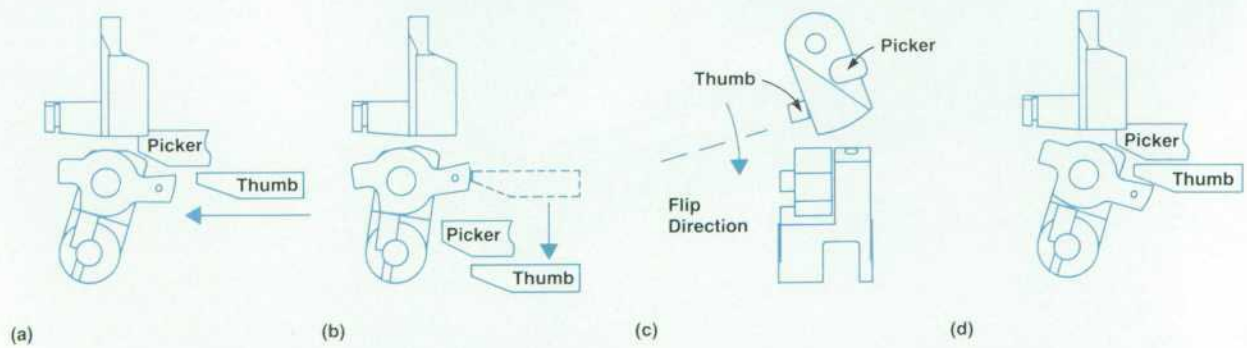


Fig. 10. *Flip latch operation. (a) Initiation of flip. Thumb pushes release arm. (b) Start of flip. Picker falls through. (c) Start of cam opening. (d) End position of flip. Release arm cocked.*

of the materials had to be self-extinguishing when exposed to flame. The top and bottom are made of polycarbonate with glass and Teflon fillers. Polycarbonate is used because of its low cost, since these are the two largest parts. The Teflon (PTFE) is put in for friction reduction. The glass is added because the parts need to be flat and strong enough to hold the whole assembly while the user or picker pushes on the mechanism from either side. A special milled glass fiber was chosen as a filler because it tends to produce flatter parts (±0.2 mm across the part) and adds the required stiffness. The carrier and actuator are both molded out of polyethersulfone (PES) with Teflon filler. PES has several characteristics that are needed in the part design. It is very dimensionally stable material. The PES and polycarbonate materials wear against each other very well. PES can be color matched to the custom color required by HP. Also, PES can be ultrasonically welded. The carrier is approximately 120 mm deep over a section 11 mm high. Molding this would be very difficult if not impossible, so the carrier is made in two pieces and the two are welded together.

To assemble the mailslot, all parts are either added to the top half of the enclosure or placed directly into the bottom half. The top and bottom are then screwed together.

Magazines

The four magazines in the HP Series 6300 Model 20GB/A rewritable optical disk library system each hold eight magnetooptical disk cartridges (Fig. 12). The magazines must hold the cartridges in position for the picker to remove and replace them without too much force, but they must also hold the cartridges while the machine is physically moved. The magazines are designed so that referencing and alignment are correct when the assembly is inserted into the structure.

The alternative of molding the magazine assemblies in one part was ruled out because of schedule constraints and tooling costs. Minimum wear on the cartridge contact surfaces required a plastic part, and for ease of assembly we use a sheet-metal mating part. The entire magazine assembly uses only three parts: a plastic part used four times, a sheet metal part used twice, and a plastite screw used eight times. Details needed for holding the cartridges are easily created in the plastic part. Sheet metal is an inexpensive and effective way of holding the plastic parts together and referencing them into the structure.

The plastic guide in the magazine assembly is made of polycarbonate with 10% PTFE (Teflon) and 10% aramid (Kevlar) fibers. The Teflon is added for friction reduction. Testing during product development showed that the standard cartridge case made of ABS wore too easily against even the lubricated guide, so the cartridge cases were changed to polycarbonate. Extensive testing has shown making both the cartridge case and the guide of polycarbonate is acceptable because of the lubrication in the guide.

The aramid is an additive for dimensional stability. During molding, the aramid ensures that the part shrinkage is the same in both flow directions. This keeps the plastic guide stable in all of its required referencing tasks. The cartridge is spaced and held vertically by the guide. The catch details for holding the cartridge are molded into this part and close tolerances are needed for a proper snap fit. The plastic molding process allows a design that lowers insertion force but keeps the removal force at desired levels. Fig. 12 shows the cantilever spring and cartridge snap details.

The autochanger is designed for a million exchanges.

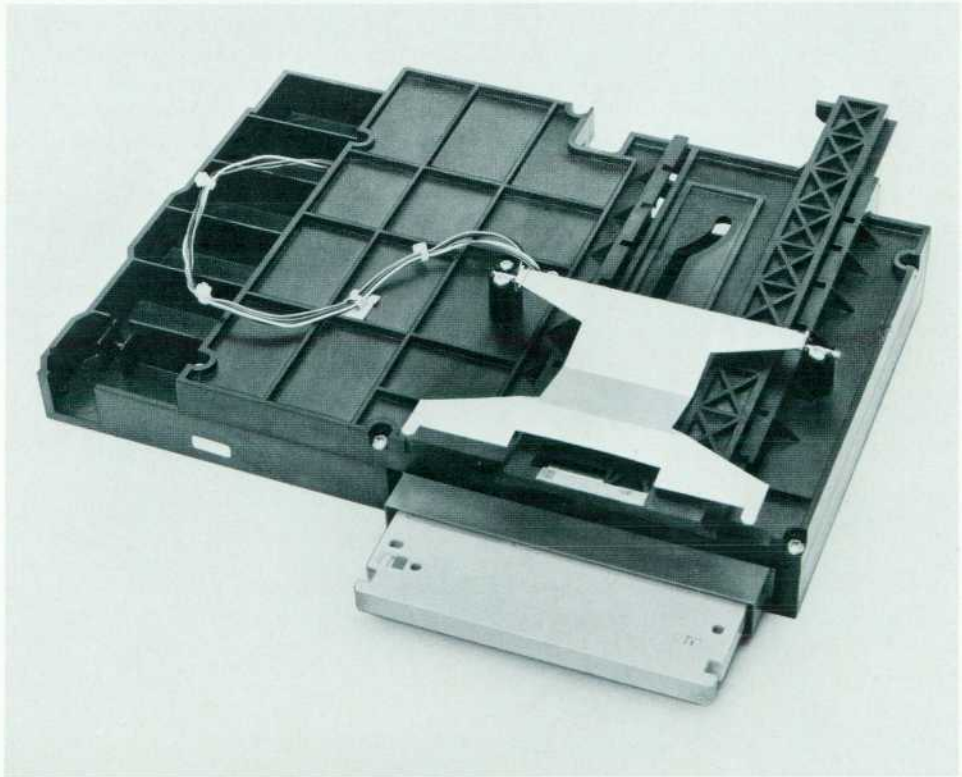


Fig. 11. *Mailslot assembly.*

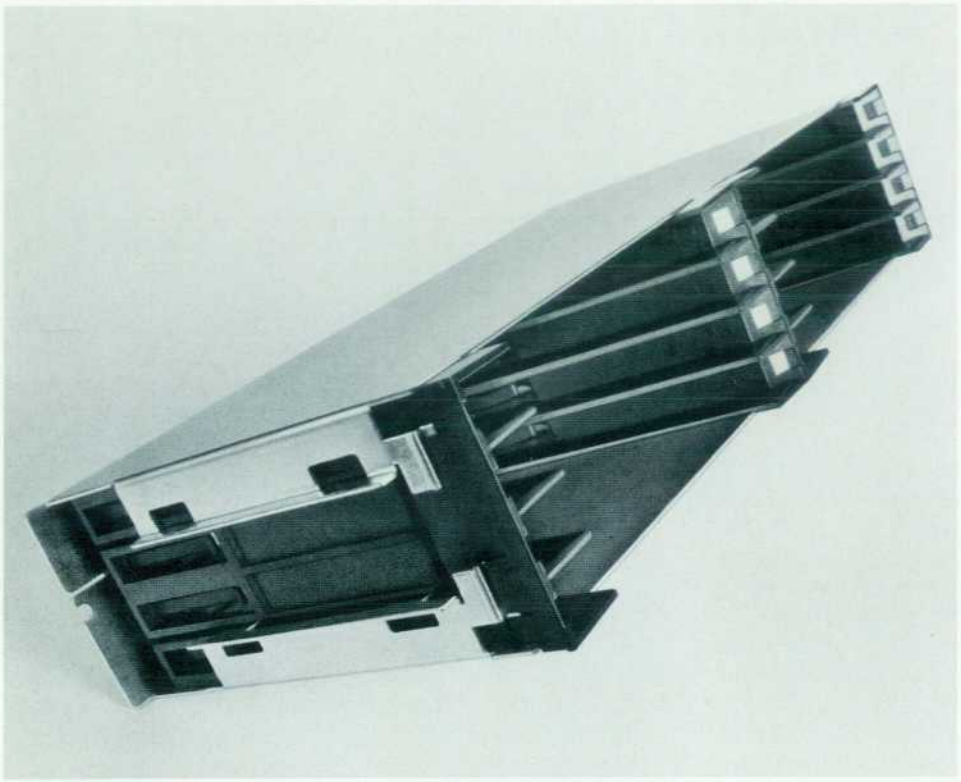


Fig. 12. *One of the four eight-cartridge magazines. Cantilever springs and snap details to hold the cartridges are molded into the plastic side parts of the magazines.*

Therefore, the springs designed into the guides must have a long fatigue life. The Kevlar is helpful in this area, but not as much as fiberglass would be. A trade-off of dimensional stability for part strength and therefore fatigue life was made in the choice of materials. This trade-off has required testing to ensure that the spring life is at least 150,000 cycles per spring. Considerable testing has established that the part meets the requirement. The cantilever springs are designed for constant stress over their entire length.

The sheet-metal part is a horizontally symmetric part that holds the plastic parts together and allows easy installation of the assembly into the structure. This part is folded such that the side plastic parts are accurately located with respect to the reference surfaces on the sides of the structure. This reference scheme keeps the tolerances between the magazines and other critical elements to a minimum. Once the magazine assembly is built, it can only be installed correctly into the machine. Hard-tooling of the sheet-metal part reduced magazine location errors and improved performance of the unit during of the course of testing and design.

Acknowledgments

Special thanks to Dave Jones and Doug Fleece, whose efforts helped this project meet its goals. Dave, who worked on the project from the beginning, designed the sheet-metal enclosure and did the product design. He was also a great help with a lot of good ideas in many areas. Doug, who came on the project later and finished up in some needed areas, designed the front bezel and mailslot door and finished up several smaller parts.

Optical Disk Autochanger Servomechanism Design

A "sense of touch" and error recovery routines contribute to reliability. Data capture, error injection, and mechanical regression testing facilities improved the productivity of the designers.

by Thomas C. Oliver and Mark J. Bianchi

THE SERVOMECHANISM OF THE HP Series 6300 Model 20GB/A rewritable optical disk library system is a collection of electronics and firmware algorithms that control the autochanger mechanism. The servo provides the muscles and brains that bring the mechanical limbs to life. Muscles are provided using motors, power supplies, and sensors. The brains are contained in the firmware program that controls how the muscles are energized.

It is the responsibility of the servo to control the autochanger mechanism reliably. Designing for reliable control requires that many aspects of the system be analyzed and optimized. The design encompasses a broad range of engineering disciplines, including system models for stability and performance, continuous and discrete-time control theory, firmware architecture, motor parameter optimization, analog hardware design, and digital logic design.

Goals and Solutions

The goals for the servomechanism design were high reliability, flexibility, system stability, high performance, user safety, and contributions to other parts of the development effort.

Techniques for achieving high reliability include the sense of touch for adaptive, gentle movements, minimizing the number of components through firmware integration, the use of proven, reliable technologies (HP ICs, standard LSI, surface mount technology), increasing hardware design margins by overrating, and the use of self-calibration, error detection, and recovery techniques.

Flexibility is provided by firmware implementation of servo functions, a modular design architecture, and the use of a high-level programming language.

System stability is ensured by extensive system modeling before and during implementation, optimized programmable compensation for each movement, and margin verification over all operating ranges.

High performance is achieved by optimal compensator selection for each movement, motor and power supply optimization based on the performance model, and overlapping of movements.

User safety is ensured by continuously monitoring applied forces in firmware and hardware.

Contributions to the development project outside the

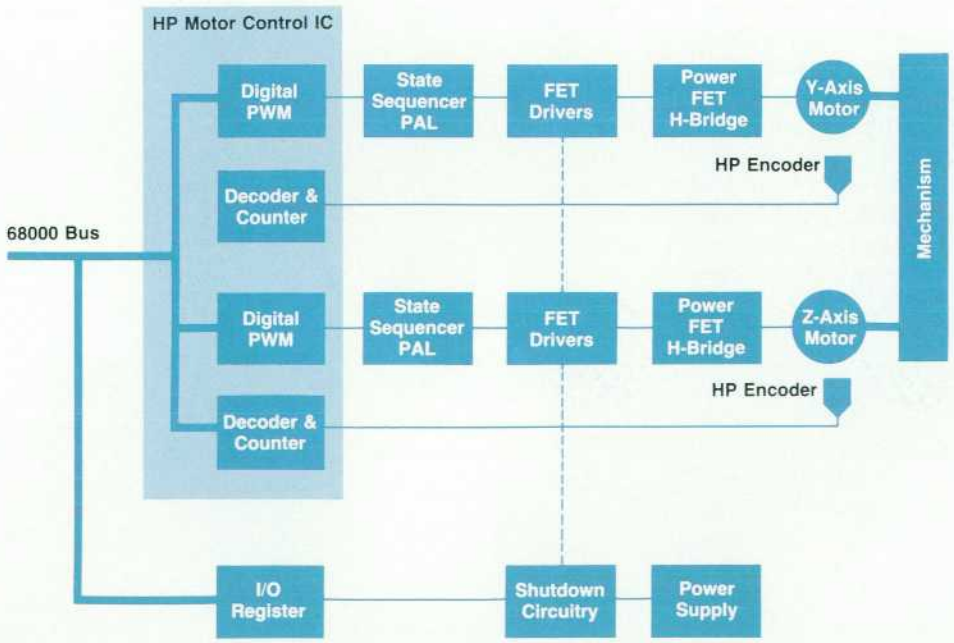


Fig. 1. *Servo hardware architecture of the HP Series 6300 Model 20GB/A rewritable optical disk library system.*

servo area include a data capture system, mechanical regression tests, error injection, and code that can be leveraged by other projects.

Design Philosophy

Extensive modeling was performed before the design was implemented. A performance model was developed so that motors, gears, and power supplies could be selected to meet the swap time goal. Plant models and compensator techniques* were simulated so that optimal control schemes could be investigated. Bandwidth analysis was required to ensure that a microprocessor could close the control loop fast enough. Root locus and Bode analysis methods were vital tools used to ensure adequate stability margins.

Proven, reliable HP technologies are employed in the servo design. Digital implementations were selected over analog techniques because they offer greater flexibility and fewer components. A custom digital IC from the HP DraftPro plotter family is used because it has a proven track record and is used in large volumes. HP optical encoders were a natural choice based on their exceptional reliability and manufacturability. A power driver from the HP 7980A tape drive is also used. Design margins were increased on all critical components, particularly the devices that dissipate large amounts of power. Additional reliability is achieved by sharing a single microprocessor between the servo and interface functions.

The servo firmware architecture contributes to HP Series 6300 Model 20GB/A reliability on many levels. The firmware is designed to provide maximum integration of servo functions such as closed-loop control, profile generation, and error detection. Firmware integration helps reduce parts count and increase flexibility. A "sense of touch" technique was developed to eliminate the need for sensors on the moving transport, a key contribution to reliability. Control of each mechanical function (vertical movement, flip, translate, I/O) is tailored to provide gentle, adaptive movements. Self-calibration, error detection, and error recovery play key roles in increasing reliability and manufacturability.

Another project philosophy was to increase the productivity of the design team through the creation and use of tools. Features such as data capture, error injection, and mechanical regression tests are incorporated into the firmware to give the design engineers a "mechanical oscilloscope" for the mechanism. Our investment in networked HP 9000 workstations provided a common development, debug, and testing platform for the design team.

*A plant model is a control theory model of the device being controlled. Compensator techniques are methods of stabilizing the control system.

Hardware Architecture

The servo hardware is kept to a minimum to ensure reliability. Digital circuitry is used whenever possible because it usually requires fewer parts and is more flexible than analog implementations. Real-time functions, which can't be performed in firmware, reside in hardware. These functions include the motor drivers and the motor position encoding. Fig. 1 shows the servo hardware architecture.

The motor driver consists of a pulse width modulator and an H-bridge amplifier. A custom HP ASIC (application-specific IC) is used to generate a PWM (pulse width modulation) signal for two motor drivers configured for bidirectional operation. The IC contains a register that is used to transform a digital value into a TTL PWM signal having a duty cycle proportional to the register value. A state sequencer PAL (programmable array logic) transforms the IC's outputs into four time-sequenced signals that control operation of the FET H-bridge. The PAL prevents cross conduction in the FET H-bridge and offers a flexible alternative to analog time delay circuits. The FETs amplify the sequencer outputs and present voltage pulses to the motor. The motor averages out the high-frequency pulses and responds as if a dc voltage were applied at its inputs.

As the motor turns, its two-channel shaft encoder sends a series of pulses to the HP ASIC. Quadrature decoding is performed by the ASIC, and a register representing the motor position is made available to the system. Thus, the firmware uses the ASIC as a single interface with which

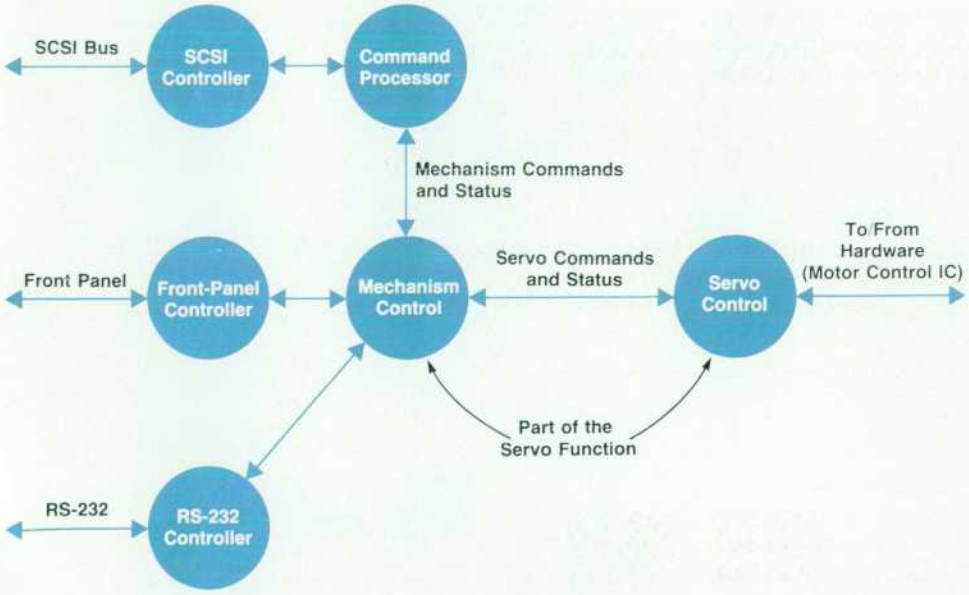


Fig. 2. *Top-level firmware architecture.*

to control the motor voltage and receive positional feedback.

Firmware Architecture

The servo firmware is partitioned into two types of code: real-time and nonreal-time. Real-time firmware is referred to as servo code while nonreal-time firmware is called mechanism code. Servo code is responsible for the closed-loop operation of the system. Real-time code is placed in an interrupt routine, which executes once every millisecond. Mechanism operations are coordinated by the mechanism code through control of the servo code. The two processes communicate through a set of primitive routines, which manage the flow of commands and state information. Fig. 2 shows the overall firmware architecture and Fig. 3 shows the servo firmware architecture.

Servo Code

The servo code is responsible for closed-loop compensation, profile generation, sense of touch calculation, error detection and shutdown, and development tools.

Closed-Loop Compensation. Closed-loop compensation is the algorithm that determines how the motors respond to command position changes or payload disturbances. Compensation is implemented with a simple P-D (proportional-derivative) algorithm. It can be described by the following equation:

$$\text{pwm} = K_1 (e - K_2\omega),$$

where pwm is the command voltage to the PWM on the motor control ASIC, e is the motor's position error, ω is the motor's angular velocity, K_1 is the proportional gain (stiffness), and K_2 is the derivative gain.

Since each mechanical function has a different load characteristic, the servo gains must be adjusted for each mechanical operation. Knowledge of the load is maintained by the mechanism code, which controls these gain values.

Profile Generation. The method used to change the motors' position from point (Y_1,Z_1) to point (Y_2,Z_2) is referred to as profile generation. Profile generation is the creation of a series of command positions to the Y and Z compensators that will cause the motors to move a desired distance. A trapezoidal velocity profile is generated based on parameters of distance, acceleration, and peak velocity. Scaling is also performed to allow the motors to move different distances at different speeds simultaneously.

Sense of Touch Calculation. The servo code is responsible for determining the forces being exerted by each motor/load system. Forces are monitored by the servo and mechanism firmware to obtain an additional form of feedback from the mechanism. During each interrupt, the servo calculates force using an equation based on motor voltage and velocity. The equation is:

$$f = K_3\text{pwm} - K_4\omega,$$

where f is the applied force, pwm is the command voltage to the PWM on the motor control IC, ω is the motor speed, and K_3 and K_4 are constants based on the motors and gearing. Traditional techniques use direct current measure-

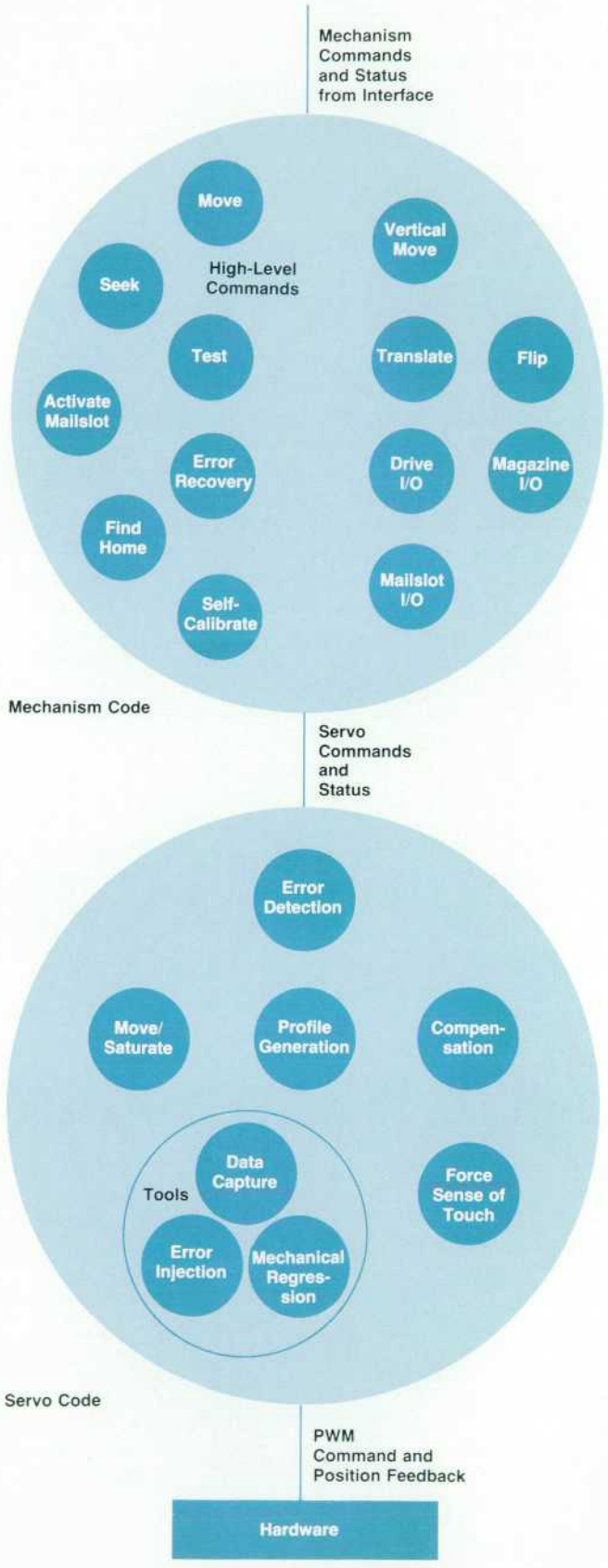


Fig. 3. *Servo firmware architecture.*

ment, which involves added circuitry and cost.

Error Detection and Shutdown. Error detection is an important safety and reliability feature of the servo code. During each interrupt, the servo firmware monitors the forces, voltages, and currents of both motor systems to determine if an error condition exists. Measured values are compared against expected thresholds for a given submove. The mechanism control code tailors the threshold for each submove within a mechanical operation. If limits are exceeded, the servo will immediately disable motor power and record pertinent information for later processing.

Development Tools. The servo code also performs functions that aided in the project as a whole. Features such as data capture, error injection, and peak force detection were designed into the firmware in its early stages. Data capture provided designers with a multichannel "mechanical oscilloscope" to observe key variables used in the servo interrupt (see "Data Capture," page 29). Error injection allowed designers to simulate error conditions within the servo code and observe the system's response (see "Error Injection," page 33). Peak force detection provides servo information that was used to identify movements that were stressing the design margins of the mechanical assemblies. Hooks for the mechanical regression test were used to determine how well a unit was operating over time.

Mechanism Code

The mechanism code provides a translation between the SCSI interface and the servo code. Commands are received and executed by transforming them into a series of smaller submoves. Adaption, error detection, and recovery functionalities also reside in the mechanism code.

The mechanism code accepts high-level commands from the autochanger interface, executes the command, and returns status. High-level commands include the following:

- Move/Exchange. Move cartridge from element A to element B.
- Seek. Position transport at target element.
- Test. Test for the presence of a cartridge at a target element.
- Actuate Mailslot. Rotate the mailslot assembly to perform I/O with the user.

An element is defined to be any possible resting place for a cartridge, including storage magazines, optical drives, the mailslot, and the transport.

The commands are transformed into a series of basic autochanger operations, such as:

- Vertical Move. Position transport to a vertical position.
- Translate. Position transport to access a given vertical stack of cartridges.
- Flip. Rotate the transport.
- Cartridge I/O. Control the plunger to move cartridges between the transport and the magazines, drives, or mailslot.
- Rotate Mailslot. Control the plunger to rotate the mailslot assembly to or from the user.

For example, "Move element 11 to element 2 with flip" would be transformed into the following sequence of autochanger functions:

1) Determine that element 11 is a storage slot and element 2 is a drive.
2) Determine if a translate is required to position the transport to the appropriate stack for the storage slot. If so, perform the translate.
3) Perform vertical move to the storage element.
4) Get cartridge from the storage element.
5) Perform flip.
6) Determine if a translate is required to position the transport to the appropriate stack for the drive. If so, perform the translate.
7) Perform vertical move to the drive element.
8) Put cartridge into the drive element.

Each autochanger function is then transformed into a series of small movements called submoves. There are two types of submoves:

- Move (Y,Z). Move motors a given distance at a specified acceleration and peak speed.
- Saturate (Y,Z). Same as a move except that motion is halted if force exceeds a specified threshold.

Move (Y,Z) is used for high-speed, unobstructed movements of a known distance. Saturate (Y,Z) is for low-speed, adaptive movements of variable distance.

Autochanger functions consist of one or more combinations of move (Y,Z) and saturate (Y,Z) submoves. Each function has a tailored set of these submoves that guarantees that the movements will be gentle. As the submoves are executed, the servo gains are adjusted to allow for changes in load characteristics. An example of the process for a flip is outlined below:

1) Move plunger backwards a fixed distance to engage the flip lock.
2) Change gain for flipping.
3) Move plunger backwards a fixed distance to perform the flip.
4) Ensure that the flip is completed by performing a saturate until the force exceeds a fixed threshold.
5) Change gain for plunger movement.
6) Move plunger forward to relieve the force.

Each submove within a function has a unique set of stability, performance, error recovery, force, and reliability critera. Each submove is assigned a unique identification code (ID) which is used to determine how the move should be performed. Before a submove is executed, its ID is used to fetch acceleration, velocity, and force limits to use. If the move fails, its ID is used to determine the type of error recovery scheme to employ. This tailored technique provides gentle, stable control of the mechanism, thus increasing reliability.

Adaption

Adaptive algorithms were developed to increase reliability and decrease sensitivities to dimensional variations. Dimensions that require adaption are the translate distance, flip distance, magazine depths, mailslot depth, and mailslot actuator throw. If a dimension is susceptible to variation, the firmware is designed to measure the distance using a two-step process. The first step is to undershoot the typical position using a move. The second step is to perform a saturate until a hard stop is encountered. The amount of variation is calculated and the proper dimensional adjustment is made. Subsequent operations will be performed at full speed using the newly calibrated dimen-

sion. This form of self-correction eliminates unnecessary impulse forces caused by tolerance buildup.

Adaption is also used to increase the reliability of the autochanger/drive interface. Initially, drive insertions are performed at slow speeds. An adaptive technique is employed to measure the point at which the drive accepts the cartridge. The results are used so that subsequent insertions are exact and can occur at high speed.

Mechanism Initialization

For the servo firmware to be able to perform controlled movements, the mechanism must be methodically set into a known initial state. This process of initializing the transport system is referred to as finding home. The successful completion of the find home routines is crucial for the proper operation of the autochanger.

The transport mechanism is designed to operate through a number of degrees of freedom using only two servo motors. As a result, motions such as flip, translate, and insert/extract require the servo to move the transport thumb to specific absolute positions within the transport sleeve (see article, page 14). The translate motion, in addition to requiring a specific location along the transport sleeve, requires an absolute reference point at the bottom of the autochanger. These positional requirements necessitate the accurate and repeatable location of two points of origin from which the servo can reference its motions. These are called the plunge origin and the vertical origin. Once these points of origin are found, the servo can reliably perform all of its fundamental movements. However, the autochanger may have power removed at any time during one of its motions. This implies that upon subsequent restoration of power, the mechanism may not be in a state that facilitates locating these points of origin. The find home algorithms must therefore be capable of interpreting the current state of the mechanics through various feedback methods, moving the mechanics in such a way that location of the points of origin is possible, and finally locating the points of origin in a very repeatable and reliable manner.

To interpret the current state of the mechanics upon power-up, the find home process employs a number of algorithms collectively referred to as initial recovery. These algorithms are charged with assessing the state of the mechanics using position feedback and force sense of touch. Once sufficient information has been gathered, these algorithms are also responsible for maneuvering the mechanism to a position from which it is possible to complete the find home process. Each initial recovery algorithm is designed to perform specific motions assuming a certain mechanical configuration and/or range of position. Therefore, each algorithm is most effective when used with a specific move type. To choose the recovery algorithm that best matches the current state of the autochanger, the nonvolatile RAM is examined to determine the ID of the last movement that was occurring when the power was removed. This number is used to select the appropriate initial recovery routine.

Once the initial recovery routine has completed, the find home process can proceed with locating the plunge and vertical origins. This process involves a number of steps that must be performed in a specific order. The transport must complete a full flip so that the mechanical hard stop along the plunge axis can be located. Force sense of touch is employed to locate this point and the servo firmware initializes the plunge axis position to zero. The transport can then be moved downward to locate the vertical hard stop at the bottom of the autochanger. After this is determined using force sense of touch, the vertical position is initialized to zero. The last basic find home motion is performed by sensing which vertical stack the transport is facing. Once the two axis origins have been located and the correct stack has been set, the mechanism is able to perform all of its fundamental motions.

Three more operations are performed that provide the firmware with additional information. The first involves a slow traversal of the two vertical stacks while monitoring the vertical force. This motion ensures that the entire path is free of obstructions. The second involves determining whether there is a cartridge in the transport sleeve. This is performed by moving the plunger slowly outward towards a solid section of the mailslot while monitoring the force exerted. The presence or absence of a large force increase denotes the presence or absence of a cartridge. The third operation involves determining which side of the transport sleeve is facing upward. This piece of information is important since it helps determine the orientation of the cartridge before it is inserted into a magnetooptical drive. These drives are single-sided devices, so proper orientation of the cartridge is vital to the successful completion of a move or exchange command.

The accurate positioning of the front of the transport is critical for reliable insertion and retraction of cartridges into and from the magazine slots and magnetooptical drives. The servo system is capable of very accurate positioning of the transport mechanism. However, vertical motion of the transport is controlled from the horizontal carriage assembly, which is in the rear of the transport. Mechanical tolerances and variations in the manufacturing of the transport assembly may result in the mechanism's not being exactly perpendicular with the vertical axis. In addition to a deviation from perpendicular, the front of the transport may change its vertical position in flipping from one side to another and from changing from one vertical stack to another. To compensate for these three variations, two optical sensors are employed in conjunction with firmware algorithms to calibrate the transport system.

Transport Calibration

The accurate positioning of the front of the transport is critical for reliable insertion and retraction of disks into and from the magazine slots and optical drives. The servo system is capable of very accurate positioning of the transport mechanism. However, vertical motion of the transport is controlled from the horizontal carriage assembly, which is in the rear of the transport. Mechanical tolerances and variations in the manufacturing of the transport assembly result in a mechanism that may not be exactly perpendicular to the vertical axis. In addition to a deviation from perpendicular, the front of the transport can change its vertical position in flipping from one side to another and in changing from one vertical stack to another. To compensate for these three variations, two optical sensors are em-

Data Capture System

Early in the development of the autochanger, it was found that much of the servomechanical testing and evaluation could not be performed by visual observation. The unaided eye was sufficient to diagnose gross mechanical problems at slow speeds. However, the design team needed some way to instrument the autochanger so that servo and mechanical parameters could be accurately correlated and analyzed. Typical methods of measurement would have involved the use of accelerometers and/or high-speed cameras to provide dynamic measurements. These techniques were dismissed since they could not provide many of the measurements needed by the team, and because of the difficulty of synchronizing their output with simultaneous measurements of the servo loops. A "mechanical oscilloscope" was deemed necessary to facilitate the debugging of high-speed, dynamic problems and to assist the designers in the evaluation of design modifications. Thus, the data capture system was born.

The data capture system is a combination of firmware-resident procedures and workstation-based tools. It provides the designers with the means to examine the variation of any important firmware variable with respect to time. The capture system employs the HP 64000 emulation system in conjunction with "homegrown" data processing and plotting tools. It is designed to be used in a windowed environment, since its output is displayed as an X-Y graph of the variables of interest. Autoplot, a very flexible, general-purpose plotting program, displays the output data in a graphics window for quick viewing. The beauty of the system is that no special hardware is needed, assuming that one has a workstation and an emulation system. In our case, each firmware designer had both of these, so each designer also had a complete data capture system. This greatly improved the team's productivity and ability to debug complex, dynamic servomechanical problems.

Since the servo system contains accurate position information and is operated on a repeatable time base (one-millisecond interrupt cycle), it is an excellent choice for a mechanical measurement system. Positions, velocities, accelerations, and forces are accurately measured in the servo loops and are maintained in digital format. The data capture firmware exploits this by simply copying the values of these variables into a buffer during the servo interrupt cycle. In this way, a log of sampled data is created that can be processed into graphical format.

The data capture system allows a number of variables (up to 10) to be traced simultaneously during a user-specified length of time. The sampling period can be set from the highest resolution of one sample per millisecond down to the lowest of one sample per 255 milliseconds. In addition, the trigger event that begins the data trace can be set to one of four different conditions: (1) the first occurrence of a specified move ID, (2) the first occurrence of a specified error code, (3) the occurrence of any error condition, or (4) a special event that can be inserted into the firmware specifically for debugging. The trigger location within the capture time is also user-definable and can be set at the beginning, the center, or the end of the capture period. This feature makes it possible to view data that immediately follows the trigger event, data that surrounds the trigger event, or data that precedes the trigger event.

Data Capture Operation

Fig. 1 shows how the data capture system operates. The capture parameters for a specific trace (variables of interest, sample period, capture time, and trigger condition and position) are entered into a text file. A shell script is invoked that converts this text file into an HP 64000 command script. After executing this command script in the emulation window, the data capture system is armed and awaits the trigger condition. At this point, the data capture routines begin copying the values of the specified variables into the data capture buffer, which is located in an unused area of emulator RAM. These routines copy the variables into sequential memory locations in the buffer every sample period. The size of the capture buffer is determined by the number of variables, the size of each variable (byte, word, or long word), the time between samples, and the capture duration. The capture firmware uses pointer arithmetic to keep track of the beginning and end of the buffer, and to implement a circular buffer. Once the capture system is enabled, the autochanger is instructed to perform the motions that will cause the trigger event to occur. When the trigger condition occurs, the remainder of the capture buffer is filled with data.

At this point, another command script is executed in the emulation window. This script copies the contents of the buffer memory to a file. This data file is then processed by a program called mkplt, which generates autoplot commands as its output. The information displayed in the output plot is determined by parameters located within the data capture configuration file. These parameters define which captured variables should be plotted on each axis, what scaling factor should be applied to each variable, and what titles should be placed on each axis. Mkplt also allows simple math functions to be performed on the variables (e.g., plot "var1 – var2" or plot "– var3"). Any of the captured variables can be used as the independent variable instead of time. For example, it is possible to plot the measured vertical force along the Y axis and the measured vertical position along the X axis. The mechanical regression utilities make use of this feature to generate a plot of friction versus position along a given mechanical axis.

A new data trace can be created by reexecuting the HP 64000 command script, exercising the autochanger, saving the buffer

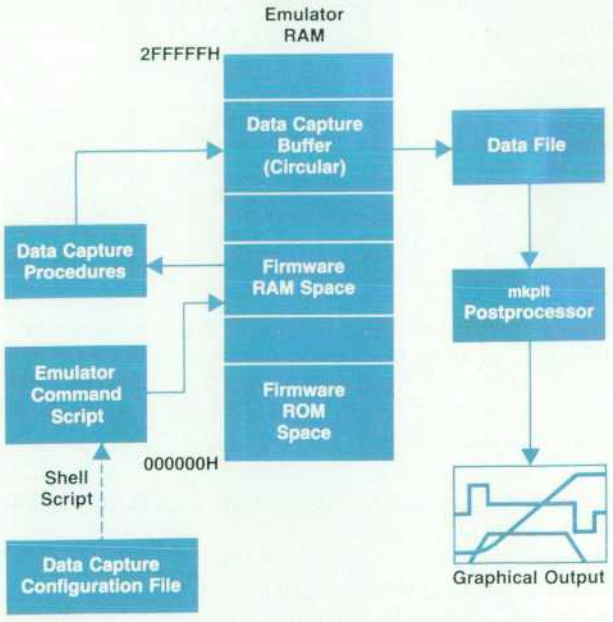


Fig. 1. *Operation of the data capture system.*

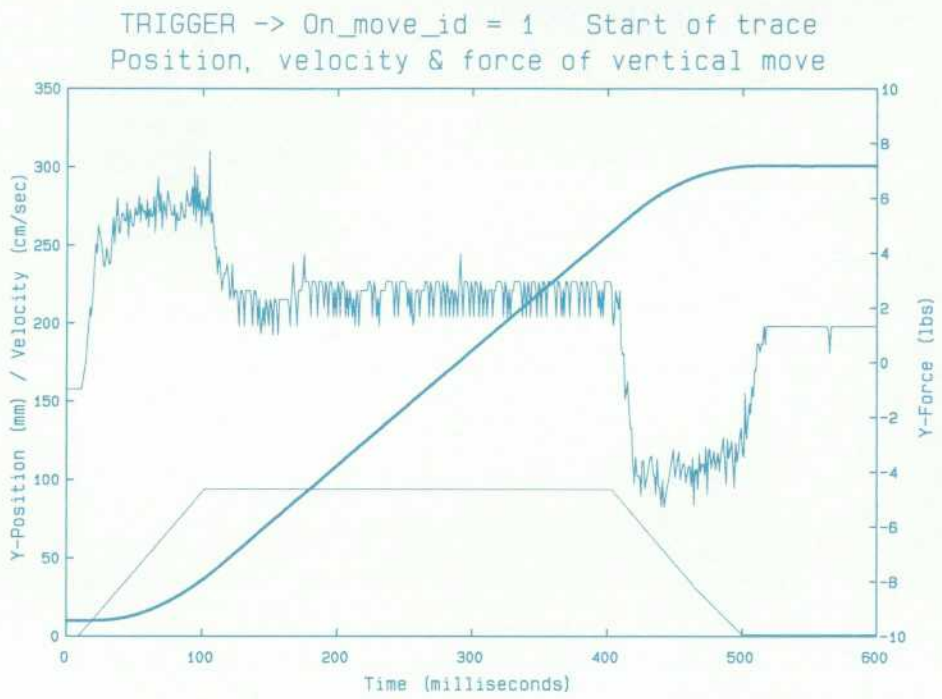


Fig. 2. *Typical data capture system plot.*

data, and then plotting the resulting data file. A trace of different variables triggered by a different event can be captured easily by simply editing a new configuration file and repeating the aforementioned process.

Fig. 2 is a plot generated by the data capture system. The data for this plot was captured during a vertical motion of the autochanger's transport mechanism. It shows the position, velocity, and force applied by the vertical motor.

Second System

A second data capture system was developed that further exploits the hardware on the autochanger controller board. This system collects a few vital bytes of data every 10 milliseconds and transfers this data to a workstation for collection and postprocessing. The controller board contains an RS-232 port, which can be connected to a terminal for debugging purposes. The design team determined that this interface was capable of transferring 18 bytes of data every 10 milliseconds if the baud rate were set at 19,200 baud. Using this information, the designers decided upon a select number of important variables that would be most useful in deciphering and debugging error conditions. Firmware was written that gathers these bytes of data and transfers them to the RS-232 port upon demand.

Data collection is accomplished by first establishing a physical connection between the autochanger under test and an HP-UX workstation via an RS-232 port. The cu program (cu is a standard HP-UX communication program) is invoked on the workstation and its output is redirected into a file. When the appropriate command is typed, the autochanger firmware begins transferring the 18-byte packets of data to the workstation. This data collection may last from seconds to hours, depending on the objective of the autochanger testing. The output file can be processed concurrently with the data collection or examined after the data transfer is completed.

Two programs were developed to process the data produced by this system. The first, called pll, is used to filter out any incomplete packets of data. The first byte of data in each packet is a counter, which is incremented after each packet is transferred. This is used by pll to screen out any data dropouts. The second program, mcsplt, is a postprocessing program similar to the one used in the emulation-based capture system. However, this program offers many more triggering and display features. With mcsplt, it is possible to scan through vast amounts of data and plot only the specific condition or conditions that are requested. Statistical values, such as minimum, maximum, and mean force measured over minutes or hours, can be plotted quickly. A weekend's worth of data, collected from an operating autochanger, can be easily scanned to locate and examine the events that led to a specific error code. The usefulness of this tool in debugging infrequent error conditions cannot be overestimated.

Acknowledgments

Autoplot was written by Bob Jewett of HP Labs. Jeff Kato wrote the data processing programs pll and mcsplt.

Mark Bianchi
Development Engineer
Greeley Storage Division

ployed in conjunction with firmware algorithms to calibrate the transport system.

The autochanger is designed so that all magazine slots and drives are accurately mounted and referenced to the walls of the autochanger structure. The optical sensors are also accurately referenced to this same structure and each one's trip point is very tightly specified. Hence, the distance from the trip point to any one of the vertical locations is known within a very small mechanical tolerance. The firmware contains a table of these distances stored in ROM. By measuring the height of the sensors with respect to the vertical origin, the firmware is able to position the front of the transport accurately at any storage slot or drive. Measurement of these sensors is performed by moving the trans-

port down toward a sensor (with the plunge leadscrew facing upward) while monitoring the output state of that sensor. The state of the sensor will change when the appropriate flag on the transport sleeve interrupts the optical beam. When this occurs, the vertical encoder position is stored as the height of the sensor. A similar measurement is performed after the transport sleeve has been flipped and the plunge leadscrew is facing downward. The difference in height between the two sides of the sleeve is stored as another calibration offset. Thus, accurate positioning to a given storage slot or drive entails a combination of three distances: (1) the mechanical distance from the optical sensor to the slot or drive of interest, (2) the distance from the vertical origin to the optical sensor located in the same stack as the slot or drive of interest, and (3) an offset resulting from the orientation of the transport sleeve.

Service

It is important that the find home and calibration processes be as adaptive and fault-tolerant as possible, since proper operation of the autochanger depends upon their successful completion. If these processes cannot be successfully completed, then the autochanger is inoperable and must be serviced by a customer engineer. Unnecessary service calls negatively impact the product's reliability and add to its cost of ownership. For these reasons, much effort was spent on the design of the find home algorithms in an effort to make them as robust as possible.

In the event that something has broken within the autochanger, repair time can be significantly reduced if the autochanger can make intelligent inferences regarding the faulty components. The find home and calibration routines perform checks after each motion to determine if an incorrect condition exists. If such a condition exists, the firmware will set an appropriate error code and will suggest a list of up to three field replaceable units that it believes may be faulty. This self-diagnosis allows the customer engineer to verify and repair the faulty units rapidly.

Error Recovery

Error recovery is the process by which an unexpected condition is rectified so that normal operation can continue. In the autochanger mechanism, errors may occur for a number of different reasons. Some of these reasons are:

- The host computer requested that a cartridge be moved from a location that did not contain a cartridge.
- There is a temporary mechanical misalignment because of the dynamic nature of the mechanism.
- A power failure has occurred during a movement.
- There has been a mechanical or electronic failure of the autochanger.

The error recovery firmware is designed to recover from a plethora of different error conditions and provide accurate information to the host computer regarding the status of the mechanism.

The four functions of the error recovery routines are error prevention, error detection, restoration of the mechanism to normal operating conditions after an error is detected, and completion of the command during which the error occurred. Error prevention involves verifying that the requested source location contains a cartridge and that the requested destination location is empty. This is done by examining an array in nonvolatile RAM that contains the status of each element within the autochanger. If the firmware believes that the requested move would result in an error, the command is rejected before any motion is attempted. This method of prevention, although very reliable, is not foolproof. The element status array may be incorrect if a customer engineer manually moves cartridges around within the autochanger without reinitializing the element status array. In this event, the firmware must rely on the second facet of error recovery, namely error detection.

Error detection is the means by which the firmware determines that something out of the ordinary has occurred within the autochanger. The firmware detects errors in two ways. The first involves the servo loop monitors, which run continuously during autochanger motion. These routines monitor the forces that are being exerted by the vertical and plunge axes. If either of the measured forces should exceed levels specified for a given motion, the monitor routines immediately disable the servo system and set an error flag.

The second method of detection involves the use of force sense of touch at key positions during cartridge movement. While a cartridge is being extracted from its storage slot, the firmware moves the transport thumb outward slightly beyond the point at which it should engage the cartridge. If no change in force is encountered during this move, then the storage slot is assumed to be empty and an error has occurred. Upon returning a cartridge to a storage slot, the same outward movement is performed to ensure that a cartridge was in the transport. In addition, after retracting the thumb back into the transport, a small vertical motion is performed while monitoring the force on the vertical axis. A large change in the vertical force signifies a failure by the mechanism to release the cartridge. Similar tests are performed when inserting and extracting cartridges into and from the optical drives.

The design of the error recovery firmware is predicated on the assumption that because of the simple and reliable design of the mechanics, error conditions should occur very infrequently. This means that the execution speed of the error recovery routines can be reduced significantly without negatively impacting the overall performance of the autochanger. As a result, the firmware controlling the normal operation of the autochanger is greatly simplified by partitioning the code so that all error recovery algorithms are consolidated into one functional area and all motion control firmware resides in another. A simplified hierarchical diagram of the motion control firmware and error recovery firmware is shown in Fig. 4.

The motion control firmware is designed to complete its execution regardless of the status of the servo system. The lowest-level procedures that perform motion control test the status of the servo system. If the servos have been disabled, the procedures simply return and no motion is performed. All attempts to modify mechanism state information occur via procedures that verify the status of the servo system. These state variables will not be modified if the servos have been disabled. Therefore, if an error occurs, the motion control firmware completes its execution in a

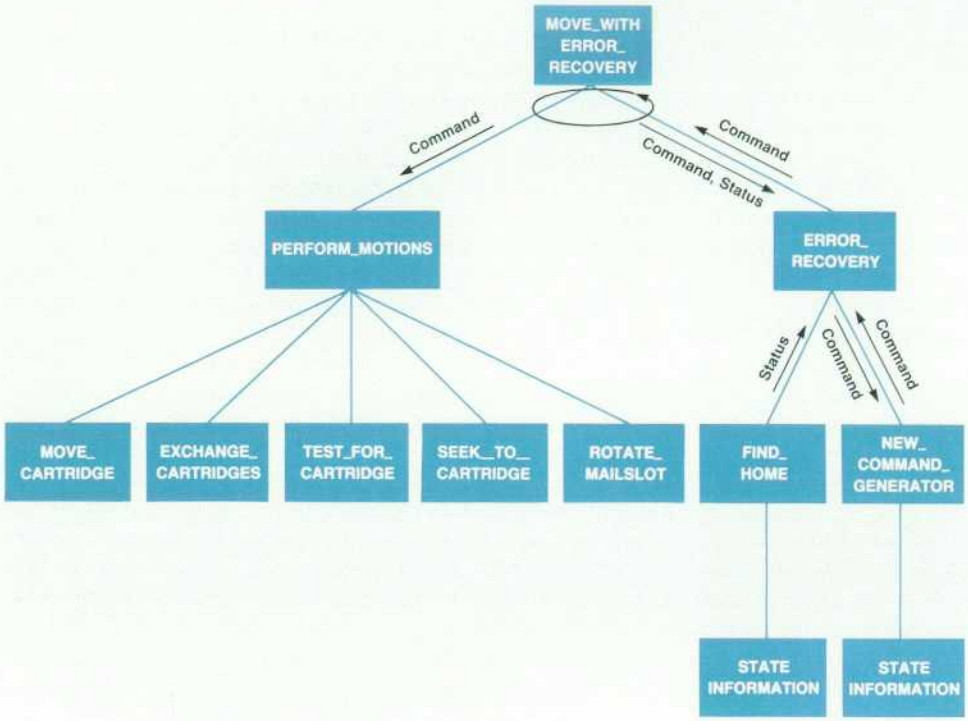


Fig. 4. *Simplified hierarchical diagram of the motion control and error recovery firmware.*

normal manner, but no motion occurs and the state of the mechanism is preserved. This provides the error recovery firmware with a snapshot of the mechanism at the instant that an error occurred. Error recovery can then proceed with restoration of the mechanics.

As can be seen in Fig. 4, ERROR_RECOVERY is composed of the two procedures: FIND_HOME and NEW_COMMAND_GENERATOR. These two procedures can be thought of as physical recovery and logical recovery procedures, respectively. The FIND_HOME procedure is responsible for successfully maneuvering the mechanism out of the error condition and then restoring the mechanics to a known initial state. It makes extensive use of the snapshot of state information that was preserved when the error occurred, and of information gathered via force sense of touch movements. FIND_HOME is the same procedure that is used for power-up initialization. Therefore, once it completes, the mechanism is in a known state and safe motions can be performed. FIND_HOME is invoked by ERROR_RECOVERY whenever a physical error has occurred that forces the servos to be disabled. The logical recovery segment of ERROR_RECOVERY is dependent upon FIND_HOME's successful completion. If FIND_HOME does fail, then ERROR_RECOVERY assumes that something is drastically wrong and calls a routine that attempts to diagnose the failure of the mechanics.

The NEW_COMMAND_GENERATOR procedure is composed of a number of routines that perform three main tasks. The first is to examine the original command and the current state of the machine. The second is to generate a new command that will either gather more information, attempt to complete the original command, or attempt to restore the autochanger to the configuration that existed just before the original command was issued. The third task is to send the newly formed command back to the PERFORM_MOTIONS routine to be executed. This process may be repeated a number of times to complete a command or restore the autochanger successfully. This looping process is initiated by any movement error that occurs while PERFORM_MOTIONS is executing. Once in this loop, ERROR_RECOVERY is directed by a state machine which determines how to resolve the error condition or exit gracefully. Each type of motion command (move cartridge, exchange cartridges, test for cartridge, seek to cartridge, or rotate mailslot) has a state machine sequence that is designed to solve the specific recovery requirements of that particular motion type. However, all state machines are composed of four common states: error recovery initialization, retry the original command, restore the autochanger to its original configuration, and return a pass or fail status. A detailed state diagram for the move cartridge error recovery algorithm is given in Fig. 5.

An error that occurs during a move cartridge command will cause ERROR_RECOVERY to begin execution. The error recovery state machine is set to its initial state, during which the original move command is stored for later use and FIND_HOME is invoked to return the mechanism to a known state. FIND_HOME will then return a pass/fail status and will provide information regarding the presence or absence of a cartridge in the transport. The state machine changes to the Test_Source state, during which a test for cartridge command is generated. This command is passed back to the PERFORM_MOTIONS procedure, which will execute a physical test of the source location for the presence or absence of a cartridge. The result of this test is passed back to ERROR_RECOVERY. If a motion error occurs during this test, ERROR_RECOVERY will invoke FIND_HOME to resolve the error and the same test for cartridge command is repeated. This process can be repeated a number of times, and if motion errors continue to occur, the state machine will decide that error recovery has failed. However, if no

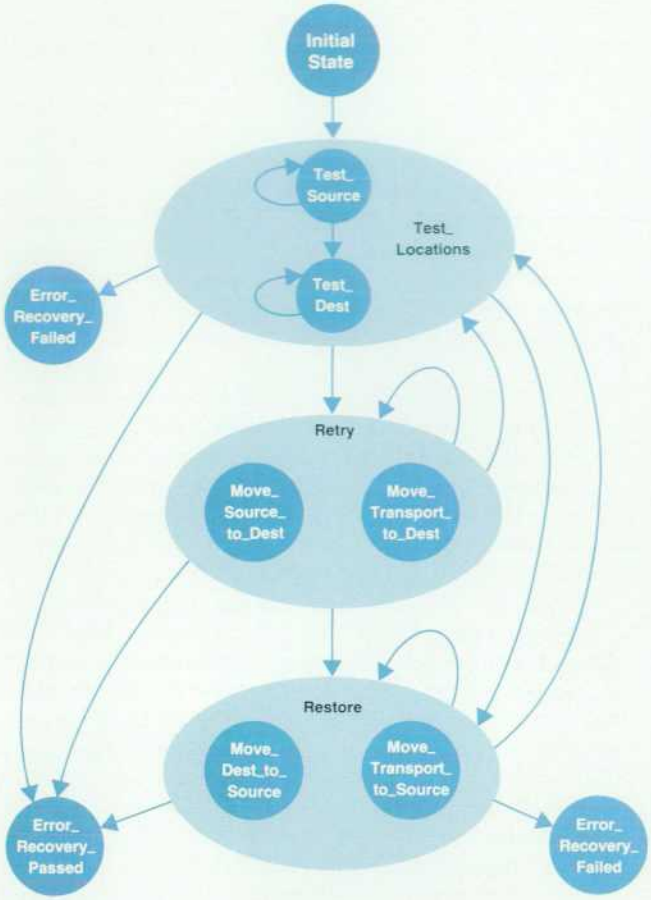


Fig. 5. *State diagram for the move cartridge error recovery state machine.*

motion error occurs, the state machine proceeds to the Test_Dest state, during which another test for cartridge command is generated and passed back to PEFORM_MOTIONS. The destination location is then physically tested and the result is passed back to ERROR_RECOVERY. As with the Test_Source state, any motion error will cause FIND_HOME to be executed again along with a reexecution of the test for cartridge command.

At this point, the state machine knows whether the source, destination, and transport are full or empty. Any full/empty combination that should not logically be possible (such as the source being empty, the transport being full, and the destination also being full) causes error recovery to fail and an appropriate error status is returned to the host. If the source is empty, the transport is empty, and the destination is full, then the command must have completed and the state machine proceeds to the Error_Recovery_Passed state. Otherwise, the state machine proceeds to the Retry state and the appropriate move command is generated. This command is either a move from the original source to the original destination or a move from the transport to the destination.

The command is once again passed back to PERFORM_MOTIONS and a cartridge movement is attempted. If it is successful, the state machine proceeds to the Error_Recovery_Passed state. Otherwise, FIND_HOME is again invoked and

Error Injection

Error recovery is a very complex procedure, as explained in the accompanying article. The number of possible situations from which the autochanger must recover is very large. To induce these errors physically would have required many engineer-hours that the development team didn't have. In addition, many errors are extremely difficult to induce. Since the error recovery testing had to be repeated every time the error recovery firmware was changed, it was deemed necessary to have error injection built into the product for the purpose of testing error recovery.

The error injection facility is enabled via the SCSI so that tests can run automatically. It can inject errors for any move at any position. Multiple errors can be queued. The facility injects errors at the lowest possible level for maximum firmware testing. It can also simulate power failure.

The built-in error injection firmware can be divided into two major functions: setting up the error trigger and injecting the error.

Setting up the Error Trigger. The objective of error injection is to be able to inject all possible errors for any move at any position on the vertical or plunge axis. All motion is broken down into many submoves. Each submove is assigned a unique move ID. When the SCSI command is sent to enable error injection, all the pertinent information needed is sent with that command to set up the trigger conditions. The injection information includes the move ID to trigger on, the axis to trigger on, the position to trigger on, and the type of error to inject.

Injecting the Error. Once the error injection command is sent over the SCSI, the built-in error injection firmware is then armed and waits for the trigger conditions to be met. When the trigger conditions are met, the error will be injected. Three types of errors can be injected:

- Servo monitor error. The servo monitor status is intercepted and an injected status is substituted. The injected status may be overforce, overcurrent, or overvoltage. This simulates unexpected physical errors.
- Force sense of touch error. The status returned from the force sense of touch is intercepted and an injected status is substituted. For example, when the autochanger is expecting to feel a cartridge, an error can be injected to tell the autochanger that it did not feel the cartridge.
- Powerfail error. When the trigger conditions are met, the built-in error injection firmware simply jumps to the power-up vectors. This makes it possible to test powerfail operation automatically for all situations.

Automatic Testing

Having error injection capability was only the beginning. The next step was to develop test suites that could could be run from an SCSI host to test the autochanger automatically. Developing these tests was the major part of the overall error injection testing development. The test suite not only had to send the appropriate injection commands to enable error injection, but also had to be smart enough to issue the correct SCSI move command to trigger the injected error, and to check for the proper SCSI status. The following test suites were developed:

- Inject errors for all possible move IDs
- Inject errors at predetermined risky positions
- Inject errors at random positions
- All of the above with powerfail errors injected.

Rick Kato
Development Engineer
Greeley Storage Division

the state machine returns to the Test_Locations sequence to determine which locations are full or empty. Once the Test_Source and Test_Dest functions have completed and returned the status of both locations, the state machine cycles back to the Retry state and either a Move_Source_to_Dest or a Move_Transport_to_Dest operation is initiated, depending on the presence or absence of a cartridge in the transport. This retry sequence may also be repeated a number of times if motion errors continue to occur, and then the state machine will proceed to the Restore state.

A very similar chain of events occurs in the Restore state, except that in this state the exit criterion is the successful restoration of the autochanger to the configuration that existed before the original command was attempted. The move commands that are generated will attempt to place the cartridge back into the source location. As with the Retry state, any movement errors that occur during an attempted restore will cause the state machine to cycle between the Test_Locations sequence and the Restore state. This process may be repeated a number of times, after which the state machine arrives at the Error_Recovery_Failed state. Once the state machine is in either the Error_Recovery_Passed or the Error_Recovery_Failed state, the appropriate status information is returned by MOVE_WITH_ERROR_RECOVERY to the host and error recovery is complete.

Firmware Development Environment

A number of factors contributed to the successful development of the autochanger firmware. One significant factor was the use of individual workstations connected by a local area network (LAN). This networking provided independent access to a common set of source code. The code files for the project resided on one hard disc, and each development workstation used the Network File System (NFS)* to gain access to the files. Each workstation had the same view of the source code, but each firmware designer was able to work independently on an HP 9000 Model 370 workstation. The bottleneck normally produced by editing, compiling, and linking on one machine was removed. In addition to increasing the designers' productivity, the use of a common set of code files facilitated revision control, tool development, and system administration.

Another factor that greatly contributed to the autochanger program was the development of the data capture tool (see box, page 29). This tool provided a means for capturing and displaying any time-varying global variable within the firmware. Data capture was primarily used to focus on mechanical or servo aspects of the autochanger, such as the position, velocity, or force of one or both axes of motion. It provided the designers with a "mechanical oscilloscope" that produced enlightening views into the operations of the autochanger. Hence, it was extremely useful in examining and diagnosing operational errors.

While data capture provided useful insight regarding errors that occurred during the autochanger development, another tool provided the means for exercising the error recovery code without requiring that specific errors occur. The error injection tool (see box, page 33) allowed the firmware designers to force an error to occur during a specific motion and at a specific position or range of positions. By using this tool, the many different states of the error recovery code could be debugged, tested, and verified. Since an error can occur during any portion of the mechanism's operation, simulating all errors by physically inducing an error would be extremely difficult to do. Error injection solved this problem and provided a powerful and flexible means for ensuring the reliability of the error recovery firmware.

A third tool that contributed to the firmware development was the mechanical regression test suite. This set of procedures provided the means to measure various mechanical aspects of each unit. Friction tests, spring constants, hard stops and datum, performance measurements, and servo parameters could be acquired using these utilities. Measurements could be taken, then tests run and the measurements rerun to assess various factors (degradation over time or temperature, effectiveness of a part or design change, baseline measurement). These routines are now used during the manufacturing process to assess the correctness of the unit's assembly.

Acknowledgments

The authors wish to acknowledge the efforts of the other two firmware designers: Rick Kato and Kraig Proehl. All four designers contributed to the design, implementation, and testing of the autochanger firmware. Special thanks to the management team that helped make the autochanger a reality. The team includes section manager Don Stavely, mechanical design manager Mark Wanger, and electrical hardware and firmware manager Mark Gembarowski. Special thanks to Colette Howe for her assistance in testing, tracking, and debugging the firmware. Additional thanks to Jeff Kato, Joel Larner, Kelly Reasoner, John Schere, and Myron Yoknis for their assistance in refining the autochanger firmware. The authors would also like to thank the R&D and manufacturing teams from HP's San Diego Division who offered their expertise in plotter technology.

Network File System is a product of Sun Microsystems.

Qualification of an Optical Disk Drive for Autochanger Use

Ninety-three design changes were made to the stand-alone drive to qualify it for use in an autochanger

by Kevin S. Saldanha and Colette T. Howe

IN ADDITION TO THE USUAL requirements of data integrity, performance, and reliability for a mass storage device, an optical disc drive that is to be used in an autochanger requires a well-defined communications and mechanical interface that operates efficiently and reliably over hundreds of thousands of load and unload cycles.

In designing the HP Series 6300 Model 20GB/A rewritable optical disk library system, we had control over the autochanger end of this interface, but we had to work closely with the drive vendor to establish the other end. The vendor's original design goals had not included use of the drive in autochanger environments. Endowing the drive with this additional functionality and verifying it proved quite a challenge. Between the release of the stand-alone product (Model 650/A) and the autochanger product (Model 20GB/A) there were 93 changes to the drive.

Communications Interface

We needed a reliable interface to the drive that did not impair performance by tying up the SCSI bus. During loads, the autochanger needs feedback from the drive to know when the drive has accepted a cartridge that has been inserted. The autochanger also needs to be able to initiate ejection of a cartridge from the drive, and during operation and under fault conditions, the drive status needs to be communicated to the autochanger controller.

This interface is implemented with two hardwired signal lines: an eject line and a busy line. The semaphore on the busy line indicates drive status, fault conditions on load, and the acceptance of the cartridge in the drive. The timing of the signals was worked out based on a thorough understanding of the load and unload sequences and retry algorithms in the drive. As a result of this effort, we were able to provide inputs to the drive vendor that made these processes shorter and more reliable.

The spin-up sequence during loads includes a read of prestamped control tracks. Part of this is phase-encoded information (PEP) that is read before tracking is established, and can take up to 1.5 seconds to read. The autochanger requires this information just once, when the cartridge is first introduced into it. To eliminate subsequent PEP reads, a third hardwired signal line is included.

Mechanical Interface

Except for not using the eject button on the drive, the mechanical interface definition is no different from that in a manually operated drive. This is largely because of the flexibility of the autochanger architecture. Once we obtained specifications on acceptable insertion windows, angles, and distances, it was possible to program the autochanger to operate within these limits. The main requirement the autochanger has of the drive is that the cartridge eject to a certain minimum distance and with a specified minimum force.

Design Goals

The metric chosen to gauge the reliability of the drive in load/unload cycling was the mean number of swaps between failures (MSBF). The drive as released for use in stand-alone operation had an MSBF of 5,000, which is adequate for manual use. However, this fell far short of the target of 40,000 set for the release of the drive for the autochanger product. With the numerous changes implemented in the drive and media, we were able to prove an MSBF of 150,000 with a confidence level of 95% at the release of the autochanger product. For these products, the drive was operating in the normal horizontal mode. Work is currently underway on a project that will also allow operation of the autochanger and the drive in it on their sides in vertical mode. We have already achieved an MSBF of 200,000 in both these axes (Fig. 1).

Test Strategy

The ideal test vehicle was, of course, the autochanger itself. We could test the drive, the autochanger, and the interface in conditions similar to actual use. Indeed, this is how the bulk of our testing was conducted, and much valuable information was learned from it. As we began finding and fixing the more obvious problems, it took longer to find the more intermittent and wear dependent ones. It takes about two months to perform 200,000 swaps in an autochanger doing nothing else.

We developed special test fixtures to perform specific tests at much higher rates so we could accelerate the test cycle. One of these, the "scrubber," could insert and withdraw a cartridge in a loader tray and complete 200,000 cycles in a week. We also developed a drive tester that essentially was a stationary autochanger picker, which we used to test drives and cartridges. We also provided one of these to the drive vendor so they could duplicate and better understand problems that were encountered.

We maintained detailed records of failures encountered by the autochanger, drive, and cartridge. It proved useful to keep the swap count of a cartridge on the disk itself. A list of all known problems and their resolution status was also maintained and updated regularly. It served as a useful

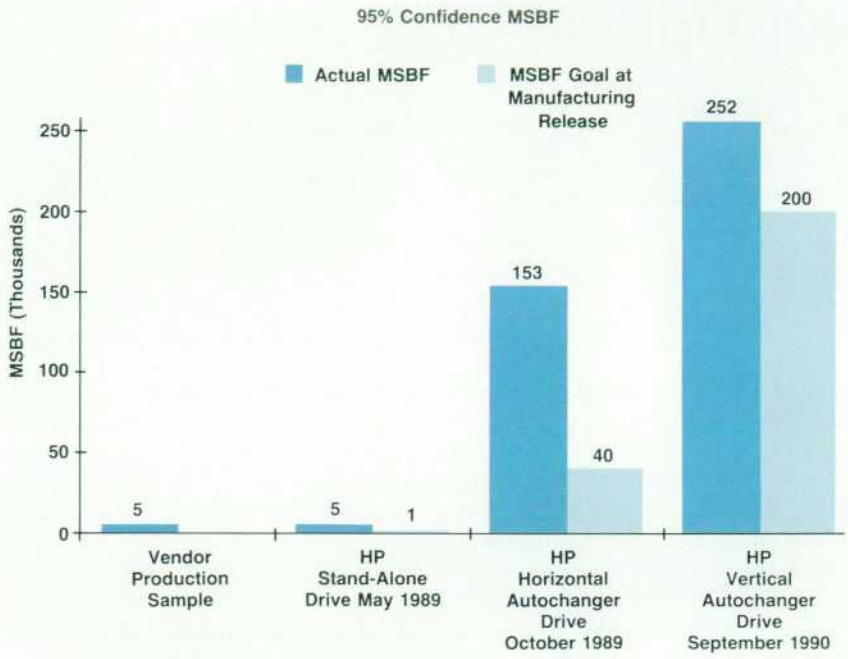


Fig. 1. *Reliability improvement in the optical disk drive. MSBF stands for mean swaps between failures.*

measure of how the qualification effort was progressing (Fig. 2).

Problems and Solutions

We were able to identify and solve problems in several areas as a result of this testing.

The early force-distance profiles used by the autochanger during inserts resulted in excessive loads at first contact with the cartridge slot door and misdetection of the hard stop in the loader tray. This force profile was tuned to deliver a much lower force to the cartridge and to accommodate significant variations in the location of the hard stop (see article, page 14).

We encountered bus hangups stemming from noise on the SCSI and the autochanger-to-drive interface lines. Cables were shortened and rerouted away from noise sources to eliminate these problems.

Some of the misload failures observed were attributable to misdetection of signals at the communications interface between the drive and the autochanger. Changes in timing and debouncing of signals both in the autochanger and in the drive made this detection far more reliable and fixed

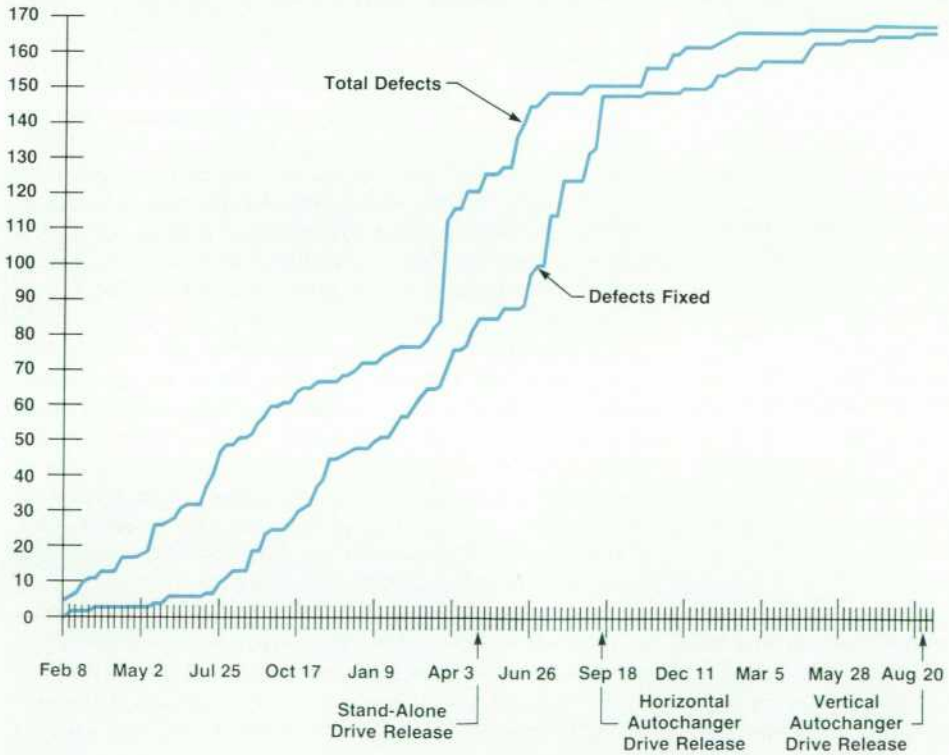


Fig. 2. *A count of known problems was a useful measure of the progress of the drive qualification effort.*

these problems.

Mechanical Problems

Several mechanical problems were encountered and fixed in the course of this development effort. Many of these were related to friction wear between contacting surfaces.

The cartridge, then made of ABS, was subject to heavy wear where it made contact with the autochanger picker, its shutter, the slot door, the loader tray, and the alignment pins in the drives. The ABS dust generated posed a potential threat to data integrity if it landed on the disc or the objective lens. More immediately, these dust deposits caused the cartridge to bind in the loader tray or catch on the alignment pins. Measures taken to minimize this include coating the loader tray with Teflon, blunting sharp lead-in angles on the tray, increasing the radii of the locator pin tips and changing the slot door material to polycarbonate.

Dust from wear was also generated within the cartridge from contact with the disk. This dust was deposited directly on the disk during unloads, when the media spins down onto landing pads on the cartridge. This contamination, which started out as a ring at the inner diameter, could migrate out over the control tracks and the data areas, and could even find its way onto the objective lens. With as few as 8,000 loads into a drive, an ABS cartridge could cause problems such as failure to spin up successfully because of obscured control tracks, improper focusing, and increases in raw byte error rates. These wear problems were overcome by changing the cartridge material to a polycarbonate and by gluing a slip ring onto the disk where it comes into contact with the landing pad on the cartridge.

There was also wear between the drive spindle motor shaft and the the disk hub. Wear on the hub resulted in a sharp edge, which at some point seized the spindle during loads and unloads. These problems were solved by modifying the spindle and hub geometry to make the capture easier and by increasing the spindle hardness. A disk leveling pad was designed to keep the disk from being loaded and unloaded at an angle. The magnetic chuck that holds the disk was modified to provide a stronger capture force-distance profile. Further improvements along these lines were made for operation of the drive on its side. The strength of a retention spring was increased to compensate for the loss of the assistance of gravity in the secure seating of the cartridge against the media sensors.

In addition to mechanical wear problems, we encountered a challenging electrostatic discharge problem. The cartridges were building up a charge as they were being inserted in drives and magazines and moved around by the picker. The cartridges would discharge to the drive loader tray. While most of the charge dissipated through chassis GND, it sometimes managed to cause glitches in the drive electronics. Reproducing these problems and testing fixes was greatly helped by the media testers in use here and at the vendor's site. Although alternative conducting materials were investigated for the cartridge, the solution to these ESD problems was found in establishing a better ground path and putting low-pass filters on susceptible drive controller signals.

In the course of improving the load/unload reliability of the drive mechanism, the testing conducted was also beneficial in increasing the margin for the basic read/write functionality of the drive. The drive accesses control tracks and reads them only during spin-up. These control tracks contain preformatted data. It is harder for the drive to maintain tracking and focus on control tracks than on data tracks. Intermittent tracking problems were observed when accessing these tracks during loads in autochangers. The changes that have been implemented to fix this have resulted in overall improvements in tracking performance in the drive.

Open Communication

Effective exchange of information has been a key ingredient in the success of this effort. The qualification of the drive mechanism and controller for use in manually operated stand-alone operation began at HP's Disk Memory Division in Boise, Idaho. After evaluating several vendors, one was chosen to supply us with both drives and media. A second media source was also identified. We began working with them to define and qualify the drive and media. This effort drew on the expertise of the program in Boise.

The project was then transferred to HP's Greeley Storage Division in Colorado, where work had already begun on integrating the drive into a stand-alone product as well as into an autochanger. The flow of information between us and our vendors had by now swelled to a torrent with facsimile messages flying back and forth daily. Despite the fact that the drive vendor was in Japan, we were able to evolve a very effective working relationship. Aside from regular management contacts, we had periodic technical meetings both here and in Japan. These served to establish engineer-to-engineer contacts. It then became easy to direct communications to the right people and obtain quick resolution of issues. We were fortunate enough to be working with a responsive vendor and it was not uncommon to send a facsimile request for information one evening and return the next morning to find a response to it.

Acknowledgments

Terry Loseke provided much valuable guidance in integrating the new optical recording technology into our products. He has also championed our interests at the ANSI and ISO standards committees. Thanks also go to Brett Mortensen and Sue Magenis for their early work on the drive, to Dave Bradley and Steve Johnson for their help with test systems, to the optical group at HP's Disk Memory Division and in particular Dave Campbell for providing much insight and advice with respect to optical recording, to Ed Sponheimer who managed, and Alfred Natrasevschi, Karen Klemm, Bob Proctor, and Norm Carlson for helping to establish the stand-alone drive, to Tom Oliver for his work in debugging the drive autochanger interface and Al Piepho for his efforts on firmware testing, to the autochanger controller group including Kraig Proehl for the design of the three-wire interface and Mark Bianchi for the drive interface code, and to the materials engineering team of Mark Maguire, Steve Castle, and Mark Davis for their valuable support. Special thanks to the design team at our vendor which included Tadashi Otsuki, who managed the project and Takashi Naito, Hiroyuki Shinkai, and Yoshimori Yamasaki, who were lead engineers.

A CD-ROM Drive for HP 3000 and HP 9000 Computer Systems

The HP Series 6100 Model 600/A HP-IB CD-ROM drive provides facilities that allow HP 3000 and HP 9000 computer system users to access data stored on CD-ROM disks, which can store up to 553 Mbytes of audio and digital information.

by Edward W. Sponheimer and John C. Santon

THE CD-ROM (compact disk read-only memory) is audio CD technology used to store computer data. The important features of the CD-ROM are its capacity, permanence, fast production process, and low per-unit production cost. CD-ROMs are ideal for data that is not expected to change soon, or information that needs to be distributed to a large number of users.

The ideal applications for the CD-ROM include:

- Providing software manuals for computer systems
- Providing a large reference document, such as a dictionary or an encyclopedia
- Providing a training package including reading materials, graphics, and tests
- Providing large-scale software distribution.

The HP Series 6100 Model 600/A HP-IB CD-ROM drive can support software distribution, on-line data or documentation retrieval, computer-based training, and other multimedia applications. Special software security features are designed into the Model 600/A to provide data protection and software distribution security. The capacity of CD-ROMs allows a whole operating system and other software subsystems to be stored on one disk. Since customers may not purchase all the subsystem software, the software protection scheme ensures that customers have access only to the subsystems they purchased. The article on page 49 describes the software protection scheme employed by the Model 600/A.

The Model 600/A uses HP Command Set 80 (CS-80) protocol[1] to communicate with the host computer over the HP-IB interface. Command Set 80 defines a set of commands that enable communication between an HP disk and the host computer. For HP 9000 users, the CD-ROM reader can be used to read non-HP CD-ROMs provided they are created in the ISO 9660 High Sierra standard file format. The article on page 54 describes the ISO 9660 support.

The rest of this article presents an overview of CD-ROM technology and the Model 600/A controller board.

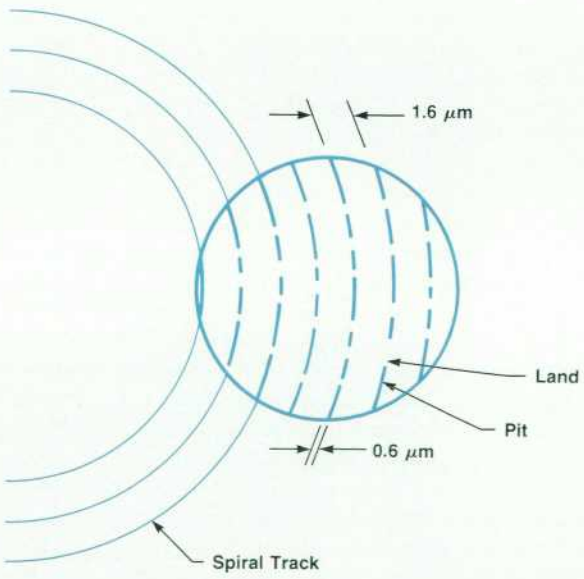


Fig. 1. *A portion of a CD-ROM surface (not to scale).*

CD-ROM Technology

CD-ROM is an optical storage technology. Data is read from the disk using a laser beam. The CD-ROM is a platter 120 mm in diameter that has pits (holes) and lands (material between holes) on one surface, arranged in a continuous spiral (see Fig. 1). The pits and lands represent the information (ones and zeros) stored on the disc. A one is represented by a transition from a pit to a land or from a land to a pit, and the length of the land or pit represents the number of zeros (see Fig. 2). The pits and lands form a track that is 0.6 micrometers wide (track pitch = 1.6 μm) which yields a track density of 15,875 tracks per inch. A combination of a pit and a land that follows it can run from 0.9 to 3.3 μm in length. To put these numbers in perspec-

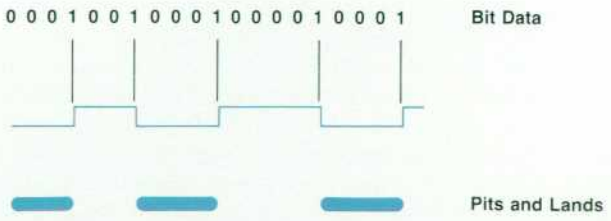


Fig. 2. *Transition between one and zero and zero and one on a CD-ROM. Ones are represented by transitions from pits to lands or from lands to pits.*

tive, the total length of the groove on a CD-ROM disk is approximately three miles and the total number of pits is almost two billion. A CD-ROM disk can store up to 600 megabytes of computer data.

Data is not written on the CD-ROM by the user's computer. Data must be mastered using a specialized publishing process. Subsequent disks can be inexpensively made from the master. There are three steps to making a CD-ROM: data conversion, premastering, and mastering. In the data conversion phase, the data is organized and formatted. Premastering adds the error codes and indexes needed for efficient CD-ROM use. The master, used to stamp the pits in the production discs, is finally created in the mastering phase. A new master must be made every time a new disk is published. However, once the initial cost of publishing the master is done, the cost of reproducing copies is very low.

Recording Format

The format of how data is formatted on a compact disc is specified in two standards called the red book and the yellow book.[2,3] The red book was the first standard and it specifies the format for digital audio data. The yellow book is an extension to the red book and it specifies the format for other forms of digital data, particularly computer data.

Data is recorded on the disk using a code called EFM (eight-to-fourteen modulation). The recording circuits use a lookup table to convert each eight bits of data to 14 channel bits. Three merge bits are inserted after each 14-bit symbol to ensure that the symbol ends do not violate the rules of the EFM code and to equalize the total lengths of pits and lands on the disk (see Fig. 3). The smallest size of an information unit on the CD-ROM is called a frame. The contents of a frame are shown in Fig. 4. The frame is the synchronization element for CD audio and the object of error correction encoding for computer data.

The synchronization pattern of channel bits identifies the beginning of each frame and does not participate in the EFM coding. The control subcode provides some information about the data in a frame such as the presence of CD audio or data symbols. The data symbols represent the information content of the frame. These symbols can represent audio, user data, or a third level of error correction coding. The last eight bytes are parity bytes used for error correction.

The 24 bytes of information represented by 588 channel bits are combined with other frames of 24 bytes of information into sectors of 98 frames. Sectors occur 75 times per second making the transfer rate from the disk 176,400 bytes/s (98 frames/sector × 24 bytes/frame × 75 sectors/second). The organization of the bytes in a disk sector is shown in Fig. 5.

Error Correction Code

The mode byte shown in Fig. 5 describes the nature of the data contained in the data field of the CD-ROM. The yellow book standard defines two modes for computer data. In mode 1, the CD-ROM can store up to 553 Mbytes of information on a 60-minute disk. Mode 1 uses the 288 EDC/ECC (error detection coding and error correction coding) bytes to improve the data error rate. In mode 2 the CD-ROM can store up to 630 Mbytes because all the bytes are used for data (including the 288 EDC/ECC bytes). Mode 2 can be used for applications in which error rates are not critical such as the storage of graphical information.

For the Model 600/A there are three levels of error correction. The first two levels, CIRC1 and CIRC2 (CIRC stands for cross interleaved Reed-Solomon code), are provided by the CD audio standard. The third level of error correction encoding is provided for mode 1 data by a Reed-Solomon product-like code, using the EDC/ECC bytes. The error correction data is placed on the disk when it is mastered. The first two levels of error correction are done on all data fields and are decoded in real time by the CD-ROM drive. The third level is decoded by the CD-ROM controller. See the article on page 42 for more information about error correction on the Model 600/A.

Disk Addressing

On a CD-ROM disk, the groove is divided into a lead-in portion that contains a table of contents, a program area that contains user data, and a lead-out area that contains all zeros. The table of contents resides in CD-ROM area 0:0:0 to 0:1:74 (0 minutes, 1 second, 74th frame). The table

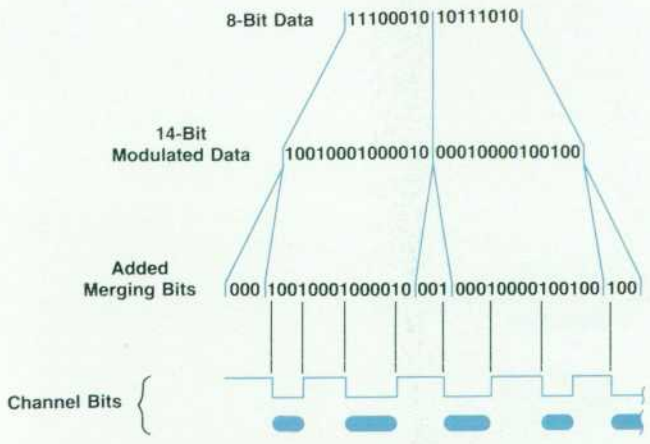


Fig. 3. *Eight-to-fourteen bit encoding example.*

	Number of Channel Bits	Derivation
Synchronization Pattern	27 Bits	(24 Bits + 3 Merge Bits)
Control Subcode	17 Bits	(1 Byte × 17 Bits/Byte*)
Data Symbols	408 Bits	(24 Bytes × 17 Bits/Byte*)
Error Correction	136 Bits	(8 Bytes × 17 Bits/Byte*)
	588 Bits Total	

* 17 Bits/Byte = (14 Modulation Bits + 3 Merge Bits)/Byte

Fig. 4. *Channel bit format in a CD frame.*

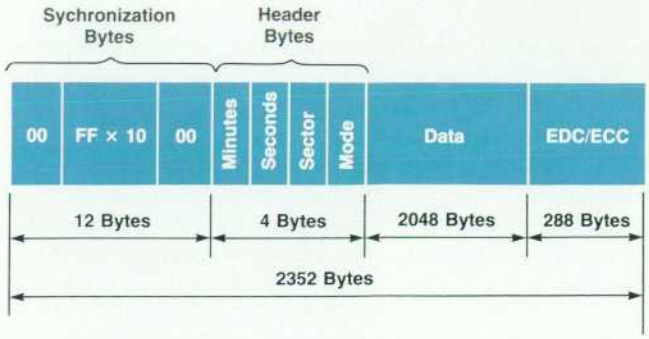


Fig. 5. *A sector on a CD-ROM disk.*

of contents contains the absolute address for each existing track (from 0 to 99 minutes). Each track can contain only one mode of data storage. Audio tracks and digital tracks have buffer areas between them. The start of the user area starts at frame location 0:2:0 (logical address 0:0:0). If present, the system area directory resides in the program area starting at location 0:2:0. The system area data files, which can exist anywhere on the disk, contain information used by the drive or host computer, such as disk security information, logical sector size, CS-80 volume boundaries, and the CD-ROM revision number. The Model 600/A looks for a constant in the system area to determine whether a disc is an HP CD-ROM.

The Controller Board

The Model 600/A controller board provides the interface between the native CD-ROM drive and the host computer. It has three sections: the controller, the disk interface, and the audio section (see Fig. 6). The controller provides the intelligence on the board and is responsible for:

- Communicating with the host computer via the HP-IB interface
- Controlling the CD-ROM drive through the 19.2-kbit/s UART
- Accessing the software protection information in the EEPROM
- Providing the software protection keys to the unscrambler
- Programming the 4-Mbyte/s DMA controller for DMA transfers to the host via the HP-IB.

The unscrambler circuit is responsible for decoding the data on the disk that is scrambled (encoded) for software security. It operates at a transfer rate of 154 kbytes/s and is capable of unscrambling data from the media or from the host. If the CD-ROM in the drive is not an HP disk, the controller switches the unscrambler out of the inbound path of CD-ROM data. The controller uses the keyword information in the EEPROM and the password information from the host to determine the key for the unscrambler.

The disk interface provides the entry point for CD-ROM data. Data arrives from the disk in bursts of 16 bits at 2.1 Mbits/s. These bursts occur once every 11.36 microseconds giving an overall transfer rate of 176.4 kbytes/s. The CD-ROM data is in serial format so it must be deserialized before being sent to the unscrambler or directly to the HP-IB interface.

The Model 600/A can also play audio information recorded on a CD or a CD-ROM. The CD-ROM standard[3] divides all data into groups of 100 tracks and each track can be either audio or digital. Audio information from the drive is serial and arrives at a data rate of 44,100 16-bit samples/s on each channel. The samples alternate between left and right channels. Audio data reconstruction is accomplished with a 4× oversampling digital filter with error interpolation, and a digital-to-analog converter (DAC) that operates at 176,400 stereo 16-bit samples/s. The oversampling filter interpolates between samples and outputs data at the rate of the DAC. The filter is used to deal with aliasing. When a sampled signal is reconstructed, a Fourier transform of the resultant signal shows a baseband signal and a modulated carrier at the sampling frequency. If the audio rate at 44,100 16-bit samples/s signal is reconstructed and the sampled signal is frequency limited to 20 kHz (top end of the audio band), the quiet zone, which is the range between the recreated baseband signal and its alias, is only 4.1 kHz wide. To filter out the alias effectively requires a very steep low-pass filter (most implementations use a seven-pole Chebychev filter). It's not practical to build such a filter without creating some phase distortion in the high-frequency end of the the baseband signal. By digitally recreating samples between each recorded sample (oversam-

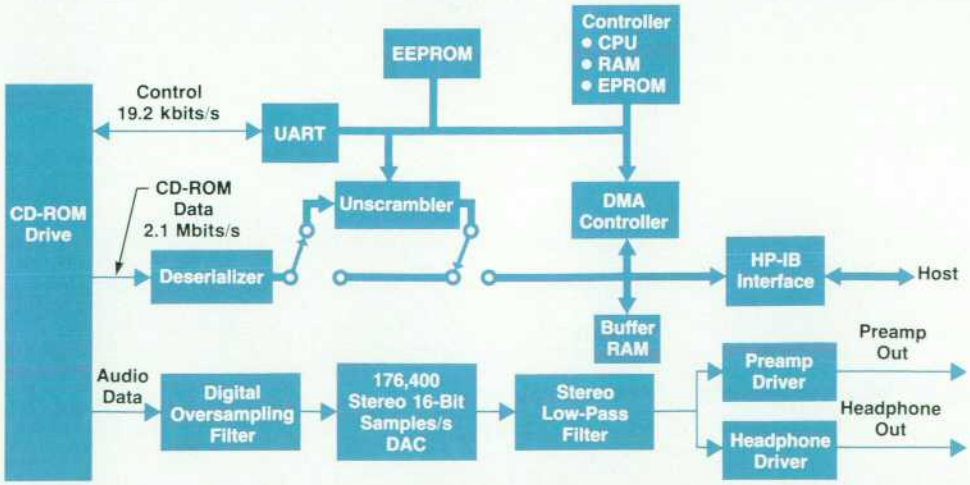


Fig. 6. *The Model 600/A CD-ROM controller.*

pling) the signal can be reconstructed at a higher sampling frequency, thus increasing the width of the quiet zone and decreasing the demand on the low-pass filter. Following the DAC is a low-pass filter that attenuates most of the alias signal. Finally, the signal is routed to the backplane preamplifier and to the potentiometer for the volume control, which drives an all-pass amplifier that drives the headphones.

Conclusion

Compact disk technology started as a technological advancement in noise reduction for the musical recording distribution industry. It has become a valuable tool in the computer industry for the distribution of information, manuals, and software. The HP Series 6100 Model 600/A HP-IB CD-ROM drive, which has complete industry-standard compatibility, full error correction support, software security, and CD audio support, provides the full spectrum of CD-ROM technology to HP's computer systems.

References

1. D. L. Voigt, "A Command Language for Improved Disc Protocol," *Hewlett-Packard Journal*, Vol. 35, no. 1, January 1984, pp. 5-6.

2. *Compact Disk Digital Audio System*, IEC Publication 908, 1987.

3. *Data Interchange on Read-Only 120 mm Optical Data Disks (CD-ROM)*, Standard ECMA-130, July 1988.

Bibliography

S. Lambert and S. Ropiequet, *CD-ROM, The New Papyrus*, Microsoft Press, 1986.

L. Buddine & E. Young, *The Brady Guide to CD-ROM*, Prentice-Hall, 1987.

Error Correction Implementation and Performance in a CD-ROM Drive

The HP Series 6100 Model 600/A implements the error protection algorithm defined by the CD-ROM yellow book standard. This extra level of protection means that the error rate is improved from one error in 10^{12} bits to one in 10^{16}.

by John C. Meyer

IN AN IDEAL DIGITAL transmission channel the information that is input at the transmitter should be faithfully reproduced at the receiver. Unfortunately, because of things like noise, power-line fluctuations, and imperfections in the media, data can be corrupted before it reaches the receiver. This is why error detection coding and error correction coding (EDC/ECC) techniques are incorporated in the digital transmission system. The data on the media is encoded at the transmitting end with the EDC/ECC bits and decoded at the receiving end to correct for errors and to recover the original data. If there are errors, the EDC/ECC algorithms are designed to detect and correct a certain number of errors or to cause retransmission of the data. The box on page 46 is a primer on EDC/ECC techniques.

Because the CD-ROM disk has a very high bit density, it has an inherent error rate of 10^{-5} to 10^{-6} errors per bit. The red book standard,[1] which has become International Electrotechnical Commission (IEC) standard 908, specifies the CD audio media format and provides a parity and error correction scheme that reduces the error rate to 10^{-11} to 10^{-12} errors per bit. All CD manufacturers provide this level of error protection.[2,3] The yellow book standard,[4] which is currently undergoing standardization as European Computer Manufacturers Association standard 130, or ECMA-130, specifies the CD-ROM format. This standard expands on the CD audio standard to include track definitions for digital data, mixed media disks, and digital data representation within a frame. For example, the minimum addressable section of a disk is defined by the red book as 1/75 of a second of audio data or 1176 16-bit audio samples, and by the yellow book it is defined as 2352 bytes of digital data or a disk sector. The CD-ROM yellow book specification provides an additional level of error protection that reduces the error rate to 10^{-15} to 10^{-16} errors per bit. Fig. 1 summarizes the results of error correction provided by the red book and yellow book standards.

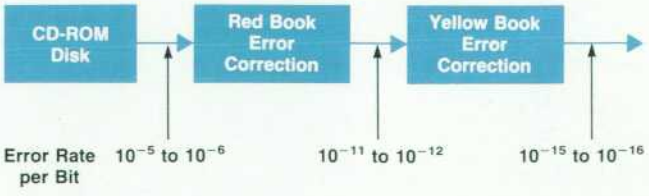


Fig. 1. *Results of error protection schemes provided by the red book and yellow book CD-ROM standards.*

The yellow book further defines three modes for sectors within data tracks. These modes all have a 16-byte header composed of 12 bytes of synchronization data (00, FF,..., FF, 00), three bytes of address data (minute, second, frame), and a mode byte (00, 01, or 02). Mode 0 sector headers are followed by 2336 bytes of zeros. These sectors are used for lead-in at the beginning of disks, lead-out at the end of disks, and transition regions between audio and digital data tracks. The table of contents for a disk is contained in a portion of the lead-in area. Mode 1 sector headers are followed by 2048 bytes of user data and 288 bytes of EDC/ECC data. The EDC/ECC bytes add the extra data protection over that which is available at the native interface as specified by the red book standard. Mode 2 sector headers are followed by 2336 bytes of user data. These sectors can be used for bit maps or other data that may not require the data protection provided by mode 1. Fig. 2 shows these mode formats.

Red Book Error Protection

The CD audio standard (red book) specifies two levels of parity and error correction which are implemented within the drive. The algorithm used for encoding the parity bytes is called a cross interleaved Reed-Solomon code (CIRC). As shown in Fig. 3, there are two CIRC encoders, CIRC1 and CIRC2, which provide the two levels of error protection. Sectors are divided into 98 24-byte F1-frames, which are processed through the CIRC encoder, which consists of three delay sections and two encode sections. The first delay section interleaves the F1-frames into two 12-byte groups. One group is delayed for two F1-frame times (24 byte times). The interleaved F1-frames then pass

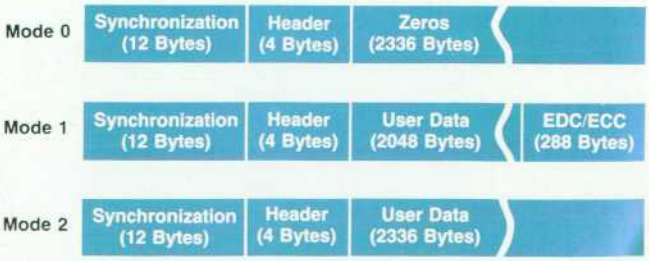


Fig. 2. *CD-ROM sector mode formats.*

through the CIRC2 encoder section, which generates four Q-parity bytes using a (28,24)* Reed-Solomon code. The second delay section delays the CIRC2 bytes from 0N to 27N F1-frame times (N = 4). Time-delayed packets of 28 bytes then pass through the CIRC1 encoder, which generates four P-parity bytes using a (32, 28) Reed-Solomon code. Finally, the third delay section delays every other byte from the CIRC1 encoder one F1-frame time. At the output of the CIRC encoder, the 32-byte packets are called F2-frames.

As a result of all this delaying and interleaving the original F1-frame bytes are delayed from three to 108 F1-frame times, and they wind up being redistributed over 106 F2-frames. Thus, one frame's data is spread over three physical sectors on the media, thereby reducing the impact of burst errors. An additional control byte is added to the F2-frames to form 33-byte F3-frames.

F3-frame bytes are 8-to-14-bit (EFM) encoded and linked together with a 24-bit synchronization header. Three merging bits are used between bytes to maintain run length limits between transitions on the media. The result is that each 192-bit F1-frame is represented by a total of 588 channel bits on the media (see page 39 for an explanation of 8-to-14-bit encoding). Most of this redundancy can be used for error recovery, either for detection of errors and creation of erasure flags or for correction of errors using the P-parity and Q-parity bytes. An erasure flag is error location information that is obtained outside the decoding process, usually from hardware. If erasure flags are available, the number of errors that can be corrected increases because syndrome bytes do not have to be used to locate errors. A syndrome byte is the result of some parity check on a received code word. Codes are generated so that syndromes are zero when there is no error and nonzero when they contain information to locate and correct errors in a code word.

As data is read from the disk, the first level of error detection is during 14-to-8-bit decoding. If bytes don't decode properly, erasure flags can be set to aid the correction process at the CIRC1 decoder (see Fig. 4). The CIRC1 decoder uses a (32,28) Reed-Solomon code that has four redundant bytes and can therefore detect and correct two errors, or correct up to four errors if erasure flags are available from earlier processes. The CIRC2 decoder uses a (28, 24) Reed-Solomon code that can also correct up to four errors with erasure flags. Correction algorithms within drives do not always take full advantage of the codes' correction capability. All manufacturers' implementations do, however, use enough of this capability to lower error rates from the 10^{-5} to 10^{-6} level at the disk to the 10^{-11} to 10^{-12} level at the interface.

*(28,24) = (28 information and parity bytes, 24 information bytes). There are four parity bytes.

CD-ROM Error Protection

The CD audio specification is good for reproducing digital high-fidelity sound, since interpolation can be used to reconstruct byte errors that are uncorrectable by CIRC encoding. This approach is not acceptable for digital data, so the mode 1 CD-ROM specification was created to address this deficiency by improving the bit error rates to the 10^{-15} to 10^{-16} range. The CD-ROM encoding process involves the calculation of EDC and parity bytes for each 2048-byte

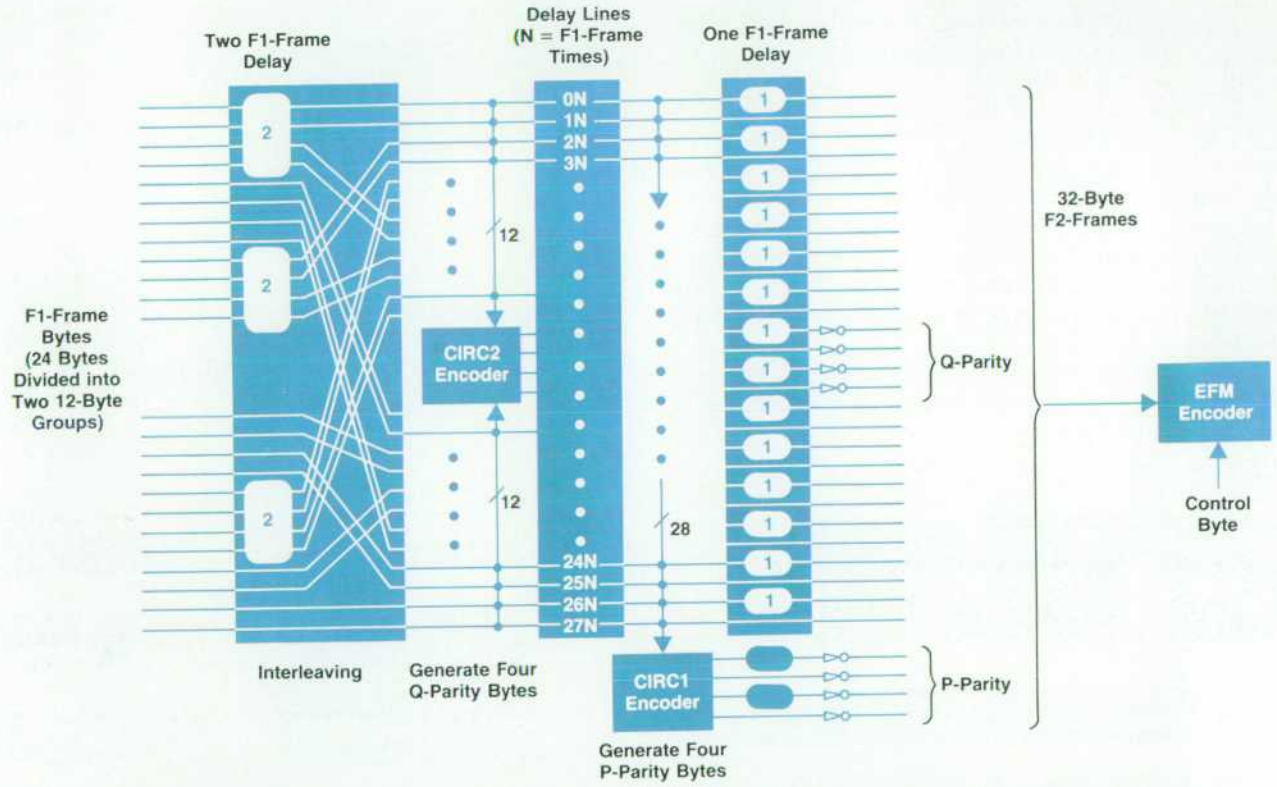


Fig. 3. *CD audio standard CIRC (cross interleaved Reed-Solomon) encoder.*

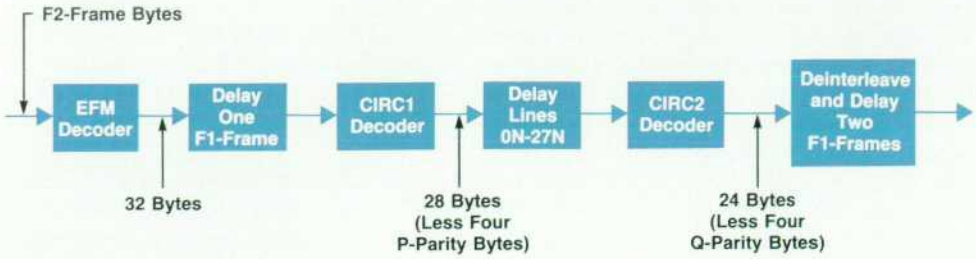


Fig. 4. *CIRC error correction and decoding process.*

sector.

Fig. 5 shows the organization of the 288 EDC/ECC bytes in a mode 1 sector. The EDC portion consists of four cyclic redundancy check (CRC) bytes and eight bytes of zeros. The CRC is a simple checksum that is calculated over the 16-byte header (12 bytes of synchronization plus the four header bytes) and the 2048 bytes of user data. The CRC calculates the value of a polynomial and stores the result in the four-byte CRC field. P-parity and Q-parity bytes, which are not the same as the CIRC parity bytes described above, are calculated using the Reed-Solomon product-like code.[4] In this algorithm error correction is applied to the four header address and mode bytes, the 2048 bytes of user data, the four EDC check bytes, and the eight pad bytes. This totals 2064 bytes. No error correction is applied to the 12 synchronization bytes. The 2064 bytes are arranged in two 1032-byte matrices, one matrix for even-numbered bytes and the other for odd-numbered bytes. For the calculation of P-parity bytes, the matrices are arranged in 24 rows and 43 columns in row-major order. The P-parity bytes are calculated down the columns using a (26,24) Reed-Solomon code and appended to the bottom of the columns, creating two 26-row-by-43-column matrices. Q-parity bytes are calculated along the diagonals of these matrices using a (45, 43) Reed-Solomon code and are appended to the ends of the rows. This calculation produces two 26-row-by-45-column matrices (2340 bytes). Fig. 6 shows these steps. Adding the 12 synchronization bytes results in 2352 bytes, which is one CD-ROM sector. These codes contain enough information to detect and correct one error per row and column, or to correct up to two errors per row and column if erasure flags are available. Since most drive manufacturers provide erasure flags from their CIRC implementations, error rates of 10^{-15} to 10^{-16} errors per bit can be achieved.

When the CD-ROM is mastered the Reed-Solomon product-like code is applied to the information bits before the CIRC1 and CIRC2 encoding. During decoding, the situation is reversed with the mode 1 decoding applied last.

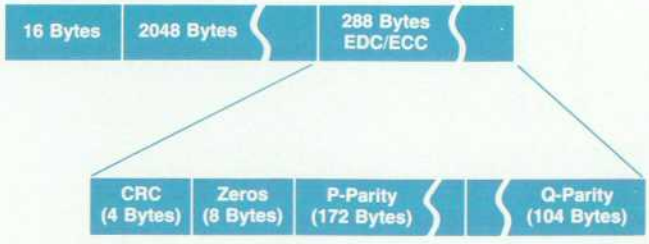


Fig. 5. *Organization of the 288 EDC/ECC bytes in mode 1 sector.*

The Model 600/A Implementation

In the Model 600/A CD-ROM drive, the mode 1 error correction algorithm is implemented in the controller. Except for some HP 3000 computer system requirements, implementing mode 1 error management was straightforward. The most important HP 3000 system requirement that impacted our implementation was a feature that restricted our error correction time, the host watchdog timer. A watchdog timer is a hardware timer on the host that is set at the beginning of an asynchronous operation, in our case an I/O operation, and cleared when the operation is complete. If the timer expires without being cleared, it sets an interrupt to notify the initiator of the operation that the operation did not complete.

The Model 600/A controller's typical response to data errors reported from the drive is to:

- Send any good data that happens to be in the buffer to the host
- Reread the sector in error while capturing erasure flags and building a correction table from them
- Correct errors using multiple read retries if necessary

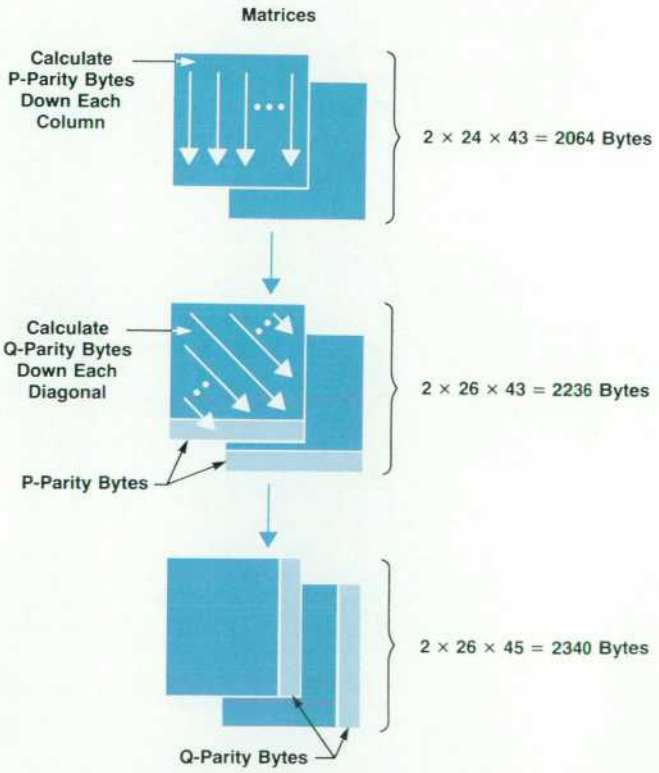


Fig. 6. *Computation of P-parity and Q-parity bytes using the Reed-Solomon product-like code.*

- Transfer the corrected data to the host
- Resume the interrupted transaction.

During such a process, in which the host is generally in the middle of a DMA transfer, the drive controller must send bytes to the host fast enough to ensure that the host does not time out the transfer and shut it down. The Model 600/A design constraint was one second maximum between bytes.

CD-ROM drives originated in the audio world where fast response is not an issue. Controller communication with the drive is via a serial port at 19.2 kbytes/s, and most transactions with the drive can be completed in tens of milliseconds. However, there is no guarantee that the drive will respond to a given command byte in anything less than one second. Also, the drive can take up to five seconds to begin transferring data on a reread. Clearly, some method was needed to avoid host shutdown during error correction.

The solution in the Model 600/A is to hold a sector of data in reserve in the controller's buffer at all times so that if error recovery is invoked, bytes can be sent to the host at well controlled intervals. With this scheme the maximum number of errors to correct in a call to the computationally intensive correction routine can then be regulated to ensure that the host receives a byte about every half second. The controller is still, of course, at the mercy of the drive if it should fail to respond in a timely manner because of focus failure* or other mechanical problems. However, this approach normally allows plenty of margin for the host watchdog timer and provides the controller with enough time to do up to twenty read retries if necessary.

Several other features of the Model 600/A implementation help to expedite error recovery. Wherever possible, computations within the ECC algorithm are replaced by table lookup. These include multiplications by alpha (the primitive element associated with the Reed-Solomon code), base alpha logarithm and antilogarithm calculations, address calculations for matrix access, syndrome association table indexing, and shift operations during CRC checksumming. Also, all error table traversals are coded to minimize table reordering as errors are corrected and deleted from the tables. This means that tables are traversed from tail to head—not a revolutionary idea, but one that saves a lot of time when errors can be corrected as they are encountered. Since it takes three or more errors in a row or column to prevent correction, a situation that multiple data interleaving is designed to minimize, most errors can be corrected during the first pass through the correction table and no table reordering is necessary. As a result, large blocks of errors within a sector, including difficult error geometries, can be corrected in just over one millisecond per error.

Testing and Verification

After CIRC error correction at the drive interface, the CD-ROM is quite error-free. During the course of the Model 600/A development, not a single data error was encountered that was not artificially induced. Nevertheless, third level error correction, as implemented in the controller, is the final defense in ensuring accurate data transfer to the host and it had to be thoroughly tested to make sure that it performs its job properly when needed. The challenge we faced was how to test this error recovery algorithm in a relatively error-free environment.

During the disk mastering process, errors can be introduced into any part of a sector, allowing disks to be manufactured with known error patterns on them. The Model 600/A was tested extensively with such a disk in a test matrix that included all possible combinations of transaction phase, logical sector size, and error position within a transaction. These three system variables affect the overall performance of the correction algorithm, and testing these variables ensured that the controller could correct all error patterns on the disk and would behave properly under all error conditions.

The test disk was designed to contain only correctable error patterns. Since the Model 600/A implementation takes advantage of erasure flags and uses the most robust iterative correction algorithm possible for the Reed-Solomon codes used in CD-ROMs, it can correct up to two errors in any row or column of the data matrix. It follows that an error pattern having nine errors that line up in three rows and three columns can block the correction algorithm. For the correction algorithm used in the Model 600/A, the probability of this happening has been calculated to be $2.84 \times 10^5 \times P_e^9$, where P_e is the probability of a random error at the drive interface after CIRC.[5] This becomes an infinitesimal number when one uses for the value of P_e the random byte error rate that is generally specified by drive manufacturers, that is, 10^{-11} to 10^{-12} errors per bit.

To gain some understanding of how many errors in a sector the Model 600/A controller can handle before unrecoverable errors are likely to occur, a test was developed that reads random sectors from a commercially available test disk, a TD-10 from Laser Magnetic Storage, Incorporated. The test inserts erasure flags and random errors at random locations in the sectors, sends the modified sectors to the Model 600/A controller, and then executes a downloaded ECC routine. The only differences between the downloaded ECC routine and the normal run-time ECC routine are in the interface to the controller's buffer and the error reporting mechanism. Otherwise, the downloaded routine makes calls to the same modules that the run-time routine uses. The following is the unrecoverable data test algorithm.

```
Loop
    Generate a random block address within Mode 1 region on TD-10
    Do long read of sector at random address
    Insert flags between bytes, set to no_error
    Generate random number of errors (1 - 120)
    For each error
            Generate a unique random location within the sector
            Generate 2 random byte error masks
            XOR the odd and even bytes at location with error masks
            Change flags to error for odd and even bytes
    End for
    Write altered sector to the controller buffer
    Execute ECC download
    Log results
End loop
```

Note that errors are generated in pairs that are eventually

*A hardware problem that might be caused by a dirty or damaged disk.

Error Detection and Correction Primer

The concept of parity is familiar to most computer users, particularly if they have ever had the experience of receiving garbled data on their screen because the wrong terminal straps were set, or have had the message "memory parity error at ff0977a8, core dumped" appear on their screen and wished that last file had been saved more recently.

Parity, in its simplest sense, is just the modulo-2 addition (exclusive OR) of all the bits in a transmitted codeword. This includes all the information bits plus a redundant bit that can be set to make parity odd or even. Simple parity can detect a single error since any one bit that changes in the transmitted codeword will change the parity of the received codeword. However, if there are two errors in transmission, parity will be restored at the receiving end and the errors will go undetected (see Fig. 1). Simple parity, then, is useful only in systems in which the probability of error is so small that the probability of two errors is extremely small.

Simple parity can be extended to enable not only detection, but also correction of single errors. If there are k information bits in the transmitted codewords, then for every k codewords we can add a redundant codeword that is made up of redundant bits for each bit position in the previous codewords. In other words, we deal with information bits as a k × k matrix and add redundancy to both rows and columns. Now if a single bit in one of the set of k+1 codewords changes, the parity of exactly one row and one column of the received codeword set will be wrong. This locates the bit in error, which can then be changed to correct the error. This system is illustrated in Fig. 2. It is known as a product code and is guaranteed to be able to correct single errors and detect double errors within the set.

Another method of detecting and correcting errors is to use a linear block code. These codes, known as (n,k) block codes, have codewords that are n bits long, containing k information bits and n−k redundant bits. Without going into great detail, these codes consist of codeword vectors, **C**, that are obtained by multiplying the information vectors, **I**, by the k × n generator matrix, **G**, that is, $\mathbf{C} = \mathbf{I} \times \mathbf{G}$. The generator matrix creates the bits to add for parity. The received codeword vectors, **R**, are assumed to be the sum of the transmitted vectors and an error vector, **E**, so that $\mathbf{R} = \mathbf{C} + \mathbf{E}$. Associated with the generator matrix, and algebraically related to it in a specific way, is a parity check matrix, **H**, which has the nice feature that for all codewords, $\mathbf{C} \times \mathbf{H} = \mathbf{0}$. An added attraction that arises from this fact is that,

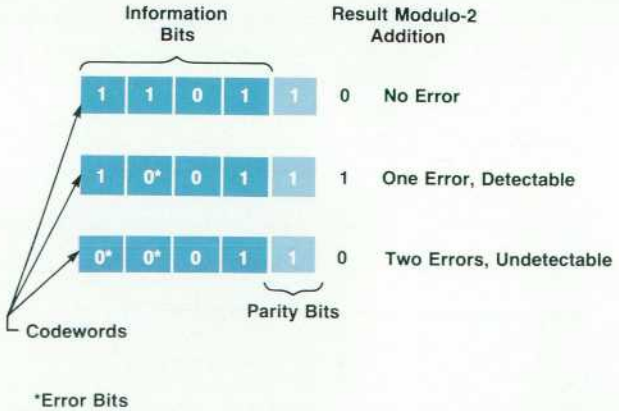


Fig. 1. *Simple parity example.*

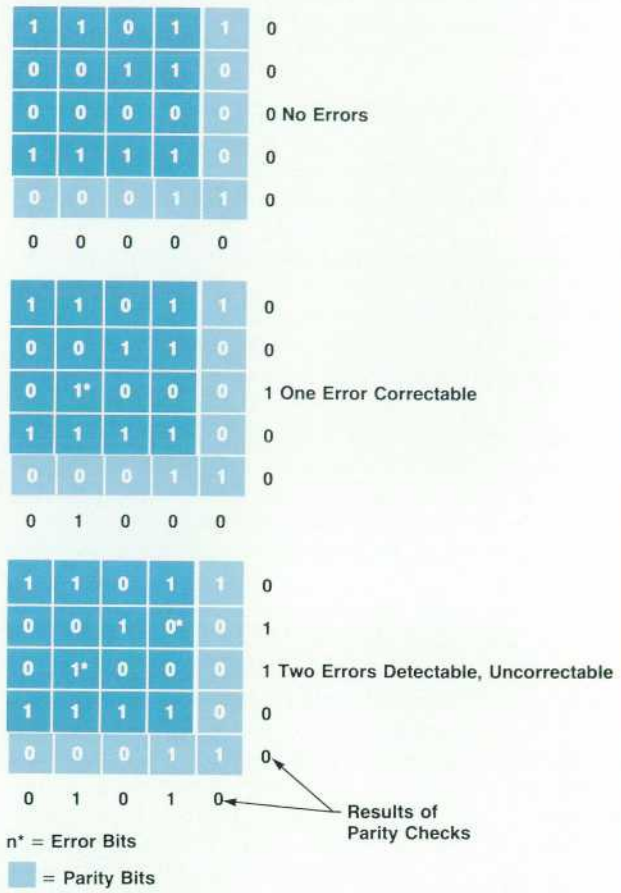


Fig. 2. *Product code parity example.*

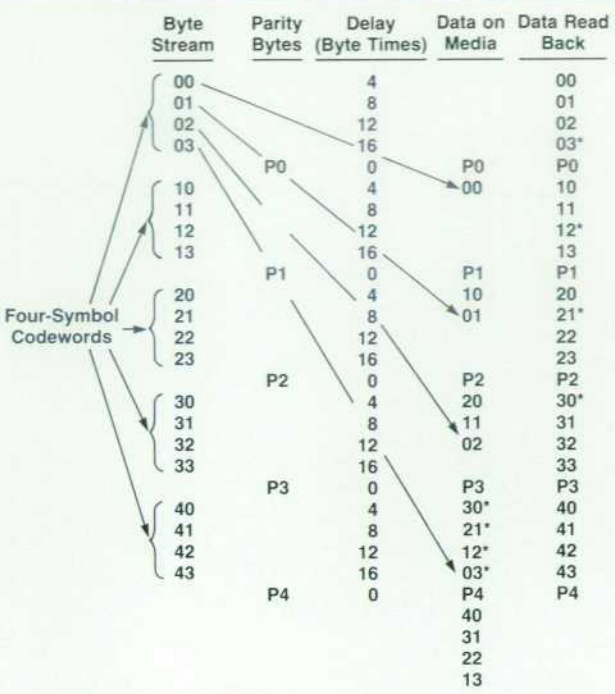
if $\mathbf{S} = \mathbf{R} \times \mathbf{H}$ does not equal zero, it contains information to locate and correct errors in the received codeword because $\mathbf{S} = (\mathbf{C} + \mathbf{E}) \times \mathbf{H} = (\mathbf{C} \times \mathbf{H}) + (\mathbf{E} \times \mathbf{H}) = \mathbf{0} + \mathbf{E} \times \mathbf{H}$ and $\mathbf{E} \times \mathbf{H}$ contains the error correction information. **S** is known as the syndrome vector (a vector that is the result of a parity check), and its power to detect and correct errors depends on the number of redundant bits in the code. A Hamming linear block code is illustrated in Fig. 3.

One characteristic of mass data storage is that it is usually byte-oriented, at least at the controller and interface levels. Codes can be built on finite fields other than the binary field, and the

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \leftarrow \text{Generator Matrix}$$

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \leftarrow \text{Parity Check Matrix}$$

Fig. 3. *A Hamming (7,4) linear block code.*



*Bytes Containing Error Data

Fig. 4. *Time-domain interleaving example.*

Byte Stream	Parity Bytes				Data on Media	Data Read Back			
00	00	01	02	03	00	00	01	02	03
01	10	11	12	13	01	10	11	12	13
02	20	21	22	23	02	20	21	22*	23*
03	30	31	32	33	03	30*	31*	32	33
10	P0	P1	P2	P3	10	P0	P1	P2	P3
11					11				
12					12				
13					13				
20					20				
21					21				
22					22*				
23					23*				
30					30*				
31					31*				
32					32				
33					33				
					P0				
					P1				
					P2				
					P3				

*Bytes Containing Error Data

Fig. 5. *Spatial interleaving example.*

codes used in optical media, Reed-Solomon codes, are constructed on the Galois field GF(256) or GF(2^8), a field of 256 symbols. Generator and parity check matrices are constructed differently than for linear block codes, but are used in much the same way. However, codewords consist of n 8-bit symbols rather than n bits, and all arithmetic is done modulo-256 rather than modulo-2. Decoding and correction are more complex than for linear block codes, but the principle is the same.

Basically, the Reed-Solomon codes are effective on random byte errors. However, errors in the mass storage environment tend to occur as bursts across multiple bytes. To help alleviate this problem and make the codes more effective, one further technique is used on optical media. This is called interleaving. On CD-ROM media, data is time shifted and interleaved as it goes into the channel (i.e., as it is mastered). Note in Fig. 4 how any n-symbol burst can be fully corrected with a single error correcting code since any given codeword has only one error in it. A different method is employed on magnetooptical media. Here every nth symbol in the data stream is used to form one of n codewords. This turns out to be more of a spatial interleaving as illustrated in Fig. 5. These interleaving techniques tend to randomize burst errors and extend the effectiveness of the correction codes.

Bibliography

S. Lin and D. Costello, *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, 1983.

W. W. Peterson and E. J. Weldon, *Error Correcting Codes*, MIT Press, 1972.

A. M. Michelson and A. H. Levesque, *Error Control Techniques for Digital Communication*, John Wiley & Sons, 1985.

N. Glover and T. Dudley, *Practical Error Correction Design for Engineers*, Data Systems Technology, 1988.

distributed to the odd and even correction matrices. This simulates drive hardware that can only detect errors within di-bytes* and can only create erasure flags for byte pairs. Fig. 7 is a graph of the results of one test run that performed 25,150 passes through the loop. The number of errors on the x axis is the number of error pairs that were generated for each 2048-byte sector. These results show that there is still a ninety percent chance of correcting sectors that have ten percent data corruption.

What does this mean to the user? These error simulation results indicate that disks with byte error rates as high as 10^{-1} after CIRC still have a very good chance of being read correctly on the first pass through the error correction routine. Since its overall error recovery effort also includes multiple read retries that attempt to fill in the blanks on sectors that are unrecoverable, the Model 600/A offers the user the maximum in error recovery for CD-ROM technology.

*16-bit words

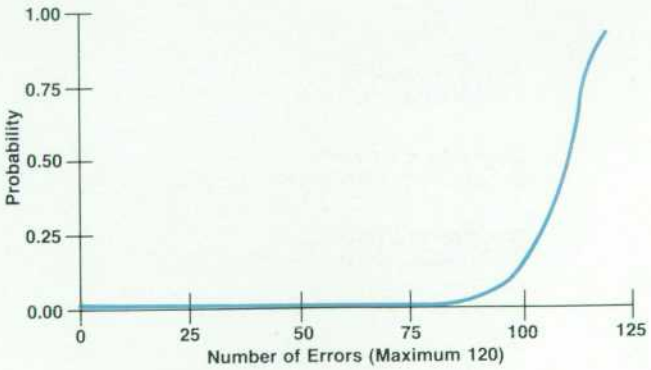


Fig. 7. *A graph resulting from 21,150 passes through an unrecoverable data test algorithm. The graph shows the probability of unrecoverable data based on the number of errors.*

Acknowledgment

I would like to thank Luis Gomez, a student employee during the summer of 1989. He worked on the test routine and compilation of the test results for the Model 600/A.

References

1. *Compact Disk Digital Audio System*, IEC Publication 908, 1987.
2. H. Hoeve, J. Timmermans, and L. B. Vries, "Error Correction and Concealment in the Compact Disk System," *Philips Technical Review*, Vol. 40, no. 6, 1982.
3. K. Odaka, T. Furuya, and A. Taki, "LSIs for Digital Signal Processing to be Used in Compact Disk Digital Audio Players," *71st Convention of the Audio Engineering Society*, 1982.
4. *Data Interchange on Read-Only 120 mm Optical Data Disks (CD-ROM)*, Standard ECMA-130, July 1988.
5. Y. Sako and T. Suzuki, "Data Structure of the Compact Disk Read Only Memory System," *Applied Optics*, Vol. 25, no. 22, 15 November 1986.

Providing Software Protection Capability for a CD-ROM Drive

The HP Series 6100 Model 600/A drive supports two levels of security for software protection: load-time security, which prevents loading a package without the proper authority, and scrambling data on the disk to prevent reading a protected disk with another CD-ROM reader.

by Kenneth R. Nielsen

AN EFFECTIVE USE of CD-ROMs is for the distribution of very large quantities of software and literature. Before CD-ROM technology, software updates were distributed on tape. This method required the creation of multiple customized tapes for each customer. The tapes contained only the software that the customer had purchased. The security solution with this method was simple—customers only received tapes for the packages they had purchased.

With CD-ROM as the distribution medium, many large software packages can fit on one disk. This capability provides a significant cost savings over the tape distribution method. The problem with using CD-ROMs for distribution is how to give customers many software packages on one disk yet restrict them from using software that they did not purchase. This article discusses some aspects of the HP Series 6100 Model 600/A CD-ROM drive security scheme.

Implementation Considerations

Two security schemes were considered for the HP Model 600/A: run-time security and load-time security. Run-time security requires each package to check the system that it is about to run on. If the system is approved for running the package, the package will continue to run. If the system is not approved, the package will shut down, not allowing the package to run on a system that it was not originally installed on. Run-time security would have been a good method if we did not have the constraints of having to run on existing systems that do not have a method of identifying themselves, and protecting software that cannot be easily modified to use run-time security.

Load-time security does not allow the customer to load a package from the disk without the proper authority. This is the method we decided to use for the Model 600/A. This method satisfies both of the constraints mentioned above. The authority for accessing packages on an HP CD-ROM is a unique password that is shipped to the customer with each disk. This password enables customers to identify themselves uniquely to the Model 600/A CD-ROM drive.

Security Toolbox

There are many opinions on and methods of implementing software security features.[1,2] If we had provided a software security method that software distributors had to use, we would have ended up with a very small number of users. Instead, we decided to implement a toolbox approach. This gives users a box of security tools that can be used independently or not used at all.

The tools provided in the toolbox include:

- The capability to lock and unlock discrete portions of the disk selectively
- The ability to unscramble or decode secured data
- The ability to provide the host with a unique identifier.

The security scheme implemented may be defined in the security information that goes on the disk when it is mastered. This information may also define which host-to-disk commands (Command Set 80 commands) the Model 600/A will accept from the host.

The security information for a disk is located in the disk's system area. When a disk is mounted in the drive, based on the information in the system area, the Model 600/A either forces the implementation of the security scheme or

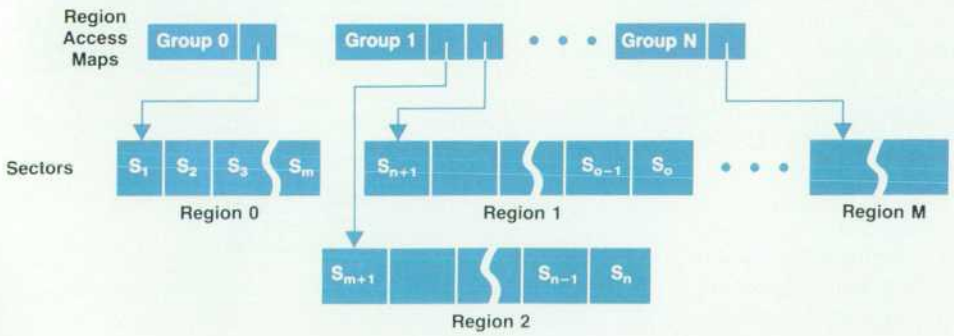


Fig. 1. *Organization of groups, regions, and logical sectors.*

redefines the default values of certain parameters. The default values are used when a new disk is loaded and after a Security Clear command is received from the host.

Region Access Map

The capability to lock and unlock regions of the disk selectively is provided using a structure called a *region access map*, which is located in the system area of the disk. The region access map logically divides the disk into regions. Each region has one or more logical sectors and each region is assigned to a group. Several different regions may be assigned to one group, but a region can only be assigned to one group. Fig. 1 shows this organization. Each region access map entry contains the start address of a region and the group the region is assigned to.

A structure called a *group access map* is used to determine which groups to lock or unlock. A default group access map exists in the system area of the disk. The group access map is a string of bits with the value of each bit representing the default locked or unlocked state of each corresponding group. Groups that are available to everyone will normally have their default value unlocked. Groups that must be individually purchased will normally have their default value locked.

For the customer to modify the group access map to unlock purchased packages, a group access map with the appropriate group representations set to unlocked and a verification password must be sent from the host to the Model 600/A disk controller. The disk controller will do some manipulation on the group access map, the publication identifier from the disk, and the internal identifier of the disk controller. The result of the manipulation is compared with the verification password received from the host. If the comparison proves that the group access map, the disk, and the disk controller all belong together, the customer's group access map is accepted as defining the locked and unlocked groups on the disk. If not, the HP Model 600/A disk controller will use the default group access map located in the system area of the disk. Fig. 2 summarizes this process.

To keep anyone from setting up a computer and sending a variety of verification passwords at full machine speed with the purpose of breaking through the security mechanism, the Model 600/A will purposely delay one second before returning to the host after discovering an incorrect password.

Fig. 3a shows a typical group of files that might exist on a software distribution disk. The operating system is contained in logical sectors 0 through 500, the COBOL compiler in sectors 501 through 600, and the Pascal compiler in sectors 601 through 700. Because of the modularity of the Pascal and COBOL compilers, both use drivers located in sectors 701 through 750. The region access map contains the disk addresses of each file. All the operating system files are assigned to group 0, the nonshared part of the COBOL compiler to group 1, the nonshared part of the Pascal compiler to group 2, and the shared drivers to group 3.

If all customers were allowed access to the operating system but not the COBOL or Pascal compilers, the default group access map would have bit 0, representing group 0, set to unlocked, and bits 1, 2, and 3, representing groups 1, 2, and 3 respectively, set to locked (see Fig. 3b). If the customer had purchased COBOL but not Pascal, the customer's group access map would have bits 0,1, and 3 set to unlocked and all other bits, including bit 2, set to locked (see Fig. 3c). Because there may be hundreds of software packages on a disk, it would be easier if the customer did not have to type in the group access map each time the system needed to be updated. Therefore, the group access map is not scrambled (encoded), allowing the customer to modify the map after receiving permission to access new packages. Allowing the user to modify the group access map does not nullify the security scheme because the group access map and the verification password must be compatible, ensuring that the customer can unlock only purchased software.

When the customer tries to access the disk, a host program will ask the customer for the password that came with the disk. The program will send the group access map and the password to the Model 600/A disk controller, and after performing the comparison process described earlier, the controller will unlock the correct portions of the disk. Once the disk is unlocked, it can be read using any standard CS-80 driver.

If the host does try to access a locked portion of the disk, the Model 600/A will normally respond with a NO DATA FOUND fault. However, there are some system drivers that will abort if this occurs. To solve this problem the Model 600/A is also capable of not responding with the NO DATA FOUND fault and returning to the host a string of meaningless data to complete the transaction so that it seems as if

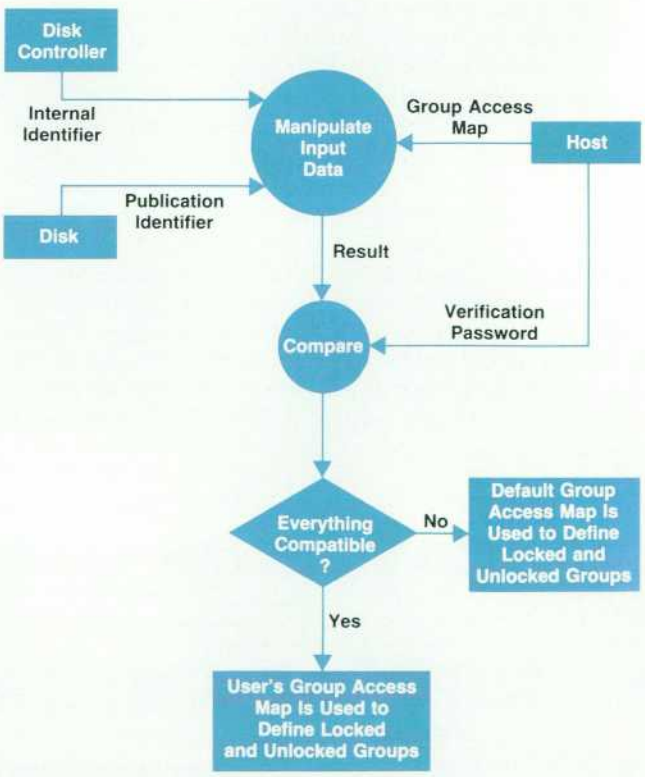


Fig. 2. *Process for determining the locked and unlocked area of a disk.*

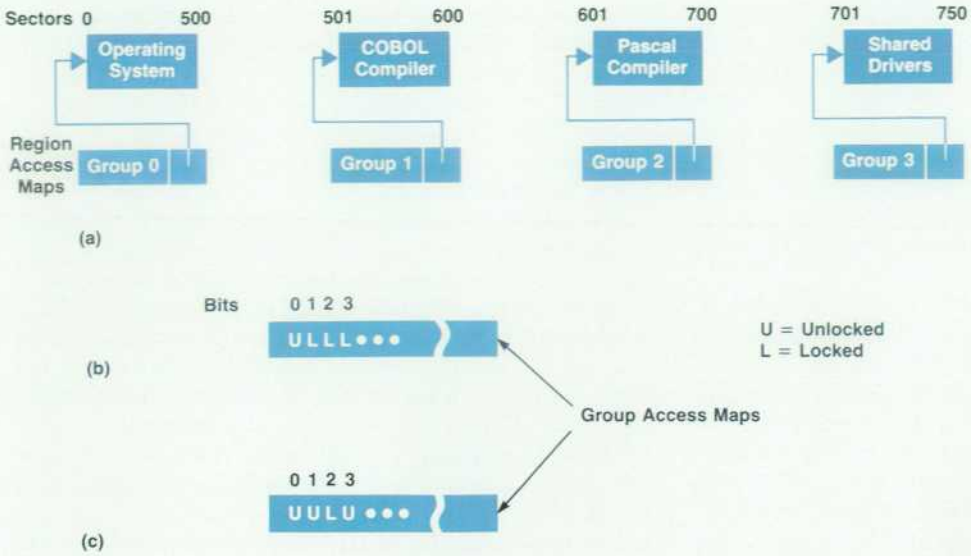


Fig. 3. *(a) A typical group of files on a software distribution disk and their group associations. (b) Group access map showing that the customer has access to the operating system but not the COBOL or Pascal compilers. (c) Group access map showing that the customer has access to the operating system, the COBOL compiler, and the shared drivers.*

nothing happened. (The CS-80 command Data Fill, which is described later in this article, provides this capability). The host can then inquire about the security status to find out if an attempt was made to access a locked region of the disk and that invalid data was transmitted.

Unscrambling Data

The lockable disk is only secure if it is mounted in the Model 600/A CD-ROM drive. To prevent reading the disk from another CD-ROM reader, the data on a distribution disk is scrambled. The Model 600/A can unscramble a disk that has its data scrambled. This option should protect the packages from being loaded via another reader, and provides an extra level of security on top of the group access map and the verification password.

When data is scrambled for security purposes, a deciphering key is also generated. The unscrambling algorithm will return the scrambled data back to readable data only if it has available the same key used in the scrambling algorithm. The Model 600/A's unscrambling algorithm is located on the drive's controller board (see page 40). The key for the Model 600/A is an 8-byte value that can be located either on the disk or sent from the host. If the key is on the disk and scrambled, it is decdoded using a predefined algorithm. If the key is sent from the host, the key will be decoded using an algorithm that is unique to each customer's Model 600/A CD-ROM drive. This scheme allows each of several customers to have a unique key even if they all have access to the same data.

The security tool for unscrambling data can be used in different ways. One method unscrambles either the whole disk or selected portions of the disk when data is read from the disk and sent to the host. Another method involves the host's using the Model 600/A as an unscrambling box. This method can be used only if certain portions of a package are scrambled. If the key used to unscramble the data is on the disk, the default method is to unscramble all data as it is read from the disk (see Fig. 4 switch position 2). If the key is sent from the host, the default method is to read the data and leave it scrambled (see Fig. 4 switch position 3).

To use the Model 600/A as an unscrambling box the host reads a complete scrambled file from the disk and then sends a customer-unique deciphering key to the CD-ROM drive. The host's unscrambling algorithm is a write, unscramble, and read sequence. First the scrambled file is written to the data buffer on the Model 600/A's controller using the CS-80 command Write Buffer (see Fig. 4 switch position 4). Next, using the CS-80 command Unscramble Buffer the host commands the controller to unscramble the data in the buffer using the deciphering key passed down earlier (see Fig. 4 switch position 1). Finally, the host uses the CS-80 command Read Buffer to transfer the unscrambled contents of the controller's data buffer to host memory.

Unique Identifier

If a customer wants to implement run-time security, the Model 600/A has an 8-byte serial number available for the host. The serial number is in the same packed format as bytes two through nine of the HP 46084A HP-HIL ID module.[3] This is the module used for system identification on HP 9000 Series 300 systems. The definition of the Model 600/A's 8-byte unique identifier corresponds to the report

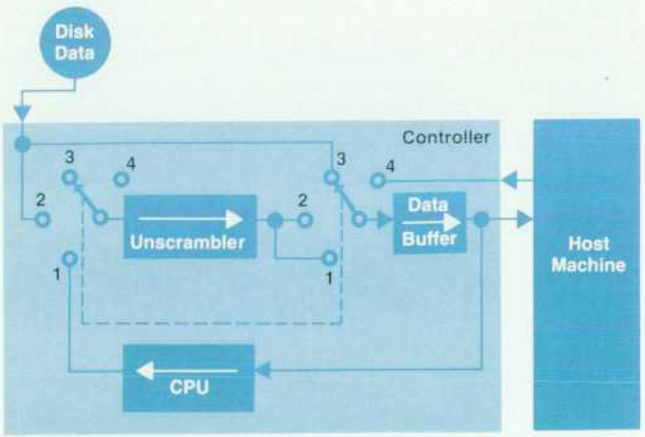


Fig. 4. *Steering unscrambled data in and around the Model 600/A's unscrambler.*

security code definition of bytes two through nine of the ID module.

Command Protocol

The HP-IB Command Set 80 protocol is used for communication between the CD-ROM reader and the HP 3000 MPE VE operating system. To simplify integration and for initial system startup the Model 600/A looks like a write-protected HP 7935A 300-megabyte disk to the HP 3000 MPE VE operating system.

Making the Model 600/A look like a write-protected HP 7935A in most respects was simple. The biggest problem was trying to support the Release command, which frees a disk to be removed from the drive. Without a button on the front panel of the Model 600/A, the customer cannot request that the disk be released. On the HP 7935A, if the customer wants to remove a disk, the front-panel release button is pressed and the HP 7935A executes a release sequence that essentially asks the host if it can release the disk and go off-line, allowing the user to remove the disk and replace it with another disk. The HP 3000 system recognizes this sequence and knows that a disk has been removed and possibly replaced.

On the Model 600/A, if the door is unlocked, the user can remove a disk caddy at any time. It would be meaningless to make a Release request to the host because if the host denied the request, the host would think that the same disk was still loaded. The solution to this problem is that when a disk is removed a report is sent to the host that a new disk of zero length has just been loaded.

The constraint of trying to look like a write-protected HP 7935A meant that commands specific to the security or audio features of the CD-ROM had to be added under the CS-80 Initiate Utility command.

Service

Servicing the Model 600/A posed a potential problem. Since each unit must have a unique serial number that is used to validate passwords and manipulate unscrambling keys, the service engineer must have a means of programming these numbers in the field when a CD-ROM drive's controller board is replaced. The alternative to this would be to return the unit to the factory for repair.

Every repair board has programmed in EEPROM the serial number REPAIRBD and a special seed that is used to generate a unique password verification number. If this serial number is present on a board, the Model 600/A will allow a special service command (Service I) to be executed that programs a serial number into the controller board's EEPROM. It will also cause the unit to derive and program a unique password verification number into the EEPROM.

If the service engineer discovers after programming the controller board that the original controller board should not have been replaced, there is a process to return the repair controller board serial number back to REPAIRBD. The process requires that a special disk be mounted into the drive before a second special service command (Service II) is executed. The combination of the special disk and the bytes sent with the Service II command will reprogram the serial number REPAIRBD and the special seed back into the controller board's EEPROM. If the Service II command is attempted and proves to be an invalid command because the wrong disk is being used or the wrong bytes are sent to the Model 600/A, the controller will either program the EEPROM incorrectly or erase the EEPROM, requiring it to be sent back to the factory for reprogramming.

Utility Commands

The utility commands are CS-80 commands developed to support CD-ROM capabilities, security toolbox functions, and status information relevant to the Model 600/A security scheme. These commands are implemented via the CS-80 Initiate Utility command. The Initiate Utility command was included in the original CS-80 definition to allow the implementation of commands that are not in the formal CS-80 definition but fit into the CS-80 protocol.

CD-ROM Commands. The following CS-80 commands are designed to support the Model 600/A and the features of CD-ROMs.

- Door Lock. Lock the drive's media door to prevent unwanted removal of the disk.
- Door Unlock. Unlock the drive's media door to allow removal of disk.
- Play Audio (length of play) (address of audio portion of the disk where to start playing). Play an audio portion of the CD-ROM. This command will return to the report phase when the audio is finished.
- Play Audio With Return Address (length of play) (address of audio portion of the disk where to start playing). Play an audio portion of the CD-ROM. This command will have multiple execution phases. At the end of each execution phase the address that is currently playing is returned to the host.
- Read TOC (track number). This command will return the TOC (table of contents) entry for the desired track number. The entry returned will consist of the address of that track and the control and address field from the Q channel of the CD-ROM.
- Set Logical Sector Length (sector length). This command will modify the logical length of a logical sector. The options available are 256, 512, 1024, 2048, 2336 and 2352 bytes. The default sector length will be either 256 bytes or the length defined in the system area of the disk. The typical frame of an industry-standard CD-ROM written with computer data contains 16 bytes of header, 2048 bytes of data, and 288 bytes of error correcting code (ECC). The 256, 512, 1024 and 2048-byte sectors will return data from the data field. If the disk has data for which data integrity is not important (e.g., video data), the ECC field may be replaced with 288 bytes of user data. The 2336-byte sector length will return all 2336 bytes of data (the full sector minus the header field). The 2352-byte length will return the full sector. If the CD-ROM is a secured disk, this command is disallowed.

Security Toolbox Commands. These are the CS-80 commands that implement the security scheme for the Model 600/A.

- Data Fill (enable/disable)(fill word). This command will either enable or disable the data fill capability. If data fill is enabled when a locked region of the disk is encountered, the fill word will finish the rest of the current transaction and the NO DATA FOUND fault is not set. If data fill is

disabled, the current transaction will abort when a locked region of the disk is encountered and the NO DATA FOUND fault is set.

- Unscramble Buffer (length of data)(address in buffer where to start). This command will cause the Model 600/A controller's data buffer to be unscrambled with the key that is currently loaded in the controller's unscrambler (Fig. 4 switch position 1).
- Unscrambled Read (on/off). This command will either send the data stream from the disk through the unscrambling algorithm (on) or not (off) before sending the data to the host (Fig. 4 switch positions 2 and 3).
- Read Buffer (length of data) (address in buffer where to start). This command will cause the contents of the controller's data buffer to be returned to the host.
- Receive Data Unscrambling Key (key). This command will cause the key received to be manipulated by the Model 600/A's unique identifier algorithm and then be used as the unscrambling key for future unscrambling.
- Receive Group Access Map (password)(group access map). This command will cause the received group access map to be accessed if the password, the group map, and the currently loaded CD-ROM's identifier all belong together.
- Return Drive Security Number. This command will cause the Model 600/A's compacted serial number to be returned to the host.
- Write Buffer (length of data)(address in buffer where to start). This command will cause the Model 600/A's data buffer to be written into by the host (Fig. 4 switch position 4).

Security Status Commands. The following commands were added to retrieve status information about the CD-ROM and to make the security toolbox easier to use.

- Report Security Quick Status. This command will return one byte that indicates powerfail, disk change, and/or a security fault. This status is cleared either by a Security Clear command or by the execution of the Request Security Status command.
- Request Security Status. This command will return a string of bits indicating the type of disk currently loaded, the security features that are present in the system area of the disk, and the security faults that have occurred. This status is cleared either by the Security Clear command or by the execution of the Request Security Status command.
- Security Clear. This command will cause all security features to return to their default state. The CS-80 Clear command will not affect the security features. The difference between the CS-80 Clear command and the Security Clear command is that the CS-80 Clear command will set the CD-ROM reader and all internal state machines back to power-on conditions. The Security Clear command will set the security features back to either power-on or new disk loaded conditions. Using the Security Clear and the CS-80 Clear commands independently will help ensure that no data corruption occurs.
- Set Security Status Mask. This command will prevent the occurrences that are masked from affecting the Security Status or Security Quick Status commands.

Conclusion

The tools designed into the HP Series 6100 Model 600/A HP-IB CD-ROM drive should be adequate for almost any user who wants to distribute software or data on CD-ROM disks. The disk publisher can tailor the security level to range from no security at all to a very complex security scheme. If the host system wants to build a security driver with a protocol that is similar to the host CS-80 driver, the commands are available to do so. If the disk distributor wants to change the unique customer password verification number, there are hooks built into the Model 600/A to allow that change to be done safely at the customer's site. Essentially, the Model 600/A security scheme provides a good balance between security and ease of implementation for both the distributor and the customer.

Acknowledgments

The HP Series 6100 Model 600/A HP-IB CD-ROM drive project was a joint effort between HP's Greeley Storage Division (GSD) in Greeley, Colorado and HP's Application Support Division (ASD) in Mountain View, California. The desire for such a product was generated by ASD. They had a good idea of the general outline of the product they wanted, but being a support division, they did not have the R&D or manufacturing resources needed to design, develop, and manufacture the product. Therefore, GSD was contracted to supply the product. The security tool definitions were designed by a committee that consisted of Chris Armbrust and Pankaj Shah from ASD, John Santon from GSD, and Steve Hand from HP's Commercial Systems Division. I would like to thank ASD section manager Jeannie Bruins who managed the host implementation part of the project, GSD project manager Ed Sponheimer who had the responsibility for managing the product and coordinating with ASD, Chris Armbrust of ASD who provided valuable insights about customer use and what the final product should look like, Mark Cousins and Pankaj Shah from ASD who were the interface team for the HP 3000 operating system, and finally the GSD development team of John Meyer, firmware designer for drive control and error correction, John Santon, hardware designer, and Bob Proctor, mechanical designer.

References

1. T.A. Rullo, *Advances in Computer Security Management*, Volume 1, Heyden and Sons, 1980.
2. D.K. Hsiao, D.S. Kerr, S.E. Madnick, *Computer Security*, Academic Press, 1979.
3. "Using HP-HIL Devices," *Facilities for Series 200/300/500 HP-UX Concepts and Tutorial*, HP publication number 97089-90081.
4. *CS-80 Instruction Set Programming Manual*, HP publication number 5955-3442.

Support for the ISO 9660/HSG CD-ROM File System Standard in the HP-UX Operating System

To allow HP-UX users access to CD-ROM files, the ISO 9660/HSG file system format standard has been incorporated into the HP-UX 7.0 operating system.

by Ping-Hui Kao, William A. Gates, Bruce A. Thompson, and Dale K. McCluskey

THE CD-ROM IS a very cost-effective and versatile electronic distribution medium. It provides large capacity (600 Mbytes), longevity, low cost, multimedia (audio/video) capability, read-only protection, and random accessibility.

The ISO 9660 standard[1] and the High Sierra Group (HSG) working paper[2] describe file system formats for publication and distribution of information on CD-ROM media. CD-ROMs and the ISO 9660/HSG standard have become well established in the MS-DOS environment. Currently, there are large amounts of personal computer software distributed on CD-ROMs. Microsoft's MS-DOS CD-ROM extensions[3,4] provide the capability to access files on CD-ROMs in the UNIX* environment.

This paper describes HP's design, implementation, and support for the ISO 9660/HSG CD-ROM file system in the HP-UX 7.0 operating system kernel. The CD-ROM file system is an implementation of MS-DOS CD-ROM extensions on HP-UX operating systems. After mounting a CD-ROM that adheres to the CD-ROM file system standard, files on the CD-ROM are accessible through normal HP-UX system calls and commands, allowing users and application programs to take advantage of the high capacity and low duplication cost of this medium without the need for any special programmatic interface.

Design Goals

The primary objective of incorporating the ISO 9660/HSG CD-ROM file system into the HP-UX operating system was to provide easy access to this new medium so that applications including HP product distributions could take advantage of CD-ROM technology. Based on this and other objectives the design goals for the CD-ROM project included:

- Mounting. The user must be able to mount ISO 9660/HSG file systems under the HP-UX system. The files on the CD-ROM are then accessible through normal HP-UX commands and system calls like those on the HP-UX native high-performance file system (HFS).[5] The user must also be able to mount other file systems such as the network file system (NFS) under directories in the CD-ROM file system.
- Networking. Files on an ISO 9660/HSG-compatible CD-ROM must be accessible via supported networking services such as NFS and RFA (remote file access—an HP-proprietary network service for file accessing).
- Application Programs. An application program that is functional in a read-only mode in an HFS environment must also be functional in a CD-ROM file system environment unless it is dependent on some features of the HFS.
- ISO 9660 versus HSG. Although the ISO standard is the official CD-ROM file system format standard, many older CD-ROMs conform to the format defined in the HSG working paper. The HP-UX must hide any differences between these two formats from the user.
- Adherence to HP's Diskless Semantics. One of the major features in HP's implementation of diskless workstations is that all diskless clients in a cluster have the same view of files on all mounted file systems. This feature must be preserved with the addition of the CD-ROM file system.
- Performance. The I/O transfer rate should be as close to a CD-ROM drive's maximum rate (150 kbytes/s) as possible when the benefits from buffer caching are excluded. The number of disk seeks must be minimized. File system buffering and path name caching must be used to help overcome the transfer rate and seek time (up to 1

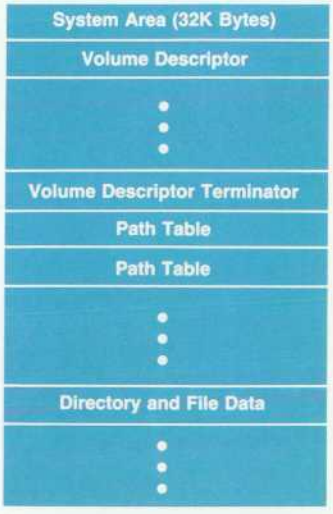


Fig. 1. *The ISO 9660/HSG data layout on a CD-ROM disk.*

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

s) limitations of CD-ROM drives.

- Configurability. Provide the user with the capability to remove CD-ROM-file-system-specific code from the kernel if it is not required.
- File execution. Programs recorded on an ISO-9660/HSG formatted CD-ROM must be able to be executed directly. Executable files in demand-loaded form should be executable as well as shared text executables and regular executable files.
- Level of Implementation. The resulting implementation must conform to level 1 implementation and level 2 interchange according to ISO 9660. These levels specify that access is limited to a file system that is described by a primary volume descriptor and that contains single-section files. Single-section files and volume descriptors are described later.
- Quality Implementation. The CD-ROM file system code must have good quality and be easy to maintain. Techniques, such as structured design, should be used to help achieve this goal.

ISO 9660/HSG CD-ROM File System

The overall data layout for an ISO 9660/HSG CD-ROM file system is shown in Fig. 1. There are typically four sections in this layout: system area, volume descriptor, path table, and directory and file data. The system area and the volume descriptors must occur in the order shown.

System Area. This section makes up the first 16 2048-byte blocks on the media. It is reserved for storing information to boot a system, the secondary loader, security keys, and other system data supplied by the disk preparer.

Volume Descriptor. This section typically contains one primary volume descriptor and zero or more supplementary volume descriptors. Each volume descriptor describes the attributes and structure of a directory hierarchy on the CD-ROM. The list of volume descriptors is terminated by a volume descriptor terminator. These volume descriptors allow multiple directory hierarchies to exist on a single volume (i.e., a physical CD-ROM), or a single directory hierarchy to span multiple volumes.

Path Table. This section contains the path tables for all the directory hierarchies on the CD-ROM. Each record in the path table contains information that enables the system to locate the directory it describes. The path tables do not have to be placed together as shown in Fig. 1. They can be placed anywhere on the disk in whatever manner is acceptable to the data preparer. This is often done to minimize disk access times.

Directory and File Data. This section contains the directory and file data for all the directory hierarchies on the CD-ROM. A directory contains records that describe directories or file sections (described later). Directories are described by two directory records (see Fig. 2). The first record describes the parent directory. The second record and a record in the parent directory describe the directory itself. Fig. 3 shows the structure of a directory record.

A file is divided into pieces called file units which are recorded with file unit gaps between them. For performance, the sizes of file units and file unit gaps can be selected to maximize disk read speed. A file can also be divided into file sections. Each file section has its own directory record. All file sections for the same file must all share the same filename. File sectioning allows a file to span multiple volumes.

An optional data structure called an extended attribute record can be used to specify additional information about the file or directory with which it is associated. An extended attribute record contains information such as the owner and group identifiers, access permissions, and creation, modification, expiration, and effective dates and times. The extended attribute record of a file or directory is recorded before the data of the file or directory (see Fig. 4).

CD-ROM File System Design and Implementation

The structure of the HP-UX file system is based on the abstraction of file systems and file system operations referred to as the vnode layer. The vnode layer was introduced by Sun Microsystems Inc.[6] This structure supports the addition of new file systems in the UNIX environment. The vnode layer had been added to the HP-UX system to support non-HFS file systems (see Fig. 5). Given that it was already in place, it seemed logical to implement the CD-ROM file system under the vnode layer. The advantages of this approach are:

- Modularity. The part of the kernel that supports CD-ROM file systems can be easily separated from the other parts of the kernel.
- Interoperability. The CD-ROM file system can easily coexist with other supported file systems like NFS because CD-ROM files on an NFS server are made available to NFS clients.
- Integration. Other file systems can be mounted on CD-ROM directories. This provides the capability of updating a directory in a CD-ROM by mounting a floppy disc on the directory. Also, additional CD-ROMs can be mounted on a directory to create a large directory hierarchy that might exceed the capacity of a single disk. By mounting these CD-ROMs on different directories the user can have different configurations of the directory hierarchy.
- Resource sharing. The CD-ROM file system can share system resources (e.g., buffer management, name cache, drivers, etc.) with other file systems.

At the vnode layer each file has an associated data struc-

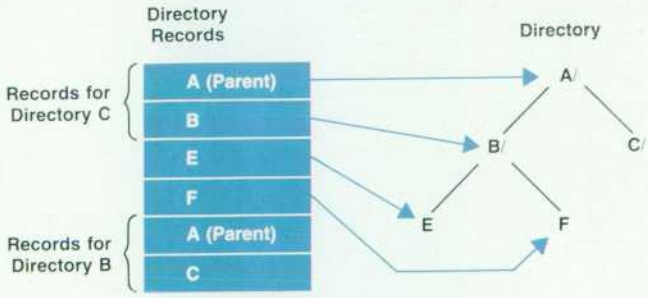


Fig. 2. *The association between directory records and directories and files. Each directory is assigned two directory records, one record for the directory itself and the other for its parent.*

Fig. 3. *The contents of an ISO 9660/HSG directory record.*

ture called a vnode. This is a generic data structure used by all the supported file system types to describe attributes about a file. One entry in the vnode is a pointer to a file system-dependent data structure. Information needed by the specific file system implementation is stored in this structure.

The higher-level layers of the kernel operate on vnodes and low-level file-system-specific operations are encapsulated in separate modules. For example, the main part of the kernel does not know about the CD-ROM file system because that knowledge is encapsulated in the CD-ROM file system code.

CD-ROM File System cdnode

In the HP-UX high-performance file system, each file has an associated data structure called an inode, which resides on the disk. Each inode is assigned a unique number by the system and the structure contains information such as file ownership, file type, internal representation, access permissions, and other data that the system uses to perform operations on a file. In the CD-ROM file system a system dependent pointer in the vnode points to a structure called a cdnode. A cdnode structure stores information similar to that found in an inode but specific to the CD-ROM file system. Unfortunately, inode-like structures with unique numbers do not exist in the ISO 9660/HSG file system. Therefore, cdnode information must be created using directory records and extended attribute records. If a file does not have an extended attribute record associated with it, some of the fields in a cdnode contain default values.

A unique number similar to an HFS inode number had to be defined for the cdnode. Our solution was to use the disk address of the directory record for each directory as the cdnode number. This number uniquely identifies a file or directory because the ISO 9660/HSG format does not support multiple links. In addition, this solution also provides us with the ability to locate the directory entry without any extra calculation.

We encountered one difficulty with using the disk address of the directory record to represent the cdnode number—deciding which directory record to use. A directory can be referenced from three locations in the directory hierarchy: from the directory itself (the "." entry), from a subdirectory (the ".." entry), and from a record in its parent directory (see Fig. 6). We determined early that we could not use the disk address for the directory entry ".." to identify the directory uniquely because there can be many subdirectories. The choice between using the disk address for the "." entry and the directory entry in the parent directory was not obvious until the case of resolving the pathname for the entry ".." was considered. If the reference from the parent directory is used, at least three reads of different directories are needed to perform a pathname lookup of the ".." entry. For example, in Fig. 6 if the current directory is /cdrom/usr/lib and we want to locate the cdnode number for directory /cdrom/usr, the three reads would be:

- Read the ".." entry of the current directory to get the location of the parent directory.
- Read the ".." entry of directory /cdrom/usr to get the location of directory /cdrom, the grandparent directory.
- Read through directory /cdrom to locate an entry that matches directory reference /cdrom/usr. The disk address of this entry would be the cdnode number for directory

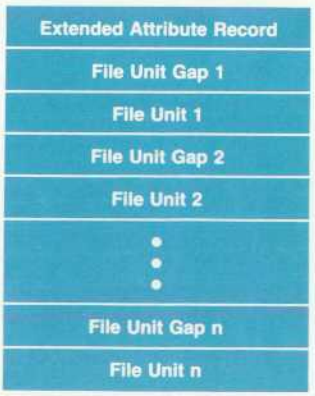


Fig. 4. *A file section record.*

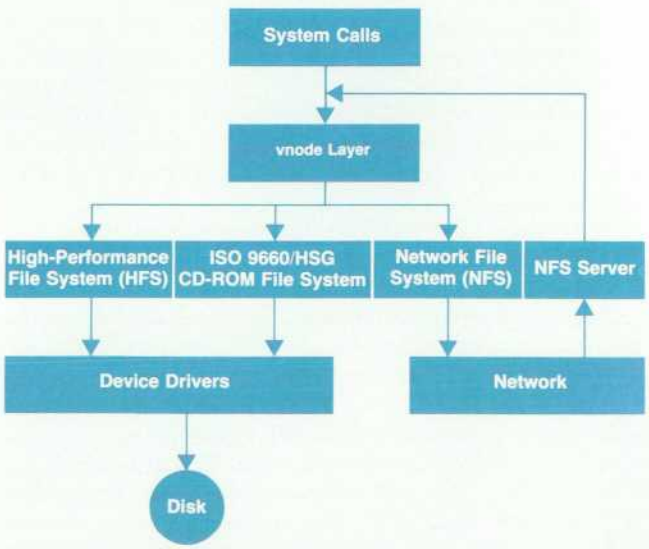


Fig. 5. *The vnode architecture.*

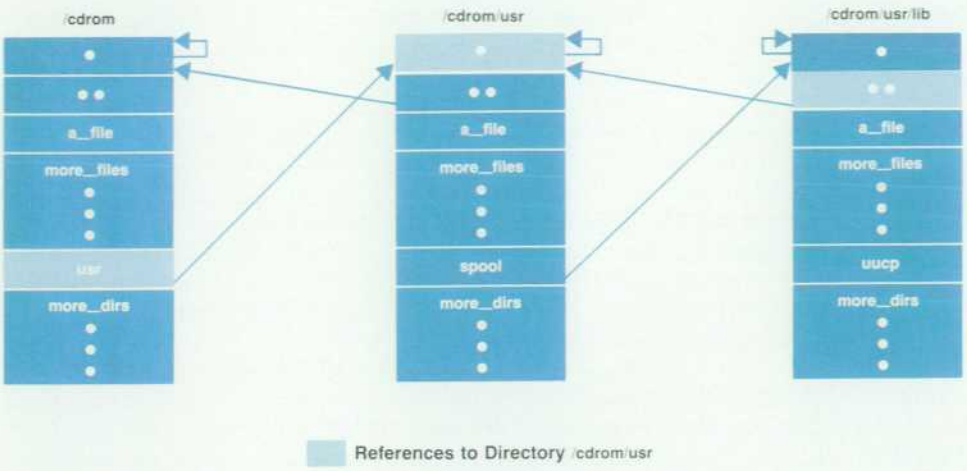


Fig. 6. *Directory references in a typical directory hierarchy.*

/cdrom/usr.

It is much simpler to use the disk address of the directory record "." as the cdnode number to obtain the cdnode number of "..". With this scheme the only operation needed is to read the directory entry ".." and from it the location of the parent directory can be calculated easily. In our implementation the cdnode number of the parent directory is stored in the cdnode whenever possible to reduce the number of read operations.

Pathname Lookup

One operation the kernel performs frequently is the resolution of pathnames. Name resolution is traditionally done one pathname component at a time to provide proper handling of mount points. With the advent of different file system types, traversing the pathname has become even more complicated.

Each vnode is defined by a unique file system type specifier and a set of services required by HP-UX semantics. To perform a pathname lookup on a vnode, the pathname lookup function for that file system is called (e.g., nfs_lookup), returning the vnode for the next component. If a mount exists on this vnode, then changing file systems is indicated, perhaps to one of a different type. The vnode of the root directory for the new file system is obtained. This vnode may contain different file system dependent functions than the previous vnode. The functions at the new vnode are used to continue the pathname lookup process until all components have been parsed. Thus, each file system is allowed to handle its own directory searches, and mounting different file systems on arbitrary directories is possible.

Because of the long execution path and the slow disk seeks, the scheme above could be very time-consuming. To help with this, the directory name lookup cache, a feature that comes with the vnode layer, is used. It caches frequently used pathname elements and their vnode pointers. This significantly reduces the amount of redundant disk accesses made during pathname lookup.

The ISO 9660/HSG format provides a supporting data structure called a path table that is included for pathname lookup. The path table describes the entire directory structure of a file system. This allows traversing the entire pathname with one seek to avoid the lengthy seek time of CD-ROM drives. This method does not allow checking of mount points during the traversal. The path table can be very large, and could consume a large amount of the available memory if kept in main memory entirely. If the path table had to be referenced from the disk, much of its potential benefit would be lost. Although not ideal, most of the path table's performance gain is already provided by the directory name lookup cache mentioned above. For these reasons, path tables are not used for pathname resolution in our implementation.

Backward Compatibility

One major design goal was to minimize the impact on application programs. The strategy to achieve transparent access to the CD-ROM file system was to map, as much as possible, the characteristics of ISO 9660/HSG onto standard HP-UX semantics and characteristics. The first area of concern was the directory library routines because the CD-ROM file system directory entries are quite different from those of the HFS. HP-UX routines use the system call getdirentries to obtain directory entries in a file system independent way. When getdirentries is used to read a CD-ROM file system directory, the fields of the directory records are mapped into the format of a standard HFS directory entry.

Another potential problem area was the system call stat, which returns inode information for a file. To preserve the object code compatibility of existing compiled programs, the stat structure used to return inode information was not changed. When stat is used for a CD-ROM file, information about a file that maps well into the stat structure is passed back and other items specific to ISO 9660/HSG formats, such as file unit size, interleave gap size, and so on are dropped. To obtain the data that does not map well into the stat structure, a new system call fsctl (file system control) was created. Standard HP-UX file attributes, such as user identifier, group identifier, and permissions are optionally specified in ISO 9660/HSG in an extended attribute record. For files without an extended attribute record, the user identifier and group identifier are simply set to an out-of-range number and permissions are set to 0555 (readable and executable by everyone as specified in the standard). Fsctl allows retrieval of information specific to any file system. The reason we rejected the idea of using the system command ioctl in favor of fsctl was that ioctl is intended for control of special devices.

Design and Development Obstacles

During the design and development of the features to support the ISO 9660/HSG standard, the following obstacles had to be overcome.

Disk Block Size. The kernel's buffer management scheme requires that all file system blocks be accessible in DEV_BSIZE (currently 1024 bytes) units. Unfortunately, the minimum size of an accessible CD-ROM disk block is 2048 bytes (a multiple of DEV_BSIZE) according to the CD-ROM standard. For read requests to a CD-ROM, care is taken in the CD-ROM file system code to ensure that reads start on 2048-byte boundaries with sizes in multiples of 2048-byte blocks.

Smaller Logical Blocks. The logical block size of a CD-ROM can be 512, 1024, or 2048 bytes as determined by the data preparer. If the size of a logical block is less than 2048 bytes, the whole 2048-byte disk block containing the logical block must be read from the disk. However, only the logical block is copied into the user's buffer.

Interleaving. To optimize access time and to match an application's expected access patterns, files on a CD-ROM file system can be recorded in interleaved mode for each file, and the size of a file unit and a file unit gap can be random as long as the sizes are multiples of 2048-byte sectors. A kernel routine, cdfs_rd, was implemented to use information in the cdnode such as file location, extended attribute record size, file unit size, and file gap size to calculate the location of each section of a file. If necessary, cdfs_rd also concatenates pieces of data from different file sections into blocks before passing the file back to callers. This routine also tries to maintain the size of a buffer to 8K bytes whenever appropriate so that buffer management efficiency can be maintained.

Demand-paged exec. One major challenge was to support direct execution of demand-paged programs on a CD-ROM. In the current implementation of HP-UX, virtual memory depends on files being physically divided into fixed-size blocks (minimum of 4096 bytes for HP 9000 Series 300 computers), except for the last block. The disk address for each block is kept in the system page table when the files are first executed via the exec call. When a file is paged in, the file's disk address is used to read in the block by calling the relevant device strategy routine directly. Since the files on a CD-ROM can be recorded in interleaved mode and the size of file units can be any number of logical blocks, we cannot rely on the page-in routine to read in pages from the disk directly by calling the device strategy routine. For CD-ROM files, instead of the file's physical disk address, the offset into the file is stored in the page table. With this setup, when the file is paged in, a routine called cdfs_strategy uses the file offset to read in the page by calling the cdsf_rd routine. The cdfs_rd routine hides the fact that the portion of the file containing the page may not be contiguous on the disk.

Diskless Protocol. HP-UX supports diskless clusters, and a client's requests are passed via a lightweight protocol to the server. This protocol assumes there is only one kind of file system. A switch was added to this protocol to accept different kinds of file systems.

Testing and Validation

To ensure compliance with our design goals, the following methods were used to test and validate the CD-ROM file system implementation.

Commercial CD-ROMs. Several commercially available CD-ROMs were purchased to see if they could be accessed through the CD-ROM file system. In selecting the CD-ROMs to purchase, first the mastering companies used to produce the disks were chosen and then at least one CD-ROM was purchased from each. By doing this, the CD-ROM file system code was exposed to different interpretations of the standard.

Regression Testing. The HP-UX kernel is subjected to nightly regression and verification testing by an automated test suite. Tests were added to this suite that mounted a CD-ROM file system volume and verified proper system call behavior. System calls were tested on stand-alone and diskless configurations.

Test Disk. A test CD-ROM was produced that contained many items not available on commercial CD-ROMs. This test disk contained several releases of HP-UX (to investigate the possibility of software distribution on CD-ROM), executable programs from HP 9000 Series 300 and Series 800 computers (in particular, to test demand-loadable executables), huge files, small files, uncommon filenames, and so on. The test disk was particularly useful in testing the demand-loadable executables, but because the CD-ROM manufacturers could not create some of the more exotic data constructs (such as extended attribute records and interleaved files), testing of these constructs was not possible.

Cdgen. Because there are many possible data constructs that are not widely used in the industry today, such as interleaving, multisection files, associated files, and extended attribute records, there was no way to test the paths in the CD-ROM file system code that handles these cases. To solve this, a CD-ROM image generator called cdgen was written to create a simulation of a CD-ROM. Cdgen takes a list of files and creates a CD-ROM image from them. This image is then written to either an HP-UX file or a hard disk. The hard disk can then be mounted and treated as a CD-ROM file system volume. Cdgen supports the following standard data formats and constructs:

- HSG or ISO 9660 format
- Multiple directory hierarchies per volume (primary and supplementary volume descriptors)
- Multiple volume descriptor set terminators
- Interleaving
- Extended attribute records
- Multisection files
- Associated files
- System use, system area, application use, and escape sequence data in all constructs that provide them.

Cdgen was first used to test obscure corner cases such as interleaved demand-loadable executables, directories ending exactly on a sector boundary, zero-length files, and system use data in "." and ".." directory records. Later, automated tests were written that incorporated cdgen, and the tests were added to the kernel regression test suite.

Conclusion

All of our design goals were achieved and the delivered performance matches the speed of the underlying device. The structured design techniques used during design made some sections of the CD-ROM file system code very modular. Pathname lookup is much simpler and more modular than its counterpart in HFS, even though the CD-ROM file system directory structures are more complicated. Although the I/O rate is very dependent on the physical layout of the data on a CD-ROM, it was measured at 149.9 kbytes/s (the maximum for a CD-ROM drive is 150 kbytes/s) without the help of the buffer cache. All nonfile-system-dependent commands and CD-ROM applications from independent software vendors that we tested ran without changes. The quality of this system improved after the last few corner case errors were identified by the images created with cdgen.

References

1. *Information Processing - Volume and File Structure of CD-ROM for Information Interchange*, ISO-9660: 1988(E) Edition, 1988.
2. *The Working Paper for Information Processing, Volume and File Structure of Compact Read Only Optical Discs for Information Interchange*, National Information Standards Organization, May 1986.
3. *Microsoft MS-DOS CD ROM Extensions Function Requests*, Microsoft Incorporated, January 1988.
4. *Microsoft MS-DOS CD ROM Extensions Hardware-Dependent Device Driver Specification*, Microsoft Incorporated, February 1988.
5. M.K. McKusick, W. Joy, S. Leffler, and R. Fabry, "A Fast File System for UNIX," *ACM TOCS*, 2, 3, August 1984, pp. 181-197.
6. S.R. Kleiman, "Vnodes: An Architecture for Multiple File System Types in SUN UNIX," *USENIX Conference Proceedings*, Atlanta, Georgia, Summer 1986.

Authors

December 1990

6 Optical Library System

Donald J. Stavely

Don Stavely is an R&D section manager at HP's Greeley Storage Division. He holds a BSEE degree (1975) and an MS degree in computer, information, and control engineering (1976) from the University of Michigan. Don is currently section manager for the design of optical drives and media. Before attaining that position, he was project manager for the HP Series 6300 Model 20GB/A rewritable optical disk library system. Originally from Pontiac, Michigan, he currently lives in Fort Collins, Colorado. His favorite pastime is windsurfing.

Kraig A. Proehl

Kraig Proehl is a member of the project team that designed the firmware and hardware architecture of the Model 20GB/A autochanger. He has been an R&D engineer with HP's Greeley Storage Division since 1983. A graduate of Purdue University, he holds a BSEE degree (1983). His past projects include design work on the HP 7974 and 7980 tape drives. He is named an inventor for a patent on the HP 7980 tape drive. Kraig grew up in Trotwood, Ohio. He lives with his wife and three children in Loveland, Colorado and enjoys sports, woodworking, and photography.

Mark E. Wanger

Mark Wanger is an R&D project manager at HP's Greeley Storage Division. He was project manager for the HP Series 6300 Model 20GB/A rewritable optical disk library system. He has done mechanical and electrical engineering on several tape drive products, and his work has resulted in six patents for tape drives and magnetic-tape heads. Mark began his career with HP in 1980 at Desktop Computer Division. His educational background includes a BSME degree (1979) from the University of California at Santa Barbara, an MSME (1980) from the Massachusetts Institute of Technology, and an MSEE (1984) from Colorado State University. He was previously with Hughes Aircraft Company where his focus was satellite control systems. Mark was born in Los Angeles, California and currently lives in Fort Collins, Colorado with his wife and two children. He plays volleyball and does woodwork in his free time.

14 Autochanger Mechanical Design

Daniel R. Dauner

Dan Dauner joined HP's Fort Collins Division in 1979 and currently is an R&D engineer at the Greeley Storage Division. He contributed to the design of the HP 7980 and 9144 tape drives, developed the magazine assemblies and mailslot of the Model 20GB/A autochanger mechanism, and was responsible for the mechanical integration of the various autochanger subassemblies. A 1979 graduate of the University of Texas at Arlington, he holds a BSME degree. Dan is a member of the American Society of Mechanical Engineers (ASME). His birthplace is Wellington, Kansas. He, his wife, and his two sons live in Fort Collins, Colorado. He is a public school system volunteer, coaches and plays soccer, bicycles, and runs in his after-work hours.

Jennifer L. Methlie

Jennifer Methlie is a manufacturing engineer with HP's Greeley Storage Division. She has a BS degree in mechanical engineering (1985) from Washington University, St. Louis. Jennifer was one of the design engineers on the HP Series 6300 Model 20GB/A rewritable optical disk library system, and is named an inventor in several pending patents related to that project. Jennifer lives in Fort Collins, Colorado. Her favorite pastimes include skiing, soccer, and golf.

Michael L. Christensen

Mike Christensen is part of the design team for the HP Series 6300 Model 20GB/A and related products for OEM customers. He joined HP's Loveland Instrument Division in 1983, and is currently a mechanical engineer for the Greeley Storage Division. His previous job responsibilities include the design of the HP 3070 board test system at the Manufacturing Test Division, and a year as production engineer at the Loveland Instrument Division's component operation. Mike has a BS degree in mechanical engineering (1983) from California Polytechnic State University at San Lusi Obispo. He is named an inventor on a patent for the design of a fixture latching mechanism, and is a member of the American Society of Mechanical Engineers. He grew up in South Pasadena, California. He, his wife, and his son live in Fort Collins, Colorado.

Raymond C. Sherman

An R&D engineer for HP's Greeley Storage Division, Ray Sherman joined the division in 1988. He contributed to the design and development of the horizontal carriage and its supporting structures for the Model 20GB/A rewritable optical disk library system, and is involved in the development of variations of this product for OEM customers. His educational background includes a BS degree in biology (1974) from Iowa State University, an MA degree in biology (1977) from Drake University, and a BS degree in mechanical engineering (1979) from Iowa State University. Before joining HP Ray worked for Eastman Kodak as a development engineer, for ALCOA as a project engineer, and for Ball Aerospace System Division as a design engineer. He is currently registered as a mechanical engineer in the State of Iowa and is a member of the American Society of Mechanical Engineers. Born in Grinnell, Iowa, Ray is married, has two grammar-school-age sons, and lives in Greeley, Colorado. He is a hunting and hiking enthusiast.

Leslie G. Christie, Jr.

Leslie Christie holds BSEE and MSES degrees from the University of Arkansas. He has been an R&D engineer for HP's Greeley Storage Division since 1984. For the HP Series 6300 Model 20GB/A autochanger, he designed the vertical carriage, mechanical drive functions, and mailslot. Previously, he designed the buffer arm assembly for the HP 7980 tape drive. His work resulted in two patents related to tape drives. Before joining HP, Leslie was an instructor at Kansas State University. Originally from Fayetteville, Arkansas, Leslie is married, has two children, and lives in Greeley, Colorado. His hobbies include gardening, golf, and tennis.

24 Autochanger Servo Design

Mark Bianchi

An R&D engineer for HP's Greeley Storage Division, Mark Bianchi has been with HP since 1984. He is a 1984 graduate of Pennsylvania State University and holds a BSEE degree. Mark designed the calibration and recovery firmware for HP Series 6300 Model 20GB/A autochangers, and was also responsible for the design and development of productivity-enhancing and analysis tools to support that project. His previous projects include designing the read/write channel electronics for the HP 9144A and 9142A tape drives. He was also responsible for the design, layout, and testing of the data compression chip for the HP 7980XC. Mark's professional interests include analog circuit design and control theory. Originally from Vineland, New Jersey, he currently lives in Fort Collins, Colorado. His outside interests include large-format black and white photography, volleyball, soccer, and scuba diving.

Thomas C. Oliver

Thomas Oliver is an R&D engineer for HP's Greeley Storage Division. He holds a BS degree in electrical engineering and computer science (1983) from Ohio State University. Tom is a member of the project team that designed the electronics and firmware algorithms that control the autochanger mechanism of the HP Series 6300 Model 20GB/A. His past projects include designing the servo systems for the HP 9144A and 9142A tape drives. Before joining HP, Tom worked for Liebert Corporation on a microprocessor-based controller design. His major professional interest is high-performance, multivariant control systems. Tom grew up in Columbus, Ohio. He and his wife make their home in Fort Collins, Colorado. His outside interests include windsurfing, weight lifting, basketball, skiing, and blues music.

35 Optical Drive Qualification

Kevin S. Saldanha

Kevin Saldanha graduated with a BS degree in mechanical engineering in 1981 from the Indian Institute of Technology, Bombay, and earned an MS in industrial engineering from the University of Oklahoma in 1984. An R&D engineer for HP's Greeley Storage Division, he is part of the project team responsible for the qualification of the optical disk drive for Model 20GB/A autochanger. Before joining HP in 1985, Kevin worked for Xidex Magnetics and authored a 1984 paper on operations research. His professional interests now center around optical recording. Originally from Mangalore, India, he is married and lives in Fort Collins, Colorado. He enjoys music, the outdoors, and travel.

Colette T. Howe

Colette Howe is an R&D engineer for HP's Greeley Storage Division. She holds a BS degree in electrical engineering (1984) from the University of Utah. She joined HP in 1984 and has contributed to the design of the HP 9153A disk drive and the design and qualification of the HP 9153B, 9123D, and 7963B disk drives. Currently, she is part of the project team responsible for qualification of the optical disk drive for the Model 20GB/A autochanger. Before joining HP, Colette was a technician with Evans & Sutherland Computer Corporation. She is a member of the IEEE Computer Society and her professional interests center around specialty drive qualifications, board layouts, optical devices, and imaging systems. She was born in Salt Lake City, Utah, and she and her husband reside in Greeley, Colorado. Her hobbies include biking, downhill skiing, windsurfing, sewing, and cooking. She also serves as Relief Society President of the Greeley University Branch for the Church of Jesus Christ of Latter Day Saints.

38 CD-ROM Drive

Edward W. Sponheimer

Ed Sponheimer joined HP's Civil Engineering Division in 1977, after receiving a BSEE degree from the University of Arizona in 1976. Ed was the project manager for the HP 9153C 40-Mbyte hard disk drive, and R&D project manager for the Model 600/A CD-ROM HP-IB drive and the Model 650/A rewritable optical stand-alone drive. He is currently the materials engineering manager for optical products. Ed came to HP from Hughes Aircraft Company, where he worked in component evaluation and analysis and test equipment engineering. Ed is a member of the Loveland Youth Baseball Associates Board of Directors. He is married and he and his wife and three sons live in Loveland, Colorado. His hobbies include baseball, golf, bowling, and the study of history.

John C. Santon

Born in Grosse Ile, Michigan, John Santon graduated from the University of Florida in 1980 with a BSEE degree. He joined HP in 1981, working in the Desktop Computer Division. He has contributed to service engineering efforts on the HP 9816 computer, has done software testing on the HP 9133D and 9153A disk drives, and has done hardware design on the HP 9153B and 9153C disk drives and the Model 600/A CD-ROM drive. John is especially interested in applications of software structured design methodology to hardware design. John and his wife and two daughters live in Johnstown, Colorado. Gardening, backpacking, and radio control modeling are some of his leisure activities.

42 CD-ROM Error Correction

John C. Meyer

John Meyer received a BEE degree from the University of Minnesota in 1984, and will receive his MSEE from Colorado State University in 1990. He has been with HP since 1984. He has worked in manufacturing support for small magnetic disk peripherals. For the CD-ROM product, he developed the error correction mechanism. He is currently working on optical drive controllers, mechanism drivers, buffer management, and error correction for a new product. Before joining HP, John spent twelve years as a self-employed carpenter and contractor. He was also a U.S. Navy Seabee, working as a construction electrician. John is married, and has eight-year-old twins, one girl and one boy. He was born in Minneapolis, Minnesota, and he and his family currently live in Greeley, Colorado. He plays guitar, and enjoys rock collecting.

49 CD-ROM Software Protection

Kenneth R. Nielsen

Born in Minneapolis, Minnesota, Ken Nielsen graduated from Dunwoody Industrial Institute in 1968 as an electronic technician. After joining HP in 1969, he attended Colorado State University, majoring in electrical engineering. Ken was design engineer for the

HP 9144A tape cartridge driver, and manufacturing engineer for the HP 97500A/B 3.5-inch hard disk drive. He worked on the firmware design of the channel code and security implementation for the HP 6100 Model 600/A CD-ROM HP-IB drive. He is currently working in manufacturing engineering, supporting firmware for autochangers and optical drives. Ken is married, and lives with his wife and three children in Loveland, Colorado. He plays racquetball, golf, and tennis, and enjoys woodworking and refereeing soccer games.

54 HP-UX CD-ROM Support

Ping-Hui Kao

Ping-Hui Kao was born in Tainan, Taiwan, and graduated from Chung-Yuan University with a BSEE degree in 1978. He received an MSEE degree from Arizona State University in 1982. With HP since 1984, he has worked on HP-UX commands and the HP-UX kernel, particularly on the kernel for the HP 9000 Series 300 file system. Ping-Hui designed and implemented the CD-ROM file system in the HP-UX kernel, and is currently working on new products for supporting CD-ROM technology in HP-UX. Ping-Hui is married, has two sons, lives in Fort Collins, Colorado, and enjoys fishing as a hobby.

William A. Gates

Bill Gates was born in Artesia, New Mexico, and is a graduate of Arizona State University, where he earned a BSEE degree in 1981. He joined HP in 1981, and has worked in HP-UX documentation and HP-UX technical support. He did product testing for the CD-ROM file system, and worked on the development of the CD-ROM file system image generator. He is currently working on conformance testing standards for HP-UX. Bill lives in Fort Collins with his wife and son, and enjoys playing guitar, writing music, and playing tennis, volleyball, and softball. He also does weight-lifting, aerobics, and hiking.

Bruce A. Thompson

Born in Iowa, Bruce Thompson graduated from Iowa State University in 1981 with a computer engineering degree. He started working at HP in 1982. He worked on the HP 7974 and 7978 ½-inch tape drives, and codesigned the DDS format for digital audio tape drives (DAT). Bruce worked on the HP-UX kernel software for the HP CD-ROM drive, and is currently working on the HP-UX kernel drivers for the optical autochanger. Bruce was the codeveloper of a software development tool called HCL (Hierarchy Chart Language), and has written several conference papers. He is single, and lives in Fort Collins, Colorado, where he enjoys photography, amateur radio, and collecting antique soda bottles.

Dale K. McCluskey

Dale McCluskey was born in Loma Linda, California, and started working at HP in 1986. He is currently a technical support engineer for HP-UX at HP's Fort Collins System Division. Dale has also written software for manufacturing and helped add CD-ROM support to HP-UX. Dale graduated from Walla Walla College with a BS in Computer Science in 1986. He and his wife live in Fort Collins, Colorado. He enjoys backpacking, cross-country skiing, and stamp collecting.

63 DTC PAD Support

Jean-Pierre Allègre

Jean-Pierre Allègre has been an R&D engineer for HP's Grenoble Networks Division since 1986. He is a graduate of the Ecole Nationale Superieure d'Electronique et de Radioelectricite de Grenoble and the Ecole Nationale Superieure des Telecommunications de Paris. He is a member of the project team that designed the DTC PAD support architecture and is now pursuing new DTC PAD support functionalities. Previously, he was a quality control engineer for NS X.25 software. His professional interests include networks and languages. Jean-Pierre was born in Grenoble. He and his wife are newlyweds and live in Seyssins. His favorite pastimes include skiing, tennis, and boardsailing.

Marie-Thérèse Sarrasin

Marie-Thé Sarrasin is a senior lab engineer for HP's Grenoble Networks Division. Her educational background includes a BS degree in computer science (1977) from Grenoble University and an MS degree in computer graphics (1982) from Grenoble National School of Mathematics and Computer Science. With HP since 1982, she helped design and qualify HP 1000 software products, HP 3000 MPE V PAD support, and X.25 MPE XL PAD support, and she participated in the investigation for Telnet/XL and the new DTC architecture. Before joining HP she was an analyst for Mutte, a transport company. A native of Bourgoin-Jallieu, Marie-The now lives in Rives, where she serves as town councillor. She is married and has two young children. She enjoys traveling, mountain climbing, skiing, running, reading and writing poetry, and telling stories to her children.

74 Message Interface

Frédéric Maioli

When schoolteacher Frédéric Maioli purchased a Texas Instruments Ti57 calculator he realized he had missed his vocation, programming. Whereupon he re-enrolled in school, graduating in 1986 from the Ecole Nationale Superieure d'Informatique et de Mathematiques Appliquées de Grenoble. Frédéric has been an R&D engineer for HP's Grenoble Networks Division since 1986. His most recent project was the DTC PAD support project. Before this he designed a configuration expert system for the HP 2334A PAD. His professional interests include functional, logic, and object-oriented programming. Born in Lyon, Frédéric has a five-year old child and currently lives in Grenoble. During his leisure hours he enjoys listening to music and is learning to play the flute. He also enjoys hiking.

88 CuBe Anisotropic Change

Nguyen P. Hung

Nguyen Hung is an R&D project manager for HP's Asia-Pacific Computer Division. He holds a BS degree (1978) and an MSME degree (1979) from the University of Michigan and a PhD ME degree (1987) from the University of California at Berkeley. His research determined which materials are best suited for use in network and signal analyzers. Presently, he is developing a proprietary manufacturing process to be used in the latest HP input devices. Nguyen joined HP's Network Measurements Division in 1979. He is a member of the American Society for Metals and has contributed articles to Experimental Mechanics, Machine Design, and American Machinists. He was born in Vietnam. Nguyen and his wife reside in Singapore. During his free time he enjoys snorkeling, hiking, and photography.

Frank E. Hauser

Frank Hauser recently retired as a professor of mechanical engineering at the University of California at Berkeley.

X.25 Packet Assembler/Disassembler Support in the HP 3000 Data Communications and Terminal Controller

The PAD support software implements the communications protocols specified in CCITT recommendations X.3 and X.29. For performance reasons, the software is in the datacom and terminal controller (DTC) rather than the host MPE XL system.

by Jean-Pierre Allègre and Marie-Thérèse Sarrasin

FIRST RELEASED IN 1986, the HP 2345A distributed terminal controller (DTC),[1] offered connectivity from personal computers, terminals, and printers to a single HP 3000 computer system running the MPE XL operating system. The DTC, connected to the host computer by an IEEE 802.3 local area network (LAN), could be thought of as a remote multiplexer.

From a hardware point of view, the DTC is built around an HP proprietary bus called the Device I/O (DIO) bus. A server card with a 68000 microprocessor handles LAN access and DIO management. Up to six multiplexer cards with Z80 microprocessors can be plugged into the backplane. Adapter cards give each multiplexer card either eight direct-connect or six modem ports. Thus the first version of the DTC allowed up to 48 direct connections to an MPE XL system (see Fig. 1).

From a software point of view, the DTC uses its own minimal operating system, called AOS. AOS is a message-based operating system that manages intertask communication. Each software module is a task in operating system terms. A task interacts with the other tasks by sending messages. The operating system manages a queue of messages. When a message is at the head of the queue, it is dequeued and given to the target task. Then the target task executes and is not interrupted until the message is completely processed. Hardware interrupts are managed by operating system handlers that transform these interrupts into operating system messages.

On top of the IEEE 802.3 LAN, the DTC implements an HP proprietary protocol, AFCP, for the transport layer of the seven-layer Open Systems Interconnection (OSI) model of the International Organization for Standardization (ISO). This transport protocol is optimized for LAN traffic and its peer is implemented in the host computer system. The

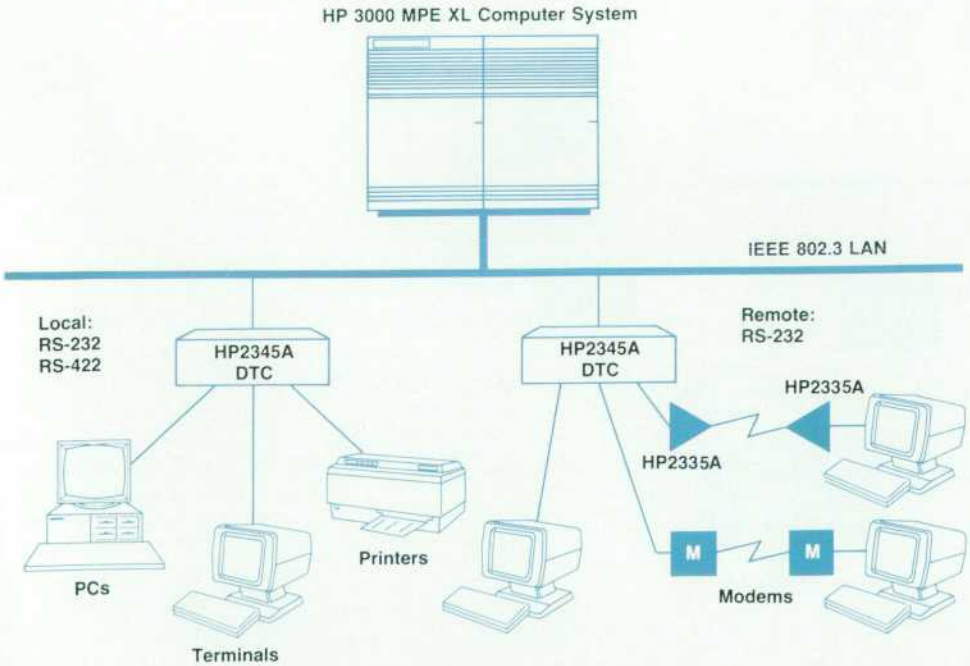


Fig. 1. *The original HP 2345A distributed terminal controller could be thought of as a remote multiplexer.*

transport layer carries terminal I/O requests that are encoded using another proprietary protocol, ADCP, which manages all the terminal read, write, and control functions. It has also been optimized for performance.

The software architecture of the original DTC is shown in Fig. 2. The management task (MGT in Fig. 2) handles all the management requests within the DTC along with initialization. Among its important functions are software download and upload. At initialization, the DTC sends a multicast request. Upon receiving this request, the host starts sending the DTC code through the LAN. This allows easy updates of the code without the need for a ROM change. When the DTC detects an abnormal condition, it is able to upload its entire memory to the host, allowing further study by an HP representative for troubleshooting. The management task also handles all the reset and status requests.

The DIODAM task is in charge of managing the DIO bus interface and also does some ADCP preprocessing.

The MUX task transforms the read, write, and control requests into characters transmitted to each terminal. It also multiplexes several independent data streams to the backplane slots.

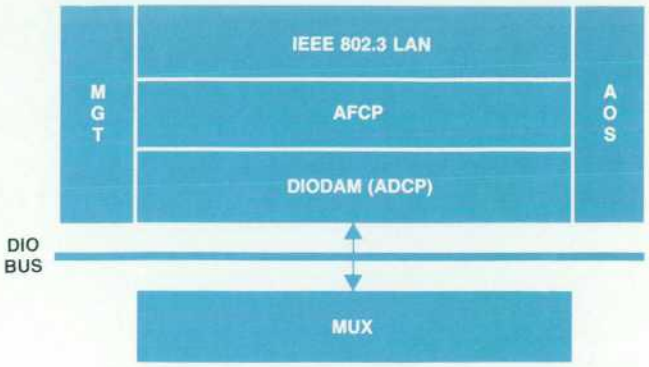


Fig. 2. *Software architecture of the original DTC. AOS is the operating system. AFCP and ADCP are proprietary protocols.*

Second DTC Release

In its second major release, the HP 2345A has been renamed the data communications and terminal controller. It now offers the following new functionalities (see Fig. 3):

- PC-based management[2]
- Access to X.25 packet-switched networks through synchronous network processor (SNP) cards
- A capability for terminals to switch from one MPE XL host to another on the LAN
- Back-to-back access to non-MPE XL systems connected via RS-232-C to another terminal controller (DTC or TS8) on the LAN.

The X.25 access provides for system-to-system communications as well as for remote terminal communications. Up to three SNP cards, which are based on the 68010 microprocessor, can be plugged into the DIO backplane instead of multiplexer cards. SNP adapters of two different kinds allow high-speed, multistandard or RS-232 SNP connections to the X.25 network. Each SNP card can handle up to 256 connections for X.25 packet sizes of 128, 256, or 512 bytes and up to 54 connections for a packet size of 4096 bytes. Access to the LAN is through the server card, as it was in the original DTC.

The second-release DTC is managed through a personal computer using the HP OpenView DTC Manager,[2] an application based on Microsoft® Windows and the HP OpenView

Microsoft is a U.S. registered trademark of Microsoft Corporation.

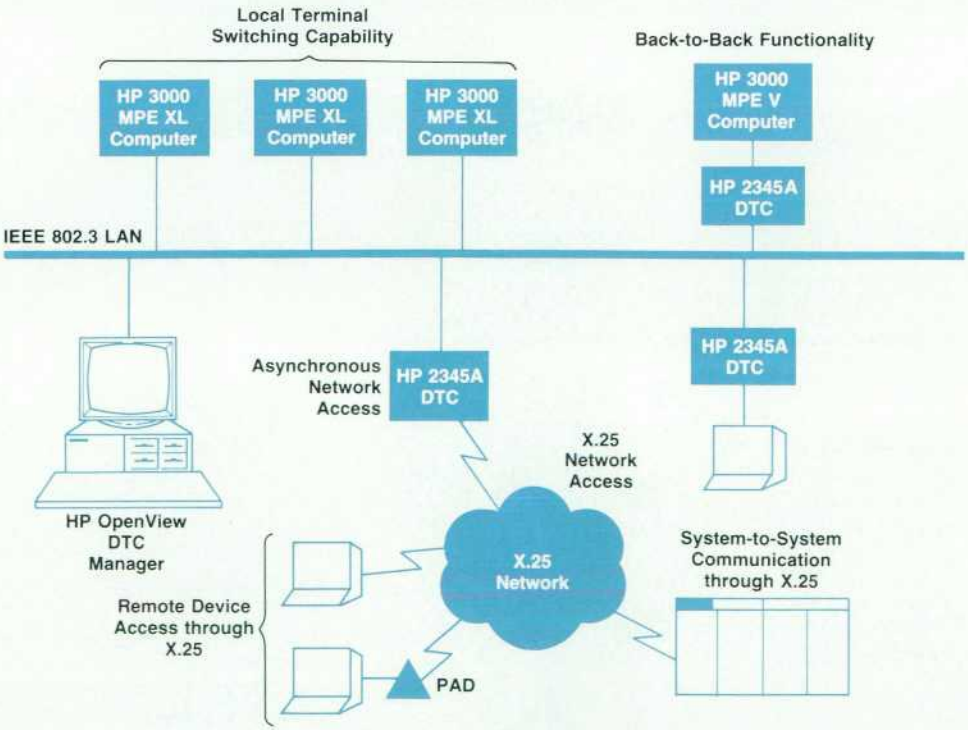


Fig. 3. *The second-release HP 2345A DTC, renamed the data communications and terminal controller, offers X.25 network access, the ability of terminals to switch from one host to another, and back-to-back access to MPE V systems on the LAN.*

network management architecture. The management features include:

- DTC configuration
- DTC download/upload/reset/status
- Board upload/reset/status
- Site management (show connections with information on end-to-end path)
- Protocol status
- Connection upload/reset/status
- Logging and tracing.

The same operating system runs on the server card and on the SNP cards. From the operating system point of view, the server card is the master card and each SNP card is a slave card. Communication between tasks from the server board to an SNP board over the DIO bus is managed by the operating system. A message can be sent to any task from the master board to the slave board and vice versa in a transparent way thanks to the operating system's multiboard communication mechanism. Messages from the server to the SNP are copied by the operating system onto the SNP board. Messages from an SNP task to the server don't need to be copied because the server board can access all of the SNP cards' memory.

To make these functionalities possible, many protocols have been added in the DTC. Fig. 4 shows the possible paths through the DTC and its protocols.

ALCP is an HP proprietary protocol based on the CCITT X.213 recommendation, *An OSI Network Layer Service for Connection-Oriented Protocols*. ALCP offers X.25 access for host-to-host communications. Two operating system tasks, X.25 Level III and X.25 Level II, implement ALCP, the X.25 level III recommendation, LAP-B (X.25 level II), and X.21 bis (X.25 level I). X.25 data is encoded in ALCP messages and transmitted to the host using the AFCP transport layer (level IV) and vice versa.

The terminal switching capability is offered by DTC user interface commands located in the DIODAM task.

The back-to-back functionality is offered with the implementation of the Telnet/TCP/IP stack. Telnet is a DARPA standard network virtual terminal protocol and is implemented in the DIODAM task. ADCP (the HP virtual terminal protocol) has been enhanced to support the added Telnet functionality. The TCP and IP protocols have been implemented as two independent DTC tasks on the server board.

Communication through remote asynchronous PAD devices is implemented in a PAD support task (PADSUP) on the SNP board.

The DTCMGR agent task replaces the management task of the first-release DTC. This task receives terminal and PC requests and transmits them to the appropriate DTC task.

The target name resolution task (TNR in Fig. 4) is in charge of name-to-address resolution using configuration information, HP proprietary protocols (Probe or Name Lookup Protocol, NLP), or standard protocols (Address Resolution Protocol, ARP).

The DTCMGR extension (EXT) task provides the SNP logging and tracing facilities. These functionalities send data to the personal computer running the HP OpenView DTC Manager software, which is able to format and display the data.

PAD Support Functionality

A PAD (packet assembler/disassembler) is a piece of software and/or hardware that allows an asynchronous device, such as a terminal or a printer, to communicate with an X.25 network. PAD support in the DTC provides a host computer with support for terminals connected to public PADs (also called network PADs) and with support for terminals and printers connected to private PADs.

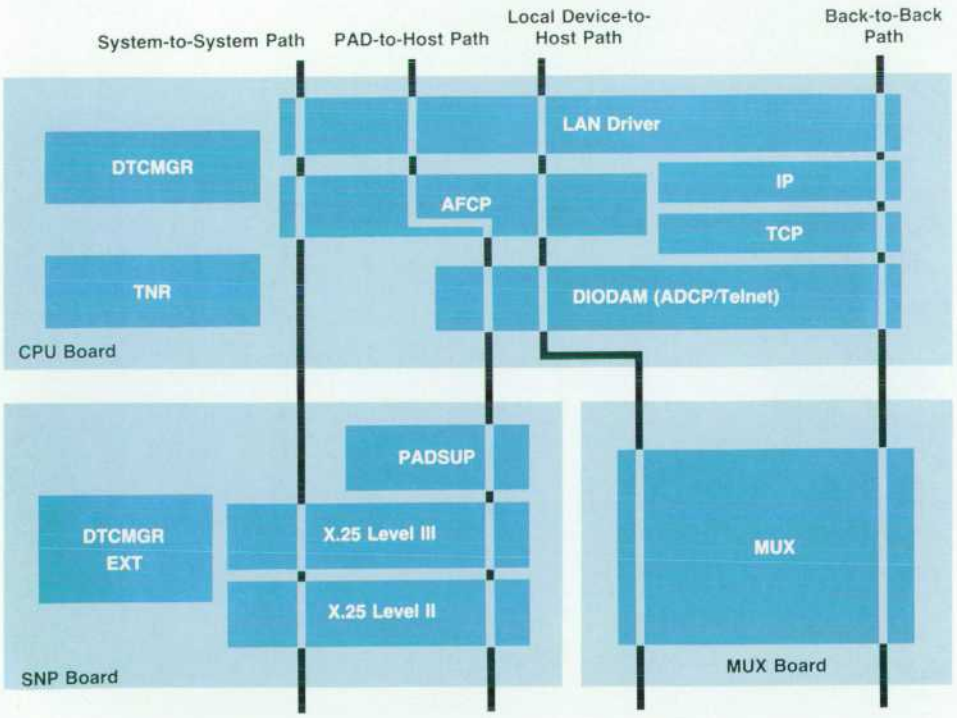


Fig. 4. *Software architecture of the second-release DTC. X.25 access is through SNP cards in the DTC. The PAD support software is on the SNP card. TNR is the target name resolution task. Telnet, TCP, and IP are standard protocols. AFCP and ADCP are proprietary protocols. The path through the protocols depends on the type of connection.*

Communication between a PAD and an asynchronous terminal is described in the CCITT X.28 recommendation. The different parameters necessary to set up the proper terminal profile are defined in the CCITT X.3 recommendation, and communication between the host and the PAD is ruled by the CCITT X.29 recommendation (see Fig. 5).

A private PAD can be thought of as an external PAD. It is connected to a PDN (public data network) as a host and has an X.25 address, but it behaves like a PAD when communicating with another host. HP provides the HP 2334A and HP 2335A X.25 cluster controllers, which allow up to 16 terminals and/or printers to access a network (see Fig. 6).

Applicable Standards

The CCITT has issued X.25, X.28, X.29, and X.3 recommendations in 1976, 1980, 1984, and 1988.

The objective of MPE XL PAD support in the second-release DTC is to support applications by using the X.3 and X.29 recommendations and philosophy. To achieve this, the PAD support must manage the X.3 parameters in such a way that the user does not have to care about them. In the new DTC, the same processing is done for every type of PAD—public or private. The PAD support sets parameter values according to the 1980 X.3 recommendation so that the DTC can support as many PADs as possible.

The PAD support in the new DTC uses X.3 and X.29 to transmit control requests to the PAD. The X.3 recommendation defines a set of couples of parameter references and values to control the asynchronous interface of the PAD. According to the 1980 X.3 recommendation, eighteen parameters are available to manage a PAD:

1	Escape from data transfer and enter a PAD command
2	Echo
3	Data forwarding condition
4	Idle timer (time between two characters)
5,12	Flow control using XON/XOFF
6	PAD service signals
7	Break processing
8	Discard output
9,10,14	Padding after CR, LF, LF line folding
11	Line speed
13	Linefeed insertion after CR
15,16,17,18	Editing parameters.

A simple example is the parameter reference 2, defined for echo processing. If this parameter is set to the value 0 the PAD will not echo the data sent to the network. When the value is set to 1 the PAD is in charge of echoing the data entered on the terminal keyboard to the screen. Therefore, a reference number specifies a functionality (e.g., echo processing) and many values can be attached to this functionality (do echo, do not echo).

In the 1984 X.3 recommendation, four more parameters were added along with the possibility of having network dependent formats for PAD devices. The four added parameters are:

19	Editing PAD service signals
20	Echo mask
21	Parity treatment
22	Page wait.

In the 1988 X.3 recommendation, the main modification is that PAD service signals (prompts and user messages) can be in English, French, or Spanish.

To transmit the X.3 parameters to the PAD, the X.29

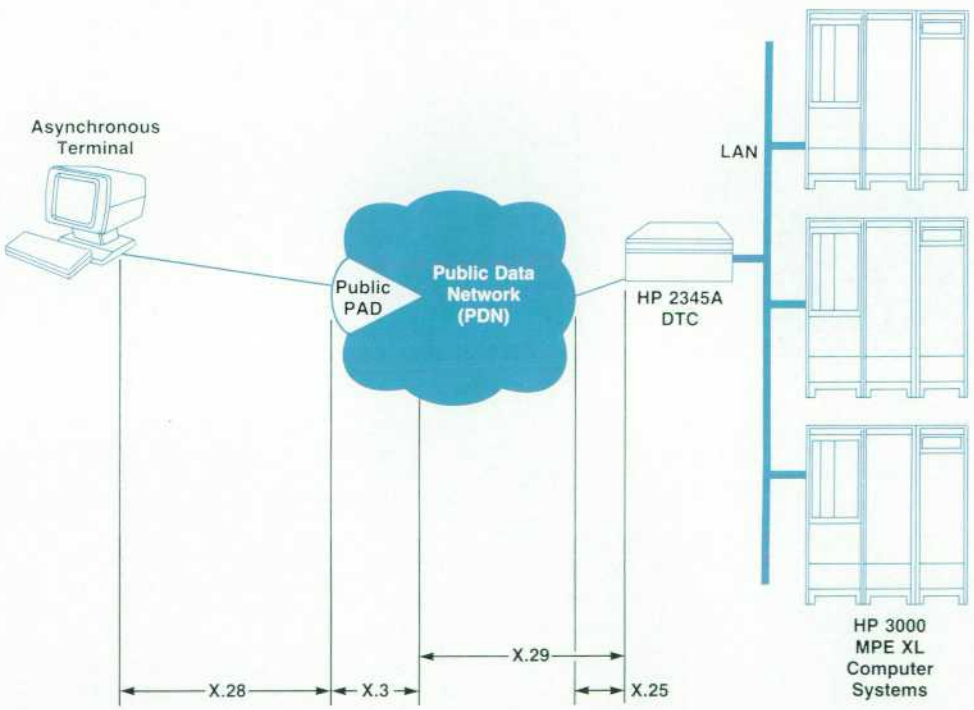


Fig. 5. *Connection through a public or network PAD, showing the CCITT recommendations that govern communications.*

recommendation is used. This recommendation defines the operations a host can perform on a PAD. According to the 1980 X.29 recommendation, the main operations on a PAD are:

Set	Set a profile of X.3 parameters
Read	Request the PAD X.3 parameter values
Set and read	Set a profile and then read it
Parameter indication	Receive the values of the required current PAD profile after a read or set and read
Indication of break	Notify the host that the **Break** key has been pressed and tell the host whether the PAD is discarding data
Invitation to clear	Clear a virtual circuit as soon as all data sent to the PAD has been sent to the device
PAD error message	Indicate a PAD problem (invalid message code received, for example).

A profile is a set of X.3 parameter references and values. All these messages are coded and transmitted using X.25 packets with the qualifier flag set. The usual convention for describing a profile is

[<parameter reference> : <parameter value>,]*

For example, 2:1, 3:2, 4:1 means that parameter 2 has value 1, parameter 3 has value 2, and parameter 4 has value 1.

In the 1984 X.29 recommendation, a new message was added, called the reselection PAD message. It makes it possible to clear an X.25 circuit to a destination after transmission of all the data and to establish a new circuit to a given destination in the same message. In the 1988 X.29 recommendation, this was enhanced to support the TOA/NPI (type of address/numbering plan indicator) address subscription facility defined in the X.2 recommendation. This enhancement is mainly for future extension to ISDN.

DTC PAD Support Architecture

We could have built a PAD support task in the host computer on top of the NetIPC (network interprocess communication) software using the VT (virtual terminal) network service. This was done for MPE V PAD support. For the DTC and MPE XL, we have instead built a task called PADSUP within the DTC, mainly for performance reasons. This decreases the amount of character processing that must be done by the host, since this is done remotely by the PADSUP task in the DTC front end. All X.3 and X.29 processing is in the DTC PADSUP task, so the entire X.25, X.3, and X.29 implementation is within the DTC.

The PADSUP task receives ADCP requests from the DIODAM task as if it were a multiplexer with a physically attached terminal. It transforms these requests into X.25 and X.29 requests through the ALCP interface.

This architecture has many advantages. Remote terminals have the same data path in the host as local terminals, thereby decreasing the time needed for implementation, support, and maintenance. The known ADCP protocol, which was designed for the original DTC, was retained because it is simple and efficient for asynchronous communications, and very few extensions were needed for PAD devices. The ALCP interface, which is already used for system-to-system communication, is used to communicate with the X.25 level III module. The efficient, reliable AFCP transport protocol is used for communication with the MPE XL host, and access to the transport layer is hidden by DIODAM.

The architecture also allows switching capability. Because the user interface is in the DIODAM task, the functionality implemented for the local terminals can be reused by the PAD support task to connect remote terminals to any MPE XL computer on the LAN.

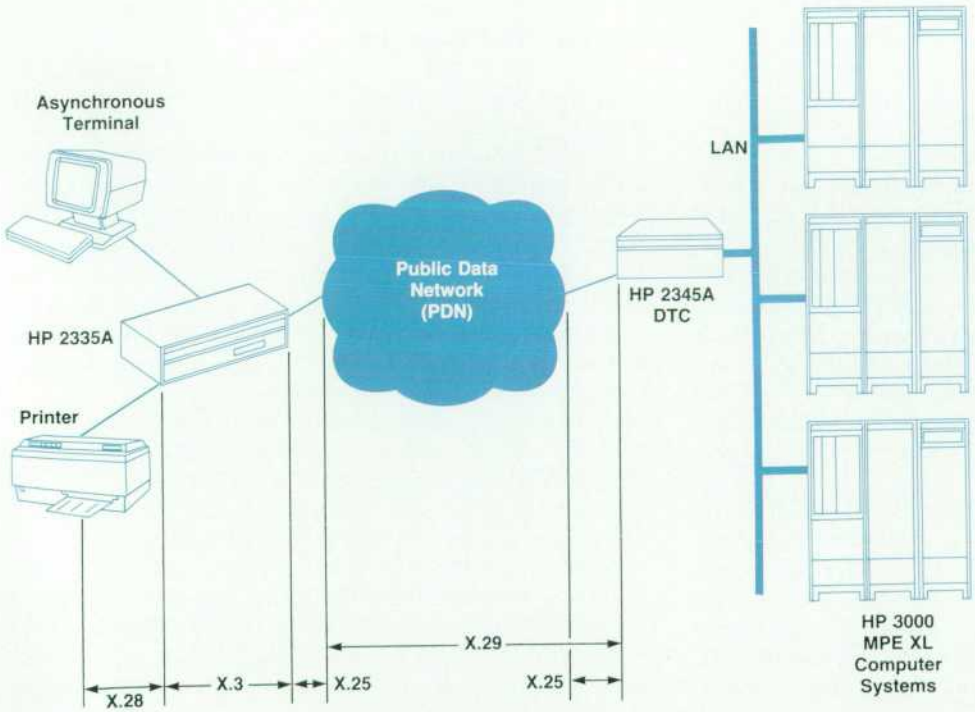


Fig. 6. *Connection through a private PAD such as the HP 2335A.*

Security

Through the X.25 network and gateways, numerous PAD users may try to connect to various systems. This causes security concerns for system and network managers, who need to restrict the use of some systems. For example, when private X.25 networks are tied to public networks, access is often authorized for users within the private networks and restricted for public PAD users. On the other hand, some customers don't care about security. To meet varied customer needs, the DTC PAD support offers flexible security features.

Two kinds of security are provided. First, a management function available from the HP OpenView DTC Manager PC workstation can be used to start or stop the PADSUP task. When the PADSUP task is not started on a card, all PAD calls will be cleared.

The second type of security is a security list based on the X.25 calling address. Users with unauthorized X.25 device addresses are filtered out at the DTC level. A table, configured at the HP OpenView DTC Manager PC, associates X.25 device addresses with host names and passwords. When the user tries to log on with security configured, the password tied to the list must be entered in the user data of the call packet or through the DTC user interface. The user is allowed three attempts to enter the password. If a wrong password is entered or if the node name to which the user wants to be connected is unauthorized, the connection will be cleared. Users who know the password can be allowed access to all systems or only to certain systems. It is also possible to reject all users calling from a particular address. Wildcard digits are available for use with public PADs, where the X.25 address is not relevant.

Testing the PAD

A major problem for PAD support is managing multivendor PADs connected to multivendor devices. The X.3 recommendation defines a set of parameter values that look like driver characteristics. There have now been four successive editions of this recommendation from the CCITT. Each PAD vendor implements only a subset of the latest X.3 parameter values. From experience, we know that even some mandatory values aren't implemented in some PADs. It is always a support nightmare to characterize a PAD problem, especially if the problem is that the system is hung up in the middle of an application.

To alleviate this problem, a PAD test sequence is implemented in the DTC. It is run at connection initialization to verify the X.3 parameter values supported by the PAD. After this test, information is available to inform the network administrator of potential intrinsic mappings that may fail because of PAD restrictions. Once data transfer starts, the PADSUP task always knows what it can and cannot ask the PAD to do.

To map all the intrinsics used in the terminal I/O world, only a subset of the X.3 couples (parameters and values) is needed. PAD support in the DTC tries to set all the values at connection initialization that will be needed during the connection to support all the applications. If a parameter is not supported by a PAD, an error is logged and the DTC PAD support never sets this parameter again, thereby avoiding further problems. This implementation exposes all the multivendor PAD problems at connection initialization, which seems the best point to detect problems.

The PAD test is performed by the PAD support task just after the X.25 call confirmation packet is received by the DTC from the host (host calling terminal) or sent by the DTC to the host (terminal calling host). The sequence is as follows:

1. Identification of PAD parameters. An X.29 read of all PAD parameters is sent to identify the parameter numbers supported or not supported by the calling or called PAD.

2. PAD test. An X.29 set and read of a given set of X.3 couples is sent to the PAD. These couples consist of the parameter numbers supported by the PAD with the parameter values required by all types of applications. To minimize the number of messages needed to determine the characteristics of the PAD, the first X.29 set and read message sets alls the values that may be used for each parameter. For example, the profile of couples might be 1:0,1:1, 2:0,2:1,3:0,3:2,4:0,4:1,4:10, and so on.

The PAD support task then waits for an X.29 parameter indication, in which all unsupported parameter values are marked by an error flag. When such a value is detected, an event is logged if the value is considered optional (for example, if parameter 4 is not supported, HP VPLUS applications can't work with this PAD), or the connection is cleared if the value is considered mandatory (for example, parameter 5 with the value 1, PAD flow control). Some PADs don't follow the 1980 X.29 recommendation and answer with X.29 PAD error messages for the values they don't support. In this case, many set and read messages are sent to identify the unsupported couples.

At the end of this sequence, a list of supported PAD parameter values is computed and stored for later use in data transfer.

3. PAD configuration. An X.29 set of all the parameters is sent to initialize the PAD profile.

Mapping ADCP Requests

The PADSUP task must transform ADCP requests into X.25/X.29 data, and is in charge of user data editing. Only the main path is described here.

To establish a connection to a host MPE XL system from a remote terminal, the user enters the X.25 address of the desired DTC or host. The PAD transmits an X.25 call to the DTC, which decodes the X.25 host address and translates it into a host name using configuration data. With this host name, the DTC can send a request onto the LAN using the HP Probe protocol to retrieve the LAN address of the MPE XL host system. With the LAN address, the connection can be initialized to the host.

Upon a host request to establish a connection to a PAD port, the DTC receives a string that identifies the device. Associated with this string is an X.25 address obtained from configuration data. With this X.25 address, an X.25 call packet can be transmitted to the PAD.

Write data is transmitted to the PAD as X.25 data. For a write and read operation, the PADSUP task sends data to the device and expects data from the device. When data comes from the device, the PADSUP task is in charge of data editing (suppression of backspace, detection of subsys-

tem break). The read data is sent to the host as soon as a read termination condition is encountered by the PADSUP task (end of record detected, byte count reached, time-out, etc.).

Control requests correspond to setting driver parameters. They are mapped into X.3 couples by the PADSUP task and are sent to the PAD. The list of supported parameter values, built during the PAD test, is used to ensure that the PAD is sent only the couples it is able to use. For example, when an HP VPLUS block mode application is used, the PADSUP task receives a write port configuration command with a flag indicating that the HP VPLUS block mode is enabled. The task then configures the PAD with the X.3 parameter values corresponding to no escape from data transfer (1:0), no echo (2:0), data forwarding condition (3:0,4:10), break enabled (7:21), XON/XOFF flow control enabled (5:1,12:1), no editing (15:0). When the user presses the **Enter** key, all of the data on the current screen is sent by the terminal to the PAD, which sends it on to the DTC PAD support task because of the data forwarding condition. All full packets are sent to the DTC and the last packet is sent using the idle timer value specified by parameter reference 4 set to the value 10.

Another example is the transmission of a break to the application. If the application authorizes the user to press the **Break** key, the PADSUP task receives a control request consisting of a write port configuration message with a break enabled flag. The task sends an X.29 set to the PAD to enable break processing (7:21). When the user presses the **Break** key to interrupt processing, the PAD sends an X.25 interrupt packet to the DTC PADSUP task along with an X.29 indication of break packet. The PAD will discard all data received from the DTC. On the X.25 interrupt, the PADSUP task sends an asynchronous break detected event to the DIODAM task, which transmits it to the host. On the X.29 indication of break, the PADSUP task sends an X.29 set to the PAD to reset the PAD discard output state (set 8:0). This message flushes all data buffered in the X.25 network and puts the PAD in a normal delivery state. New data coming from the application can again be displayed on the terminal.

In the case of a control request to clear a connection, the PADSUP task receives in AFCP abort connection message from the DIODAM task to shut down the X.25 connection. Special care must be taken when it comes to closing a connection. X.25 clear packets are not flow controlled as X.25 data packets are, so it is possible that the clear packet may overtake the last data packet sent and therefore the last data for the PAD device may be lost. For this reason, X.29 specifies a safe way to close a connection. The PADSUP task does not send the X.25 clear packet, but instead sends an X.29 invitation to clear message, which is flow controlled like normal data. When the PAD receives this message, it will already have received all the X.25 data, and will then issue an X.25 clear message. On receipt of the X.25 clear message, the PADSUP task can clear its internal table.

When the connection is cleared by the user PAD (with the X.28 CLR command, for example), the PADSUP task sends an ADCP asynchronous link level disconnected message to the DIODAM task, which transmits it to the host, closing the AFCP connection.

Editing

One of the functions performed at the PADSUP level is data editing. This consists of processing the byte stream received from the X.25 network and detecting any special characters for which actions are to be performed. These special characters include backspace, line delete, subsystem break, and others. They are configurable ASCII values that the application sets using file system intrinsics such as FCONTROL. The PADSUP task has to be able to process these characters according to their configured ASCII values. Usually, part of the processing is done by the PADs, most of which are able to process backspace and line delete characters. However, because ADCP provides more editing features than X.3, and because all PADs don't support the editing functionality, the PADSUP task has to be capable of doing this processing itself.

Editing is on the critical code path. It is invoked for any inbound X.25 packet and requires character-by-character processing. The more straightforward algorithm is to use a cascade of IF statements. Because there are eleven special characters to test for, this makes eleven tests for each character. The most frequent case, a normal character, is the worst case because all eleven tests must be performed before it is known that there is nothing to do. On the other hand, this algorithm consumes very little memory because it requires storing only the eleven special values. Each connection can have a different set of values configured, so the total number of values that must be stored is 11×256, where 256 is the maximum number of connections.

The algorithm we have implemented is more memory-consuming but has much better performance. The main principle is to use a structure that we call a character filter. It is an array of 256 entries for each connection (256×256 total entries), indexed by the ASCII values. Each entry represents a particular action to perform. For example, if the backspace character is represented by the ASCII value 08,

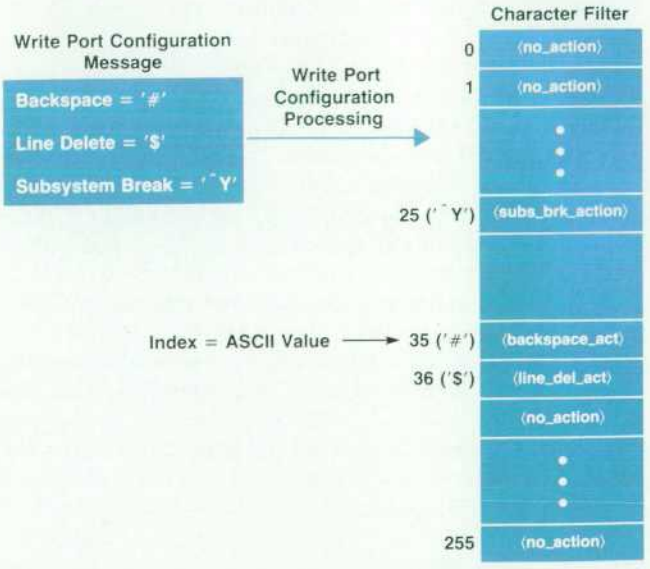


Fig. 7. *A write port configuration message results in the building of the character filter.*

at the index 08 the PADSUP task records the code for the backspace action. The most frequent action is no action for the normal characters and is coded with 0. The array is initialized to zero and the proper actions are recorded at the ASCII values of the special characters when a special ADCP control message called a write port configuration message is received from the host (see Fig. 7).

When the data arrives from the X.25 network, the editing algorithm is executed (see Fig. 8). For each character, the action to be performed is fetched from the character filter and executed. In the case of a normal character, nothing is done. For the others, the appropriate processing is triggered.

To avoid a test of the length of the buffer on each character, a special character called the end of data marker is written at the end of the buffer. A special action is recorded in the character filter for this character. When this character is encountered, this action is triggered, and only in this case is the test of the length of the buffer made. If the end of the buffer has not been reached, it means that the end of data marker was part of the user data and should be treated as a normal character, so no action is taken.

PADSUP Development Methodology

In developing the PAD support component, the first step was to define the external specifications of the product. All MPE XL intrinsics and the X.3 and X.29 recommendations were analyzed to define what could be supported and what could not be supported because of X.3 recommendation restrictions. The configuration and the support features were studied with the participation of HP marketing people.

The second step was to define the best architecture to provide the desired functionality. We analyzed the data flow from the host to the DTC and the PAD, and then defined the component responsibilities for the host modules and the DTC modules.

The third step was to analyze the exact characteristics of the PADSUP task within the DTC. For this, we used the structured analysis method[3,4] with the HP Teamwork SA/RT tool to define what needed to be done without regard to how it would be done. We then drew a functional model (what the system does), a data model (how data is transformed), and a behavior model (state transition diagram and matrix, and decision tables that characterize the system). This method was of considerable benefit for analyzing and detailing all the mechanisms needed in the PADSUP task. The graphic representation and the top-down method helped us better locate where the complexity and the main incidence of all the features occurred on the model. A review was done to verify the PADSUP analysis. This analysis gave us some metrics (Bang,[5] estimated number of decisions or branches, etc.) to characterize the module's complexity, helped us in schedule estimation, and gave a good idea of the tests needed during the qualification phase. During this phase, the project model and the preliminary test plan were completed.

PADSUP Task Design

During the PADSUP task design phase, we defined what modules and what interfaces were needed to meet the specifications. We organized, ranked, and ordered the characteristics to meet the requirements. We chose to follow the structured analysis with a structured design, providing three kinds of documents: structure chart, module specifications, and design dictionary. The structure chart shows the basic components of the solution and shows the interfaces in a top-down manner. The module specifications define the procedural aspects of the solution, or the sequence of interactions. The design dictionary defines the interfaces.

A first review of the main module organization was done to verify the encapsulation, design cohesion, and coupling of the different modules. The main modules were then designed independently.

A complete review of each module was done to find the design flaws. This was highly successful because we found all the complex areas before coding and were able to redesign the parts that were too complex. As a result, very few design bugs were found during testing.

In the coding phase, the design documents helped us to be productive rapidly. The code was done in C on HP-UX workstations.

PADSUP Task Testing

The PAD support task has been tested in a multiple-step process. The target code (to be downloaded to the hardware later) was produced with the HP-UX C compiler on an HP 9000 Series 300 workstation and then postprocessed. This was possible because the microprocessor on the SNP card is a 68010, which is of the same family as the workstation microprocessor. This made it possible to test the code in a comfortable environment (diskless workstation) and then port the code to the hardware with a high level of confidence, since the target code was almost the same. One of the main advantages of doing this was the availablity on the workstation of many C tools that we could use on the target code (C debugger, branch flow analyzer, prof for per-

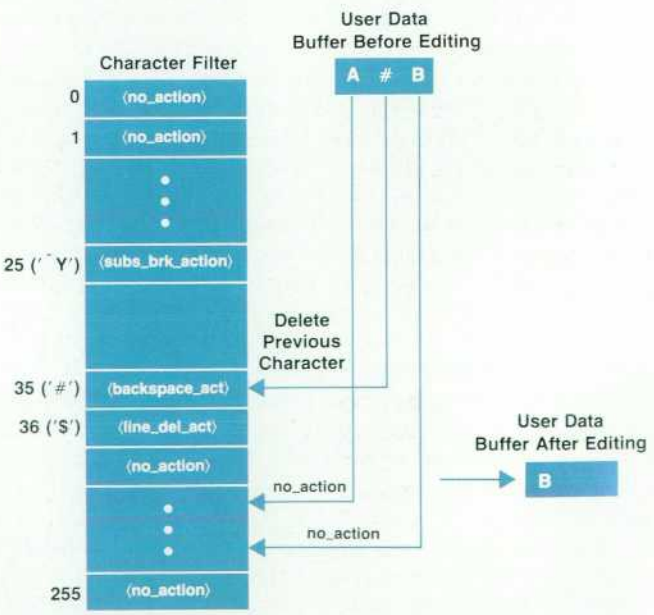


Fig. 8. *Editing with the character filter.*

formance analysis, etc.).

The multistep testing process consisted of unit testing, integration testing, and real-environment testing. Related issues include testing in the maintenance phase and testing of multivendor PADs.

Unit Testing. The objective of this step was to bring each individual component (procedure or function) to a good quality level for later integration. The designer of each procedure or function was responsible for developing code stubs to exercise the component. This was done at the unit level, that is, the component was tested by itself and not in a complete environment. This level of testing exposed most of the coding errors.

Functional Testing. At this test level, all the individual modules of the PADSUP task were merged together and exercised in a functional way, that is, we were testing whether the module was indeed performing the functions specified. This was mainly achieved with the message machine described in the article on page 74, which is capable of building messages, sending them to the module under test, receiving the output, and decompiling it in a readable format. In regression mode, output files were compared to reference files that had been manually checked during previous testing. Since both the output file and the reference file were in readable format, the differences were easy to see. All of these tests were run on the HP-UX workstation.

The message machine tool was really convenient to use. With it, we easily achieved the branch flow analysis (BFA) coverage required. The objective of the project was to test 85% of all the software branches. We have achieved 93% without the need of any additional operations such as setting breakpoints or adding code. Moreover, because the tool provides for easy automation, all of the test suite has been maintained and now we can run a regression test overnight without any external intervention and reach a BFA coverage of 93% on each version or patch released. The number of test cases is quite large. Up to 50,000 different messages are sent to the module under test, and the overall size of all the output files that are checked for differences with reference files is 36 megabytes.

Integration Testing. These tests were conducted at two different levels: in the simulated environment provided by the message machine under the HP-UX operating system and in the target environment (hardware).

The objective of the HP-UX integration testing was to add step-by-step all of the tasks external to the PADSUP task in the simulated environment. In the first step, all of the other tasks were emulated by the message machine. These were then replaced, one by one, with the actual tasks, and the system was exercised by the message machine. Finally, all of the interfaces with other modules were tested and debugged with the actual tasks still on the workstation.

For hardware integration, the code was downloaded to the hardware and run in its final environment. Because of the multiple tests already conducted, the quality at this stage was already rather high and we found very few defects during this testing. The principal tool we used was a debug port that allowed us to connect a terminal to the board and set breakpoints, dump memory areas, and so on.

Real-Environment Testing. These tests were done mainly by the QA department to test the quality objectives of the product in the areas of functionality, localizability, usability, reliability, performance, and serviceability. The setup used was close to a real customer environment. For ease of testing, two different environments have been used: a real PAD environment and the XXPAD environment.

The real PAD environment is a customer-type environment. It consists of an MPE XL system, a DTC with SNP cards, an X.25 network or switch, PADs, terminals, and printers. The tests consist mainly of manual validation (for DTCMGR functions, for example) and of test programs run on the MPE XL system that exercise the datacom functions through the system intrinsics. The problem with this setup is that the number of sessions is limited by the hardware. The DTC supports 256 PAD connections per synchronous card and three synchronous cards per DTC, so a full test requires so much hardware that it is impractical. To approx-

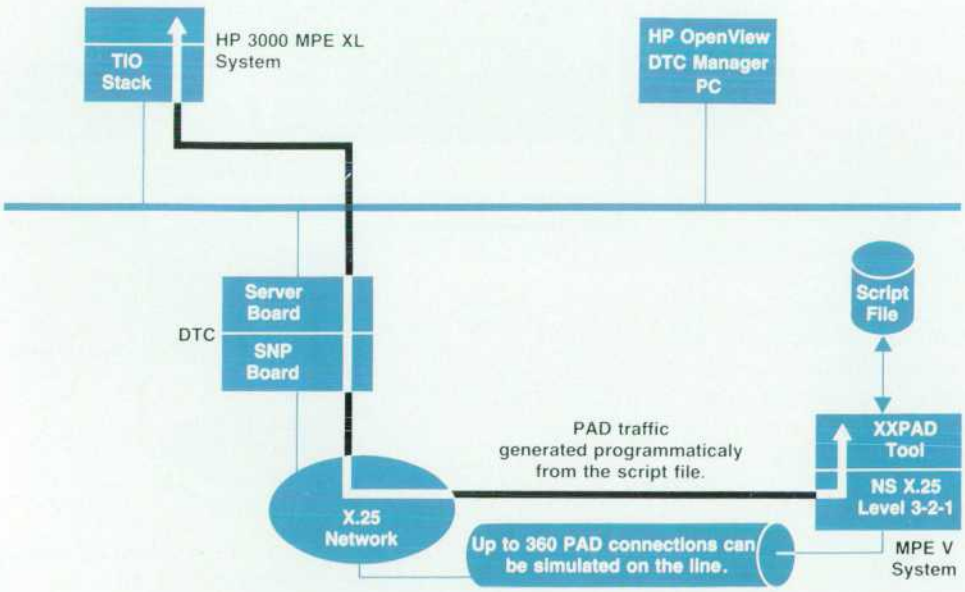


Fig. 9. *The XXPAD tool was developed to test the PAD support software in the DTC without excessive hardware. The tool can simulate multiple connections. The first versions ran on an MPE V system.*

imate a full test (i.e., limit, stress, and volume testing), we use the XXPAD environment.

The XXPAD environment makes it possible to minimize hardware setup. The PAD is basically a multiplexer of many asynchronous lines (16 in the case of the HP 2335A PAD) to a single X.25 line. Therefore, instead of managing 16 or more terminals, we can just put on the X.25 line a piece of hardware and software that emulates numerous asynchronous devices. This way, the number of connections tested can be increased simply by increasing the number of virtual circuits handled by the test machine. For ease of implementation and for hardware availability, this tool was developed on an MPE machine running HP NS X.25 software, which provides programmatic access to X.25 level III. On top of the level III access, we developed a test program called XXPAD, which is able to handle as many virtual circuits as required. This program reads the instructions to execute (basically user commands) from a script file and handles both X.25 and X.29 data.

For the PADSUP tests, the scripts simulated multiple users connected to the DTC and to the MPE XL system and running some basic applications such as the command interpreter and editor. Some test programs were also developed on the MPE XL system to stress special data transfer cases, such as large writes, large reads, and stress conditions of control requests. These ran in conjunction with the XXPAD tool on the PAD side.

In its first versions, the XXPAD tool ran on an MPE V system (Fig. 9). When the XL version of the HP NS X.25 software became available, it was possible to migrate to the MPE XL system using a loopback configuration. The XXPAD tool accessed the DTC through the NS stack (system-to-system) and then looped back on the network to enter another SNP card on which the connection was routed to the PADSUP module as formatted by the XXPAD tool (Fig. 10). The XXPAD tool eliminated a great deal of hardware and test setup for the tests implying multiple connections.

To determine when to end the quality tests, each quality engineer logged the time taken for each test and the number of bugs encountered. This data was input to a software tool along with a measure of the quality level we wanted to reach. Using the Musa model,[6] the tool gave us a weekly forecast of the end of the tests. Two months before the scheduled end of the tests, the weekly forecasts became stable, indicating the same date for the end of the tests.

Maintenance Testing. In the maintenance phase of the PADSUP task, the functional test suite continues to be maintained with the message machine. Each time a new version of the PADSUP code is built, it is validated against the test suite. It requires some extra work to maintain the test suite. The reference files must be kept up to date, and because the test coverage is high, any small change in the code causes a modification in the result of the tests (i.e., the difference between the reference files and the output files) that has to be manually validated. But the test suite ensures a high level of quality and confidence when a new version or patch has to be delivered on a tight schedule. We are also working with the real hardware environment, of course, and with the XXPAD environment because it allows automatic regression and reliability tests in a realistic environment.

Conclusion

The PAD support in the DTC architecture offers remote access to HP 3000 MPE XL systems in a PAD multivendor environment by masking all vendor-specific or nonstandard implementations without adding code or configuration complexity. It improves both performance and reliability by using the existing terminal I/O path on the MPE XL system. Offloading of all data editing from the host to the DTC increases throughput and connectivity by allowing a single network access for several systems. Modularity of both design and code has been preserved to make the PAD support task reusable in future products.

Formal methodology has been used throughout the development. The use of structured analysis, structured design, and formal testing at different levels, all with the permanent collection of metrics, has been fundamental to the success of the project. While such techniques cost us time in the first phases of the development, they saved a lot in the integration and test phases and left the product in good shape with regard to maintenance and expandability.

Acknowledgments

The conception and the implementation of the PAD support task required the effort of several teams. We'd like to thank all the people involved at both the Information Networks Division and the Grenoble Networks Division. Francois Gaullier and Christian Gresset, the PADSUP project managers, provided indispensable management and technical support. We would like to acknowledge the effort and perseverance of Sylvaine Roy who did the analysis, part of the design, the test plan, and part of the tests with Amar Kebaili. Other key contributors in the project were Christian Lotito, Remy Poulailleau, Olivier Bordes, and Bernard Wagner for design review, coding, and testing. Frederic Maioli provided a tool to test the PAD support

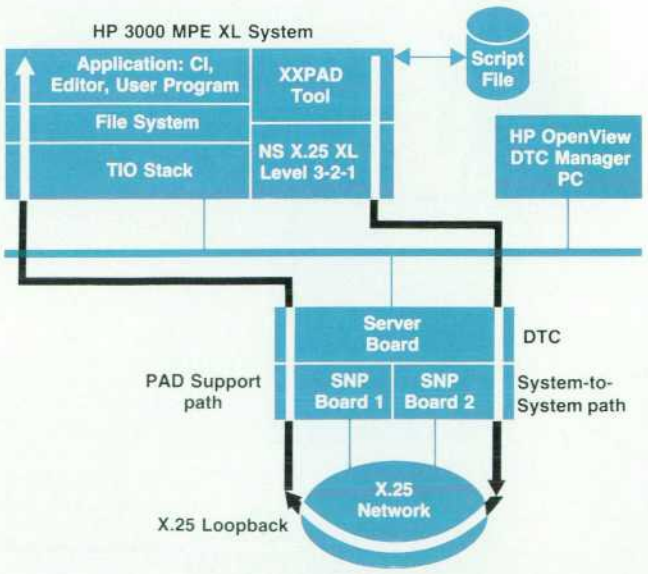


Fig. 10. *Later versions of the XXPAD tool run on the MPE XL system with a loopback connection through the network.*

software under simulation. We would also like to thank Jean-Philippe Caradec who was responsible for the DIODAM task and participated in the design of the general architecture. The XL driver modifications were done by Celeste Ross and Michael Reasoner and the XL tests were conducted by Chris Smith and Larkin Ryder. We appreciate the very good relationship and the team spirit between the two divisions.

References

1. G.F. Buchanan, et al, "A Distributed Terminal Controller for HP Precision Architecture Computers Running the MPE XL Operating System," *Hewlett-Packard Journal*, Vol. 38, no. 3, March 1987, pp. 21-28.
2. S.Y. Amar and M.A. Prieur, "Network Management for the HP 3000 Datacom and Terminal Controller," *Hewlett-Packard Journal*, Vol. 41, no. 2, April 1990, pp. 76-84.
3. S.J. Mellor, *Structured Development for Real-Time Systems*, Yourdon Press, 1985.
4. M. Page-Jones, *The Practical Guide to Structured Systems Design*, Yourdon Press, 1980.
5. T. DeMarco, *Controlling Software Projects*, Yourdon Press, 1982, p.90.
6. J. Musa, "A Theory of Software Reliability and Its Application," Vol. SE-1, no. 3, September 1975.

An Object-Oriented Message Interface for Testing the HP 3000 Data Communications and Terminal Controller

Creating a general-purpose message compiler/decompiler using symbolic expressions, expert systems concepts, object classes, and inheritance reduces software testing overhead and improves test readability and portability.

by Frédéric Maioli

THE HP 2345A DATA COMMUNICATIONS and terminal controller (DTC), which operates with the MPE XL 2.0 operating system, provides access to X.25 networks, PAD (packet assembler/disassembler) support, asynchronous terminals and printers connected locally or through modem links, back-to-back connections, and other capabilities. All of these services can be shared by multiple MPE XL systems connected through a local area network (LAN). With the back-to-back feature, a DTC can work without a host.

The HP 2345A DTC software is based on a multiprocessor, multitasking operating system. In this system, parallel processes or *tasks* communicate among themselves by exchanging *messages*.

An important step in developing a task is testing it. This is done by building a *message machine*, a program that simulates the message environment of the task under test. During the development of the DTC network software, several message machines were written in the MPE XL operating system and on HP 9000 Series 200 Pascal workstations.

Reusing an existing message machine can seem very attractive to someone starting a new project. However, even when this can be done effectively, each version will usually be maintained separately, and the benefits of having some common code are gradually lost.

Looking for a better solution, we have used object-oriented concepts to write a message machine for testing the HP 2345A DTC PAD support software (see article, page 63). The program is declarative rather than procedural. This makes it smaller in code size, more reusable, and very easy to adapt to new message structures. The main drawback is a decrease in performance, which can be overcome by running the tests on a more powerful hardware platform.

Task Interaction

As already mentioned, a *task* is a process inside the DTC. A *byte-code message* is a sequence of bytes that a task sends to another task. The syntax of the different messages a task can send or receive constitutes its *message interface*.

Each task maintains its own behavior as it interacts with its environment by sending and receiving messages. Testing a task consists in checking its behavior by sending it a sequence of messages or a *message script*. The tested task reacts to the message script by sending back a sequence of messages that constitutes a *message trace*.

A message script file is written in a *message source language* and is interpreted by the message machine, which sends the corresponding byte-code messages (see Fig. 1). The advantage of having a message source language and an interpreter is the readability of the message scripts. This also makes it possible to send a message from an interactive interface, run a message script in step-by-step mode, or put the tested task in a determined state (to debug it, for example).

The message machine catches any message sent by the tested task in its byte-code format, translates it to source format, and logs it in the message trace file. Translating from source to byte code is called *message compilation*, and the reverse operation is called *message decompilation*.

The complete *task test* is a set of message scripts that test as much as possible of the task's behavior.

In the process of maintaining the DTC software, new versions of a task may be released. Compatibility of the old

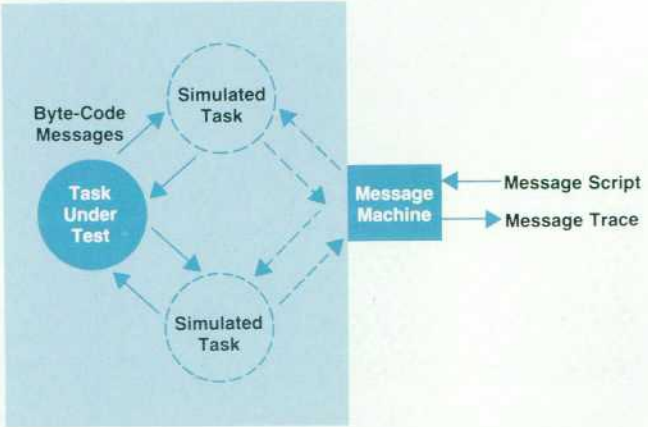


Fig. 1. *To test a task, the user creates a message script. The message machine translates this script to byte-code messages that simulate the environment of the task under test. The tested task's responses are returned to the user as a message trace.*

and new versions is checked by running the same task test on both versions and comparing the respective message traces. This operation is called a *regression test* because the comparison is expected to validate the new version's compatibility.

The usual way to run a regression test is to run a comparison program such as the HP-UX diff utility, which searches for any differences between two message trace files.

An Example

To illustrate these concepts, let us consider a protocol for communication between a task named device and a task named server. These tasks have been chosen for purposes of this example, but one can imagine that the device task manages a set of RS-232 terminal links, while server handles a computer system's I/O.

Fig. 2 shows an example of a message exchange between the two tasks server and device. Five different messages can be exchanged. The open_session message signifies that a device has become active and requests that a session be opened (logon). The close_session message is sent when a device logs off. The write message sends data to a device. The read_line message occurs when a device enters a line of text. The read_break message signifies that a break interrupt key signal has been received from a device.

The byte-code syntax of these messages is described in Fig. 3. The information inside a byte-code message is structured in fields. The message_code field is a 2-byte integer containing a constant related to the type of message. The dest_task field is one byte long and is used by the operating system to identify the task the message is going to. In this example the ASCII value is 25 for the server task and 32 for the device task.

The data field is a string of bytes representing the content of a write or read message. The data_length field is a 2-byte integer field representing the length of the data string in bytes.

The break_code field is a 1-byte field that distinguishes the read_line message (value = 0) from the read_break message (value = 1). These messages have the same message code. The dev_num field is a 2-byte integer that identifies the hardware device managed by the device task.

As mentioned above, our message machines define a source language associated with the byte-code message syntax. Examples of source messages are:

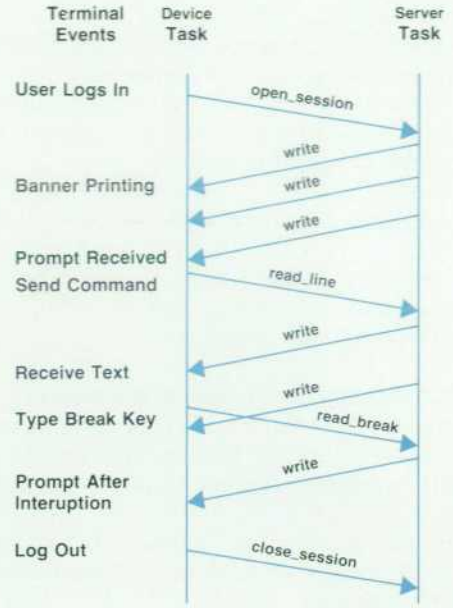


Fig. 2. *Messages exchanged between the two tasks* server *and* device.

```
open_session     dest_task: 25 dev_num; 30
read_line        dest_task: 25 dev_num: 30  data: 'listf,2'
read_break       dest_task: 25 dev_num: 30
close_session    dest_task: 25 dev_num: 30
```

In this example, each line represents a message to server, which is identified by the dest_task value 25. The first symbol in a line identifies the message. It is followed by a sequence of identifier/value pairs. The identifier is a parameter name (terminated by :) corresponding to a field in the byte-code syntax, and the value is the value of this parameter in the message.

The byte-code fields corresponding to parameters in source messages are called *parameter fields*. Examples are dest_task, dev_num, and data. Not all the fields of a byte-code message are parameters in the source syntax. There are also *constant fields*, or *selector fields*, whose value identifies a unique *message structure*. Examples are message_code and break_code. Finally, there are fields, such as data_length, whose values depend on the values of other fields. These

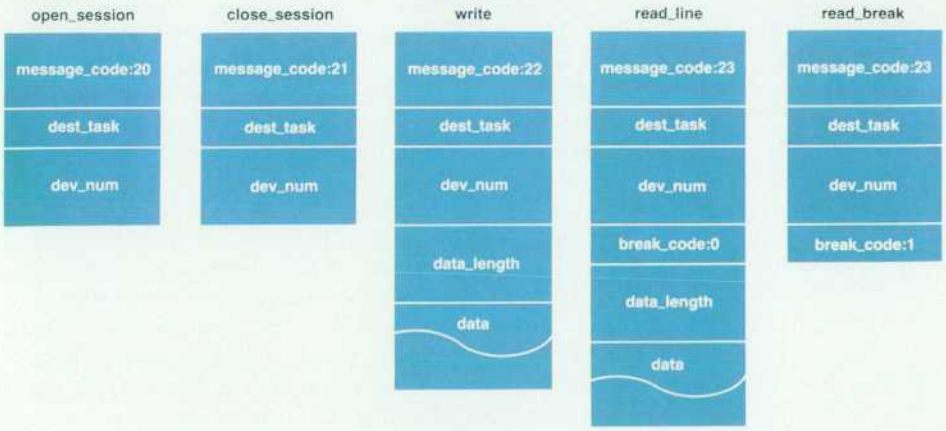


Fig. 3. *Byte-code syntax of the messages in Fig. 2.*

are called *computed fields.*

Notice that the list of source messages given above could be used as a message script by a message machine and would correspond to a test of server. This reproduction of the dialog of the example of Fig. 2 would make the message machine produce the message trace:

```
write dest_task: 32 dev_num: 30 data: 'XYZ MPE XL A.30.00. TUE,
                                      MAY 29, 1990'
write dest_task: 32 dev_num: 30 data: ':'
write dest_task: 32 dev_num: 30 data: 'FILE1 FILE2 FIL'
write dest_task: 32 dev_num: 30 data: ':'
```

Conventional Compilation and Decompilation

When one wants to reuse a message machine and adapt it to new message formats, most of the changes consist of rewriting the message compilation and decompilation procedures.

There is a procedure compile_message, which takes as input a line of message source code and returns the byte-code format of this message:

```
procedure compile_message(line)
    local variables: idf, result
    read an identifier idf in line;
    if idf = 'open_session' {
        let result be a new string
        write the integer 20 on 2 bytes in result; /* message_code */
        compile_parameter_in_type_size(line,'dest_task',result,
                                       'integer',1);
        compile_parameter_in_ type_size(line,'dev_num',result,
                                       'integer',2);
        return result; }
    else if idf = 'close_session'
            ...
    else if idf = 'write'
            ...
    ...
    else error('bad message name')
    end if
    return result
end procedure
```

In this procedure, the message type is identified. Then the byte-code message is synthesized parameter by parameter in the following procedure:

```
procedure compile_parameter_in_type_size(line,param_name,
                                         result,type,size)
    local variables: idf
    read an identifier idf in line;
    if idf and param_name are not the same then error("bad
                                         parameter name");
    if type = 'integer' then
      read the integer int in decimal format in line;
      write the integer int in binary on size bytes in result;
    else if type = 'string' then ....

    end if
end procedure
```

Decompiling a message is slightly more difficult. First, one must match a message structure to a byte-code format. This is done by a decision tree that tests values in fixed fields:

```
procedure decompile_message(byte_message)
    local variables : message_code, break_code
    let message_code be the integer from byte 1 to byte 2 in
                byte_message;
    if message_code = 20
       return decompile_open_session(byte_message);
    else if message_code = 21
       ...
    else if message_code = 23
       let break_code be the integer at byte 6 in byte_message;
       if break_code = 0 then
          return decompile_read_line(byte_message);
       else if break_code = 1 then
          ...
       end if
    else error("bad message code");
    end if
end procedure
```

There is a procedure for each message structure, which is called when the message structure has been determined. For example, the one corresponding to the open_session message is:

```
procedure decompile_open_session(byte_message);
    local variables: result
    let result be a new string;
    write 'request_session' in result;
    decompile_dest_task_in(byte_message,result);
    decompile_dev_num( ... );
    return result;
end procedure
```

For each parameter, there is the reciprocal of the parameter compilation procedure. One example is:

```
procedure decompile_dest_task_in(byte_message,result)
    let dest_task be the integer at byte 3 in byte_message;
    write 'dest_task' and dest_task in result
end procedure
```

Writing the message compilers and decompilers in algorithmic language is simple for a few messages, but it is more difficult for 50 messages, a typical number for the message interface of a real DTC task. The message compilers and decompilers are programs that are very dependent on message byte-code and source syntax, so they contain much redundant information.

This approach has several drawbacks. First, the byte-code-to-message matcher is complex, difficult to validate, and difficult to maintain when new message structures are added. Second, some fields are common to several messages. It would be possible to write some partial compilation and decompilation procedures common to several messages, but modifying the syntax of one message could require modifications in other messages. Third, when

changing or adding a field, there are interdependent modifications in the compiler and in the decompiler. This creates the need for a compatibility test between compilation and decompilation procedures.

A further drawback of the conventional approach is that, for the same script and the same tested task, some parameters can have different values in message traces that do not constitute regression test violations. For example, there can be a message trace that contains some information about the current date:

```
write dest_task: 32 dev_num: 30 data: 'XYZ MPE XL A.30.00. TUE,
                                       MAY 29, 1990'
```

This causes useless differences between message traces that increase the difficulty of analyzing these files and of automating the tests.

A final drawback of the conventional approach is that no message documentation is maintained as the message machine evolves. The documentation used in reality is the source listing of the compilation and decompilation procedures.

Using Symbolic Programming Concepts

Object-oriented languages provide symbolic expression facilities. This gave us the idea of representing the message source language using symbolic expressions. This is the same as using lists in Lisp,[1] arrays in Smalltalk,[2] or terms in Prolog[3].

The traditional source message:

```
open_session dest_task: 25 dev_num: 30
```

is equivalent to the symbolic expression (using Lisp lists, for instance):

```
(open_session (dest_task 25) (dev_num 30))
```

More generally, any source message will have the syntax:†

```
(<message name> (<parameter name> <parameter value>)*)
```

There are significant advantages to the use of symbolic expressions. First, it is easy and improves readability to give name values instead of integer values to some parameters. For example, (dest_task 'device') is better than (dest_task 32).

Second, intelligent regression is possible. When comparing message trace files, the HP-UX diff utility provides a byte-to-byte comparison. It is a better solution to compare two symbolic expressions by traversing their structures (like the equal function in Lisp[1]). This is because:

- It is more efficient to compare symbols than the bytes of strings.
- One can customize the comparison algorithm by not comparing some parameters that contain variable information, such as the current date.
- The formatting of regression violations is easily improved. The incorrect messages are displayed rather than the incorrect lines.
- It is easier to recover automatically after a nonregression violation and continue the comparison, since the comparer synchronizes on messages rather than lines.

A third advantage of using symbolic expressions is automatic indentation. Some environments provide indenting formatters for symbolic expressions (such as Lisp pretty printing[1]). Using it, large source messages with many parameters are very readable:

```
(command_name (param1 val1)
              (param2 val2)
              (param3 val3)
                  ...
              )
```

We used Smalltalk arrays and found it simple to implement such a printing formatter.

Specifying the Compiler and Decompiler

The most important drawback in adding or modifying a new message format in the message compiler and decompiler is that the same information for message translation is coded at least twice, once in the compiler and once in the decompiler. This is a source of errors, code overhead, unreadability, and nonportability of the message machine. The question arises: Wouldn't it be possible to define only once the necessary information for both compilation and decompilation? Many symbolic programs define a data formalism and an interpreter over this formalism. Most of the behavior of these programs is embedded in specification data bases rather than procedures. This is the case for most expert systems, which contain an interpreter (the inference engine) over a specification of the expertise (the rule base, which is a data base). The question is then: Does there exist a formalism to specify the mapping between the source format and the byte-code format of a message, with general-purpose compiling and decompiling procedures?

We have been able to answer this question in the affirmative. We have obtained an efficient specification formalism, which now covers 95% of our message compilation and decompilation. The remainder is implemented conventionally.

Our formalism makes the distinction between two kinds of fields in messages. There are fields with a *statically known position* in the byte-code syntax, and fields with a *statically unknown position*. "Statically" means that the field is always located at the same place in the byte code. This location does not depend on the message content. For example, the dest_task field in all messages has a statically known position (at byte 3) in all byte-code messages, but the data field does not have a statically known position, since the end position of the field is known only when the message to be compiled or decompiled is known.

To simplify, we will call fields with a statically known position *positioned* fields and fields with a statically unknown position *unpositioned* fields. The specification formalism only covers positioned fields. Given a message's specification, we have written the general-purpose compi-

†In all syntax description in this paper, syntax definition is denoted by ::=, syntax repetition is denoted by *, syntax alternatives are separated by |, and syntax concatenation is denoted by juxtaposition.

lation and decompilation procedure that interprets the specification.

The following symbolic expressions describe the specification of the messages of the example above:

- Declaration of all the parameter fields of all the messages and their types (integer, string, or symbolic integer which translates a string to an integer):

```
(parameter_fields
  (devnum Integer)
  (dest_task (SymbolicInteger ('device' 32)
                              ('server' 25)))
  (data String)
  )
```

- Declaration of constant and computed fields:

```
(constant_fields message_code break_code)
(computed_fields data_length)
```

- Declaration of the size (in bytes) of each field in the byte-code messages (positioned fields only):

```
(fields_size
 (message_code 2)
 (dest_task 1)
 (dev_num 2)
 (data_length 2)
 (break_code 1))
```

- Specification of all the messages. For each message, there is an expression for called a *metamessage*. A metamessage formalizes the notion of message structure. First, there is the corresponding message name, then three subexpressions called *metafields*. The parameter_fields metafield contains all parameter field names. The constant_fields metafield maps values to constant fields. The byteCode_order metafield defines how the fields are ordered in byte code.

```
(message_specs
   (open_session
     (parameter_fields dev_num dest_task)
     (constant_fields (message_code 20))
     (byteCode_order message_code dest_task dev_num))
   (close_session ...)
   (write ...)

   (read_line
    (parameter_fields dev_num dest_task data)
    (constant_fields (message_code 23)
                     (break_code 0))
    (byteCode_order message_code dest_task
                    dev_num break_code))
   (read_break
    (parameter_fields dev_num dest_task)
    (constant_fields (message_code 23)
                     (break_code 0))
    (byteCode_order message_code dest_task
                    dev_num break_code data_length))
  )
```

The message specification is now completed. Before using it to perform compilation and decompilation, there is an initialization step called the *metacompilation* phase where the specification is checked for correctness. Examples of checked constraints are field name consistency and collisions, and field juxtaposition consistency in each message.

The metacompilation also generates intermediate results. A field_position metafield is added to each metamessage to complete byteCode_order information. It relists the byte-code fields, adding to each of them the pair <first byte, last byte>. This pair defines the position of the field in the byte-code string:

```
(message_specs
    ...
    (read_break
    ...
     (byteCode_position (message_code 0 1)
                        (dest_task 2 2)
                        (dev_num 3 4)
                        (break_code 5 5)))
  )
```

A constant field always has the same position in all the messages that contain it. Otherwise, it would be impossible to map a byte-code onto a message structure. The positions of all constant fields are kept in the constant_field_position list:

```
(constant_field_position
  (message_code 0 1)
  (break_code 5 5))
```

The decoder tree is used by the decompiler to identify a message from a byte code. In the example, it is:

```
(decoder_tree
  (message_code
   (20 request_session)
   (21 close_session)
   (22 write)
   (23 (break_code (0 read_line) (1 read_break))))
)
```

The syntax is:

```
<decoder_tree> ::=
   <message name>;
   (<constant field name> (<constant field value><decoder_tree>)*)
```

It is used in the following function, which is used by the decompiler:

```
function decode_bytes(t,str)
   /* find the message structure name associated with the byte code
     str by performing the tests specified in the decoder tree t */

   local variables: x,y,val,cName,n,v1...vn,t1...tn,i

   if t is a name, return t;
   t has the form (cName (v1 t1) ... (vn tn))
         otherwise error("bad decoder tree");
   let x and y be such that the 'constant_ field_position' list contains:
   (constant_field_position ... (cName x y) ...)

   let val = the integer coded from byte x to byte y in str

   let i , 1 <= i <= n such that vi = val
        otherwise error("undecodable byte code");
   return decode_bytes(ti,str);
end function
```

The decoder tree takes the place of the decision tree in the conventional decompilation procedure. It is automatically generated at the metacompilation step:

```
procedure generate_decoder_tree(lst)

/* return a decoder tree mapping one of the message names in the list
lst from a byte code string */

   local variables: n,v1..vn,i,mi ,c,S,vk,sublist,subdecoder,result

   suppose lst has the form (m1 ... mn)

   if n = 1 return m1;
   endif
   /* find a constant field that discriminates messages */

   find a constant field c such that
     for all i, 1 <= i <= n consider the metamessage:
        (mi ... (constant_fields ... (c vi) ...) ...)
        and let S be the set of all vi
        and S contains at least two elements
   if not found error("impossible to build a decoder tree")
   end find

   let result = an empty list

   /* recursively build the tree */
   for all vk in S
      sublist = the list of mi such that vi = vk
      subdecoder = generate_decoder_tree(sublist)
      put subdecoder to the end of result
   end for
   add c to the beginning of result
   return result
end procedure
```

The general-purpose compiler looks as follows (for simplicity, type checking and parameter value translation have not been indicated here):

```
function compile(lst)
   /* translate the source message lst in byte code */

   local variables: m,n,p1..pn,v1..vn ,m,c1..cm,w1 ..wm,
                    p,b1..bp,x1..xp,y1..yp,i,j,result
   suppose lst has the form: (m (p1 v1) ... (pn vn))

   find the metamessage of the form:
     (m (parameter_fields p1 ... pn)
        (constant_fields (c1 w1) ... (cm wm))
        (byteCode_position (b1 x1 y1) ... (bp xp yp))
   otherwise error("syntax error")

   /* initialize byte code */
   let result be an empty string

   /* explained later */
   call the procedure attached to m to compute computed field values

   for 1 <= i <= p
     if j exists such that bi = cj then
        /* code constant field */
        code wj as integer from byte xi to byte yi in result
     else if j exists such that bi = pj
         /* code parameter field */
         code vj as integer from byte xi to byte yi in result
     else if ... /* and so on for computed parameters */
     end if
   end for
   /* explained later */
   call the procedure attached to m to compile unpositioned
                              parameters to result

end function
```

The decompiler is as follows:

```
function decompile(str)
   /* translate the byte code string str in source message */

   local variables: t,m,n,p1..pn,p,b1..bp,x1..xp,y1..yp,result,
                    i,j,val

   let t = the decoder tree specification;
   let m = decode_bytes(t,str);

   find the message specification in message_specs:
     (m (parameter_fields p1 ... pn)
        (byteCode_position (b1 x1 y1) ... (bp xp yp))
     )
   /* explained later */
   call the procedure attached to m to decompile unpositioned
     parameters and computed fields from str

   let result be the list made up of the single element m
```

```
for 1 <= i <= n
    if j exists such that b_j = p_i
      /* positioned parameter */
      val = the integer coded from bytes x_j to y_j in str
    else
      /* explained later */
      else val = value computed by the procedure attached to m
    endif
    add (p_i val) at the end of result
  end for
  return result
end function
```

The different statements commented /* explained later */ refer to conventional procedures necessary to compile and decompile unpositioned fields and compute values of computed fields. These procedures (not described here) represent no more than 5% of the message machine implementation (in noncomment source code lines of Smalltalk).

As a result of this design, to add, modify, or remove message structures it is only necessary to edit the message specification list, adapting a few procedures for unpositioned fields, and running the metacompilation step.

Using Objects and Inheritance

The symbolic message machine was implemented in Smalltalk/V on a personal computer and then ported to a C-coded Smalltalk implementation on the HP- UX operating system to interface with the test environment of the DTC operating system. A Smalltalk interpreter provides the interface to the DTC operating system test environment on HP-UX. The message machine and all tests are implemented in this interpreter. Its expression power and openness are powerful advantages that help develop and run the tests more efficiently than before.

A number of facilities allowed us to write an efficient and well-adapted implementation. One adaptation was the use of array objects to represent symbolic expressions. Another was the use of classes and inheritance to implement objects representing metamessages, metafields, and positions. In particular, different metamessages contain much common information. The immediate solution consists of implementing partial message descriptions, relations between them, and metamessages to propagate metafields and field structure information. Metamessages also refer to procedural information that is to be propagated along these relations. Therefore, it was helpful to map classes and superclasses to metamessages and partial descriptions, and the basic class inheritance mechanism to the propagation relation.

Thus, a class is created for each message and the metafield information is obtained from methods—that is, by inheritance. Procedures related to unpositioned fields are treated in the same way. For example, to obtain metafield information on the write and open_session messages, one creates the classes:

```
DtcMessage
WriteMessage, subclass of DtcMessage
OpenSessionMessage, subclass of DtcMessage.
```

DtcMessage provides information common to all messages. Specifying values of the parameters_field metaparameter consists of implementing the following methods:

```
method parameter_fields() on class of DtcMessage
    return the Array (dest_task dev_num)
end method

method parameter_fields() on class of WriteMessage
    result := result of sending parameter_fields to superclass of self
    return concatenation of result and the symbol data
end method
```

Other adaptations were:

- Metafield information is cached in sets, dictionaries, and ordered collections for better performance.
- The compile and decompile procedures are methods of the abstract message class DtcMessage.
- A new metafield, example, is used in a general-purpose test procedure to build an example source message. It is also used in an automatic documentation procedure.

Conclusion

The object-oriented message machine has been used in the HP 2345A DTC PAD support project (see article, page 63). Validating a new version of the PAD support software only requires starting an HP-UX shell; no further human intervention is needed.

After having being successfully used by the PAD support project, the message machine proved its ability to adapt to a new project, where it is now used.

Acknowledgments

I wish to thank the people who helped develop these new techniques in software testing: Christian Billeau, Montserrat Mane, Jean Pierre Dacher, Marie-Thérèse Sarrasin, Sylvaine Roy, Jean-Pierre Allègre, Bernard Wagner, Olivier Bordes, and Christian Gresset. Special thanks to Remy Poulailleau for a lot of ideas and encouragement.

References

1. P.H. Winston, *Lisp*, Addison-Wesley, 1981.
2. A. Goldberg and D. Robson, *Smalltalk 80: The Language and Its Implementation*, Addison-Wesley, 1983.
3. W.F. Clocksin and C.S. Mellish, *Programming in Prolog*, Springer-Verlag, 1981.

HEWLETT-PACKARD JOURNAL

Volume 41 January 1990 through December 1990

Hewlett-Packard Company, 3200 Hillview Avenue, Palo Alto, California 94304 U.S.A.
Hewlett-Packard Marcom Operations Europe, P.O. Box 529, 1180 AM Amstelveen, The Netherlands
Hewlett-Packard (Canada) Ltd., 6877 Goreway Drive, Mississauga, Ontario L4V 1M8 Canada
Yokogawa-Hewlett-Packard Ltd., Suginami-ku, Tokyo 168 Japan

Index

PART 1: Chronological Index

Protocol Stack, *Eric C. Scoredos, Kimberly K. Scott, and Richard H. Van Gaasbeck*
Interoperability Testing for HP MAP 3.0, *Jeffrey D. Meyer*
The HP MAP 3.0 Software Integration Lifecycle, *Douglas R. Gregory*
The Integrated Personal Development Environment
500-MHz and 300-MHz Programmable Pulse Generators, *Werner Berkel, Gerd Koffmane, Frederick L. Eatock, Patrick Schmid, Heino Höpke and Hans-Jürgen Snackers*
Hybrid Assembly
A 500-MHz Pulse Generator Output Section, *Stefan G. Klein and Hans-Jürgen Wagner*
A 300-MHz, Variable-Transition-Time Pulse Generator Output Section, *Peter Schinzel, Volker Eberle, and Günter Steinbach*

October 1990
An Overview of the HP Interactive Visual Interface, *Roger K. Lau and Mark E. Thompson*
HP IVI Project Management
Quality Function Deployment and HP IVI
The HP IVI Object-Oriented Toolkit, *Mydung Thi Tran and David G. Wathen*
HP IVI Application Program Interface Design, *Pamela W. Munsch, Warren I. Otsuka, and Gary D. Thomsen*
Object-Oriented Design in HP IVI
HP IVIBuild: Interactive User Interface Builder for HP IVI, *Steven P. Witten and Hai-Wen L. Bienz*
Creating an Effective User Interface for HP IVIBuild, *Steven R. Anderson and Jennifer Chaffee*
26.5-to-75-GHz Preselected Mixers Based on Magnetically Tunable Barium Ferrite Filters, *Dean B. Nicholson, Robert J. Matreci, and Michael J. Levernier*
Hexagonal Ferrites for Millimeter-Wave Applications, *Dean B. Nicholson*
HP DIS: A Development Tool for Factory-Floor Device Interfaces, *Kent L. Garliepp, Irene Skupniewicz, John U. Frolich, and Kathleen A. Fulton*
Finite State Machine
Matching Messages
Action Routines
Measurement of R, L, and C Parameters in VLSI Packages, *David W. Quint, Asad Aziz, Ravi Kaw, and Frank J. Perezalonso*
Statistical Circuit Simulation of a Wideband Amplifier: A Case Study in Design for Manufacturability, *Chee K. Chow*
System Level Air Flow Analysis for a Computer System Processing Unit, *Vivek Mansingh and Kent P. Misegades*

December 1990
A Rewritable Optical Disk Library System for Direct Access Secondary Storage, *Donald J. Stavely, Mark E. Wanger, and Kraig A. Proehl*
Magnetooptical Recording Technology
Integrating the Optical Library Unit into the HP-UX Operating System,
Mechanical Design of an Optical Disk Autochanger, *Daniel R. Dauner, Raymond C. Sherman, Michael L. Christensen, Jennifer L. Methlie, and Leslie G. Christie, Jr.*
Optical Disk Autochanger Servomechanism Design, *Thomas C. Oliver and Mark J. Bianchi*
Data Capture System
Error Injection
Qualification of an Optical Disk Drive for Autochanger Use *Kevin S. Saldanha and Colette T. Howe*
A CD-ROM Drive for HP 3000 and HP 9000 Computer Systems, *Edward W. Sponheimer and John C. Santon*
Error Correction Implementation and Performance in a CD-ROM Drive, *John C. Meyer*
Error Detection and Correction Primer
Providing Software Protection Capability for a CD-ROM Drive, *Kenneth R. Nielsen*
Support for the ISO 9660/HSG CD-ROM File System Format in the HP-UX Operating System, *Ping-Hui Kao, William A. Gates, Bruce A. Thompson, and Dale K. McCluskey*
X.25 Packet Assembler/Disassembler Support in the HP 3000 Data Communications and Terminal Controller, *Jean-Pierre Allègre and Marie-Thérèse Sarrasin*
An Object-Oriented Message Interface for Testing the HP 3000 Data Communications and Terminal Controller, *Frédéric Maioli*
Effect of Fiber Texture on the Anisotropic Dimensional Change of Cu 1.8 wt% Be, *Frank E. Hauser and Nguyen P. Hung*

PART 2: Subject Index

P

Q

R

S

T

U

PART 3: Product Index

PART 4: Author Index

Effect of Fiber Texture on the Anisotropic Dimensional Change of Cu 1.8 wt% Be

The dimensional changes in cold-drawn Cu 1.8 wt% (11.4 at%) Be rods resulting from aging are investigated. The dimensional changes are nearly isotropic for as-quenched specimens but are anisotropic for cold-drawn specimens. The theoretical dimensional changes predicted based on the degree of preferred orientation, the crystallographic data of Cu-Be, and the geometry of the specimens agree with the experimental results.

by Frank E. Hauser and Nguyen P. Hung

COPPER BERYLLIUM (Cu-Be) ALLOY is traditionally used for spring connectors in the electronic industry because it has high conductivity, is platable, is low in cost, and has high strength with a relatively low modulus of elasticity (good spring characteristic). The material is usually machined when it is still soft after being solution-quenched and cold-worked. Machined parts are then precipitation hardened (aged) at an elevated temperature. This nearly doubles the mechanical strength without significantly changing platability or conductivity.

For precision components with tolerances on the order of ±5 μm (±0.0002 in), the use of Cu-Be alloy is limited because of the inconsistent dimensional changes that accompany aging. The shrinkage of Cu-Be during aging has been investigated by several other researchers[1,2,3] who assumed isotropic shrinkage of this material. This paper investigates the anisotropic dimensional changes of Cu-Be during aging, and discusses the effects of cold drawing, aging time, and aging temperature.

Experiments

Table I lists the properties of the tested material. The tested bars were centerless ground with minimum material removal, then machined into cylinders 25 mm (1.0 in) long. Next, the ends of these cylinders were ground parallel within 1.25 μm (50 μin). The specimens were ultrasonically cleaned, then aged at different temperatures from 200°C to 480°C (390°C to 900°F) in a mixture of 95% N_2 and 5% H_2 to minimize any measurement error caused by surface oxidation.

All samples were measured before and after each aging period. The measurements were performed in an environmentally controlled room in which temperature was held to 23±1°C. Diameters were measured with a scanning He-Ne laser system with resolution of 0.2 μm (10 μin) and accuracy of ±0.8 μm (±30 μin). The specimen lengths were measured with an indicating system with a resolution of 0.2 μm (10 μin) and accuracy of ±1 μm (±40 μin).

An x-ray diffraction method was used to characterize the sample texture. Inner and outer pieces of a bar were removed on a traveled wire electrical discharge machine* as shown in Fig. 1a. The diffractometer was set up for CuK_α x-rays with a Ni filter and a 0.4° beam split. The areas under the (200) and (111) diffraction peaks were measured with a planimeter and compared with data for copper powder.

Table I
Composition and Size of the Test Pieces

Material: Cu 1.8 wt% Be 0.23 Co 0.10 Si
Condition: Solutionized at 775°C (1425°F), Water-Quenched

	Cold-Drawn, % Diameter Change			
	0	10	20	30
Size (mm)	14.27	12.70	11.43	10.06
Size (in)	0.562	0.500	0.450	0.396

Results

While the as-quenched specimens had the same random texture as the copper powder, distinct evidence of a preferred orientation or nonuniform texture was found in the cold-drawn specimens. As Fig. 1b shows, the preferred orientation increases from the outside to the center of a drawn bar, and increases with the percent of cold drawing.

The percentage changes in the lengths and diameters of the cylindrical samples for different aging times at 315°C (600°F) are plotted in Fig. 2. The effects of aging time and temperature on the dimensional changes are shown in Figs. 3 and 4, respectively.

Discussion

The decomposition of the supersaturated phase of Cu-Be by precipitation has been found by other researchers[4,5,6] to be:

*A traveled wire electrical discharge machine uses a moving conductive wire as an electrode to erode another conductive part with a series of tiny electrical sparks.

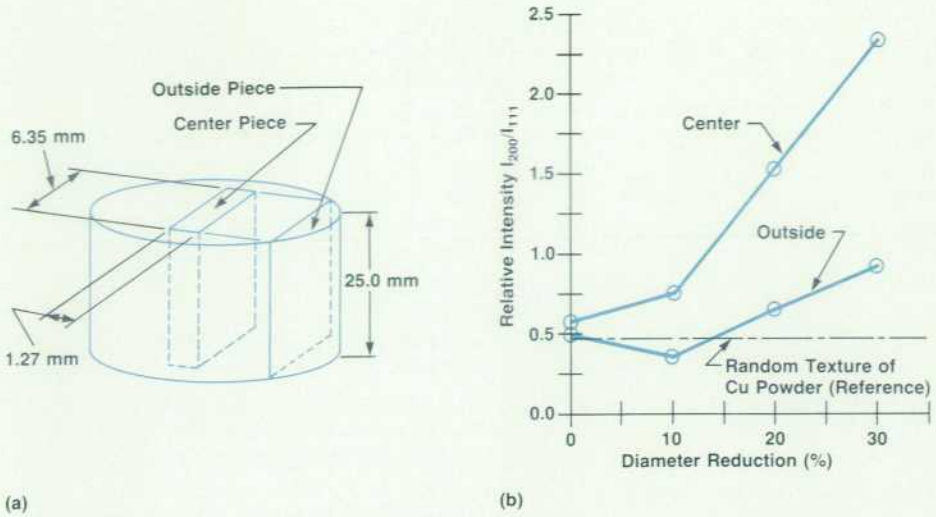


Fig. 1. *(a) Specimens for the diffractometry method. (b) Nonuniform textures caused by cold drawing.*

supersaturated $\alpha \rightarrow$ GP zone $\rightarrow \gamma'' \rightarrow \gamma' \rightarrow \gamma$.

In a supersaturated phase of this alloy, there is a random distribution of Be atoms in a solid solution with Cu atoms. The GP (Guinier-Preston) zones are locations where precipitates form.

The equilibrium γ phase does not exist in the ranges of aging temperature and time used in this experiment. The GP zone, γ'', and γ' phases consist of multilayered plates formed on {100} planes. Shrinkage results from the phase transformation because the unit cell dimensions of the GP zone, γ'', and γ' phases are smaller than those of the α phase, as summarized in Table II.

Table II
Dimensions of the Unit Cells

Phase:	GP zone	γ''	γ'	α
Lattice:	tetragonal	tetragonal	cubic	FCC
a = b (Å)	2.53	2.53	2.70	3.58
c (Å)	3.22	2.90	2.70	3.58

where a, b, and c are the unit cell dimensions and FCC indicates a face-centered cubic lattice.

The coherency of the plate-type precipitates allows dimensional change in the direction normal to the plate, but prohibits any shrinkage in the habit plane (the flat plane in which precipitation plates tend to form). If this were not the case, voids would form at the perimeters of the precipitate plates.

Anisotropic dimensional change on a macroscopic scale is the combined effect of:

- Microscopic dimensional change caused by each precipitate
- Formation of the precipitates on habit planes and directional shrinkage relative to these planes
- Preferred orientation of the habit planes induced by the cold drawing process.

Consider a cylinder of diameter D and length L. Its volume V is given by:

$$V = \frac{\pi D^2}{4} L.$$

Upon aging, the volume change resulting from small changes in both diameter and length is:

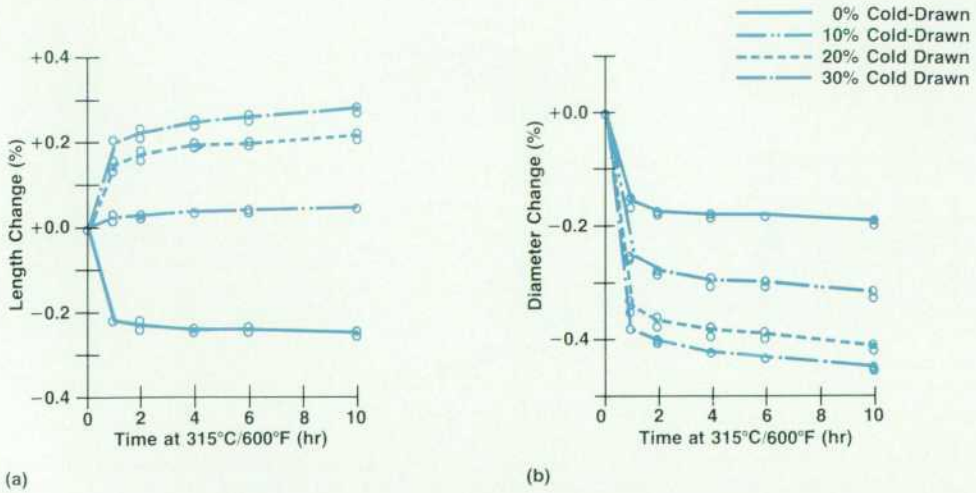


Fig. 2. *Effect of aging time at 315°C on (a) axial and (b) transverse dimensional changes.*

$$\frac{dV}{V} = 2\frac{dD}{D} + \frac{dL}{L} \quad (1)$$

All of the GP zone, γ″, and γ′ phases are present after aging at 315°C (600°F) for a short time.[5] It is assumed for simplicity that only the γ′ phase is present after long aging time (10 hr). The theoretical volume change dV/V after complete transformation of the α phase to the γ′ phase in Cu 1.8 wt% Be was calculated[7] to be:

$$\frac{dV}{V} = -0.61\%. \quad (2)$$

This agrees with results from other researchers.[1,3] By combining equations 1 and 2, the plot of dD/D (%) versus dL/L (%) is found to be a straight line:

$$\frac{dD}{D} = -0.5\frac{dL}{L} - 0.305\%. \quad (3)$$

Equation 3 is plotted in Figs. 3 and 4 together with the experimental data. As shown in Fig. 3, the measured shrinkage data of the as-quenched specimens after long aging at 315°C (600°F) agrees with the isotropic shrinkage calculated from equation 1:

$$\frac{dL}{L} = \frac{dD}{D} = \frac{1}{3}\frac{dV}{V} \approx -0.2\% \quad (4)$$

Consider a cubic lattice whose [001] direction coincides with the bar axis (longitudinal axis of a cylindrical bar). There are only two planes, (001) and (002), perpendicular to the bar axis, but there are four planes—(100), (200), (010), and (020)—parallel to this direction. Assume that the probabilities of forming plate-type precipitates on all {001} planes are equal. Since the aging-induced shrinkage is significant in the direction perpendicular to the habit plane as pointed out earlier, and since there are more habit planes parallel to the bar axis, it can be seen that the transverse direction has to shrink more than the axial direction when the material lattice structures align themselves after cold drawing so that the [001] fiber axis is parallel to the bar axis.

The experimental data can now be explained. First, since the precipitation process happens quickly at 315°C (600°F), dimensions of the aged samples change mostly during the first hour, then vary slowly as the aging time increases, as shown in Fig. 2.

Second, the data points start slightly off the anisotropic shrinkage line in Fig. 3 at the early stage of precipitation, but approach this line as the aging time increases as indicated by the direction of the arrows. (Recall that the theoretical line represents the near-equilibrium condition after very long aging time.) Experimental data also suggests that the axial dimensions of the textured samples expand to conserve their volumes at equilibrium as modeled by equations 2 and 3.

Third, for aging below 260°C (500°F), the dimensional change of Cu-Be is more complicated. As seen in Fig. 4, the as-quenched samples always shrink isotropically because of their random orientation. The textured samples change their dimensions anisotropically but the experimental data converges to the theoretical line represented by equation 3.

Conclusions and Recommendations

The anisotropic dimensional change of cold-drawn, then aged Cu-Be rods was investigated. This paper shows that:

- Cold drawing of Cu-Be forms a distinct [001] fiber texture.
- The anisotropic dimensional change during an aging process is significantly influenced by the texture.
- The as-quenched samples shrink isotropically in the temperature range 200°C to 480°C (390°F to 900°F) because of their near-random texture. The textured specimens change their dimensions anisotropically. The transverse dimension of cylindrical samples shrinks but the axial dimension expands to conserve their volume after long aging at temperatures greater than 315°C (600°F).
- A more precise shrinkage model should consider the different phases in the aged samples, grain boundaries, and other volume defects that affect the sample dimensions.
- Dimensional changes are dominated by cold-work-induced texture. Therefore, if the amount of cold work is controlled, then a fixed set of shrinkage factors for a

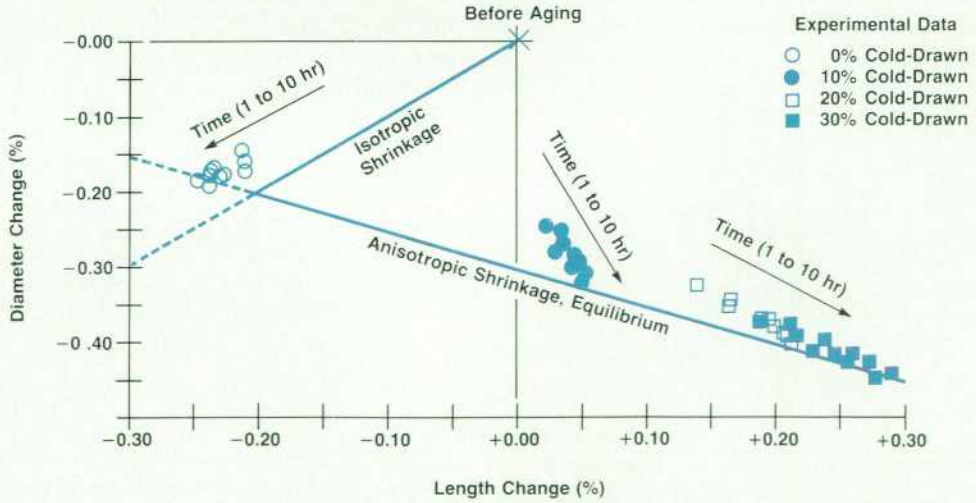


Fig. 3. *Effect of aging time on the dimensional change of Cu 1.8 wt% Be. Aging temperature 315°C.*

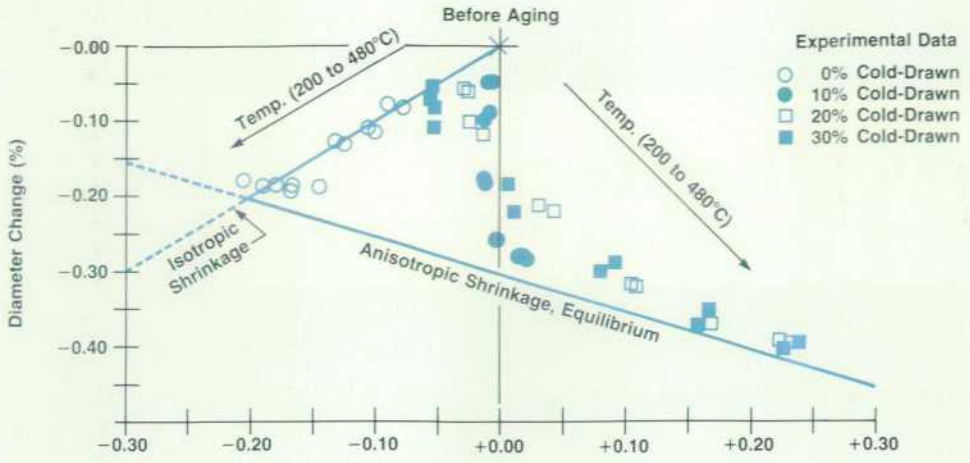


Fig. 4. *Effect of aging temperature on the dimensional change of Cu 1.8 wt% Be. Aging time 2 hours.*

particular aging process can be used in practice to predict dimensional changes of machined parts.

Acknowledgments

The authors would like to express their appreciation to Brush Wellman Inc., Cleveland, Ohio, U.S.A. for providing the material, and to Hewlett-Packard Company, Santa Rosa, California, U.S.A. for use of the metrology equipment.

References

1. M.I. Gitgarts and A.V. Tolstoy, "Volume Changes in a Copper Beryllium Alloy," *The Physics of Metals and Metallography*, Vol. 48, no. 2, August 1979, pp. 197-199.
2. H. Kreye, "Shrinkage and Warping During Age Hardening of CuBe Alloys," *Metals*, Vol. 29, 1975, pp. 1118-1121.
3. Z.P. Pastukhova and N.V. Vasilev, "Volume Changes, Warping, and Residual Stresses in Beryllium Bronze During Heat Treatment," *Metal Science and Heat Treatment*, Vol. 12, September-October 1979.
4. V.A. Phillips and L.E. Tanner, "High-Resolution Electron Microscopy Observation on GP Zones in an Aged Cu 1.97 wt% Crystal," *Acta Metallurgica*, Vol. 211, no. 4, April 1973, pp. 441-448.
5. R.F. Rioja and D.E. Laughlin, "The Sequence of Precipitation in Cu 2 wt% Be Alloys," *Acta Metallurgica*, Vol. 28, no. 9, September 1980, pp. 1301-1313.
6. Y.M. Koo, *Atomic Structure of Cu 10.9 at% Be Alloys in the Early States of Aging*, PhD Thesis, Northwestern University, 1987.
7. N.P. Hung, *The Anisotropic Dimensional Change Due to Aging in Cold Drawn Cu 1.8 wt% Be*, PhD Thesis, University of California at Berkeley, 1987.

Fr: DICK DOLAN/16PUB 00113286
446

To: LEWIS, KAREN
CORPORATE HEADQUARTERS
DDIV 0000 20BR

POSTER

Hewlett-Packard Company, 3200 Hillview Avenue, Palo Alto, California 94304

ADDRESS CORRECTION REQUESTED

Bulk Rate
U.S. Postage
Paid
Hewlett-Packard
Company

HEWLETT-PACKARD JOURNAL

December 1990 Volume 41 • Number 6

Technical Information from the Laboratories of Hewlett-Packard Company
Hewlett-Packard Company, 3200 Hillview Avenue
Palo Alto, California 94304 U.S.A.
Hewlett-Packard Marcom Operations Europe
P.O. Box 529
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

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.

5091-0070E