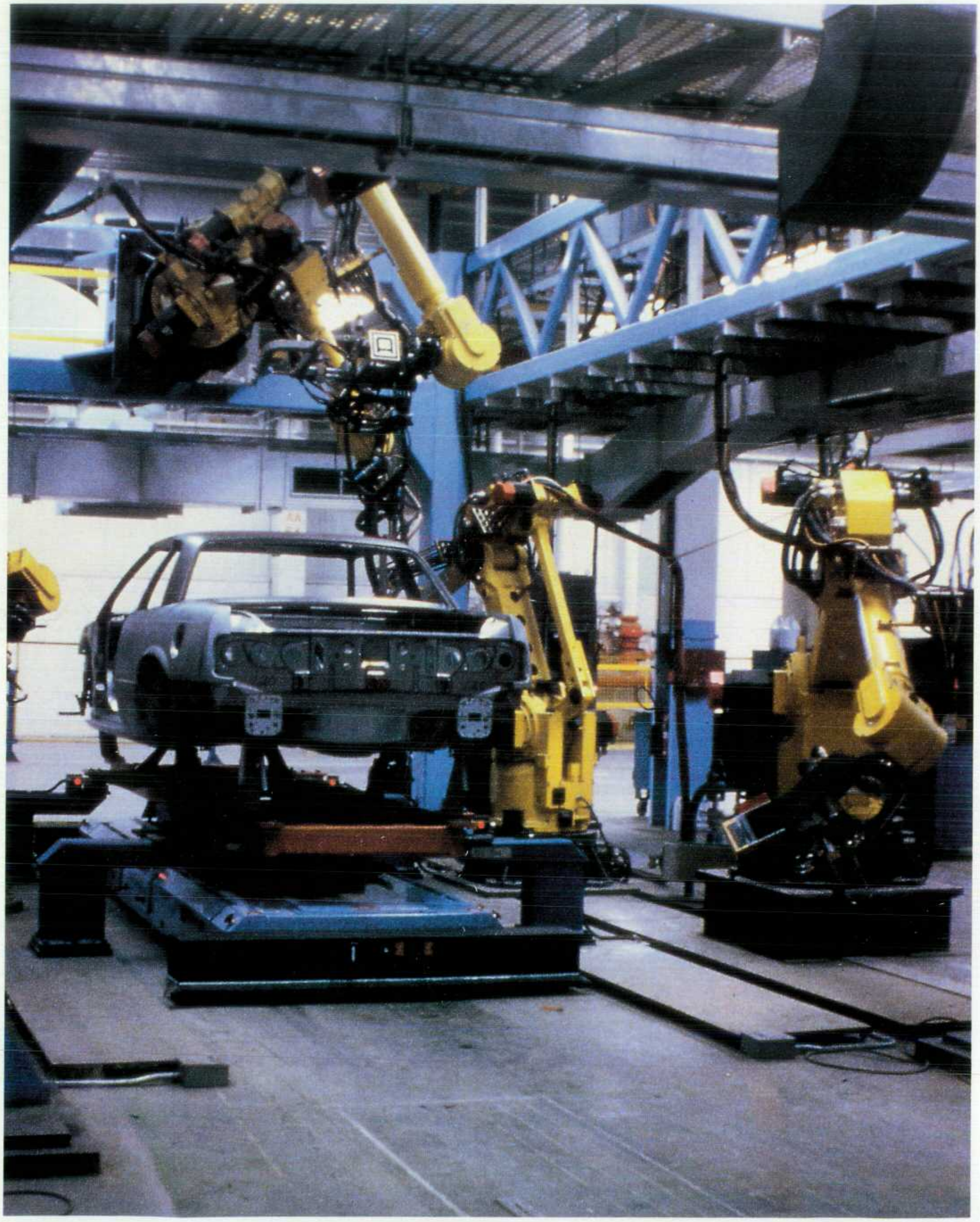
HEWLETT-PACKARD JOURNAL

AUGUST 1990



HEWLETT PACKARD

HEWLETT-PACKARD JOURNAL

August 1990 Volume 41 • Number 4

Articles

Editor, Richard P. Dolan • Associate Editor, Charles L. Leath • Assistant Editor, Gene M. Sadoff • Art Director, Photographer, Arvid A. Danielson
Support Supervisor, Susan E. Wright • Administrative Services, Diane W. Woodworth • Typography, Anne S. LoPresti • European Production Supervisor, Sonja Wirth

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

Computer integrated manufacturing, or CIM, is the vision and the goal of manufacturing companies today. In this vision, corporate mainframe computers and engineering workstations transmit order and design data automatically to the factory floor, where workcell controllers pass it on to robots and numerically controlled machines. Production, quality, and shipping information is collected automatically by the corporate mainframes for management reports and billing. In reality, as the authors of the article on page 6 describe it, CIM has traditionally meant equipment from different vendors residing in isolated "islands of automation" because, in most cases, no single vendor can supply all of a factory's requirements, and the proprietary components used by the various vendors can't communicate with other vendors' equipment. In this environment, application development and training are difficult, and a centrally managed and controlled network is virtually impossible. About ten years ago, General Motors Corporation formed a task force to address these problems. Many large and small manufacturers and leading equipment vendors, including Hewlett- Packard, have been involved in this effort. The result is the Manufacturing Automation Protocol (MAP). MAP is based on the Reference Model for Open Systems Interconnection (OSI) of the International Organization for Standardization (ISO), which in this age of networking has been appearing with some regularity in this publication. MAP specifies a set of standard communication services for factory automation, and has been accepted as an international standard by the ISO. HP's implementation of the latest version, MAP 3.0, is described on pages 6 to 60 of this issue. It has three main components: the File Transfer, Access, and Management services (FTAM, page 24), the Manufacturing Message Specification services (MMS, page 31), and the X.500 directory services (page 15). A standard interface between these services and factory automation applications is provided by the HP MAP 3.0 upper layer architecture (page 11). HP MAP 3.0 runs under the HP-UX operating system on HP 9000 Series 800 PA-RISC computers and connects to a network by means of the HP OSI Express card (see the February 1990 issue). Three new software modules in the HP-UX kernel provide reliable data transfer between the host HP 9000 computer and the HP OSI Express card (page 40). Because the purpose of the Manufacturing Automation Protocol is to allow equipment from different vendors to communicate, interoperability testing is necessary to ensure that HP MAP 3.0 services are compatible with other vendors' services and to expose errors in HP's and other vendors' implementations. The article on page 50 describes the results to date. The development of HP MAP 3.0 was a major software project involving ten project teams at three HP divisions in different parts of the U.S.A. To ensure the successful integration of all of the software, firmware, and hardware elements, a generic software integration lifecycle was developed. Described in the article on page 54, it may be applicable to other large, multidivisional projects.

The high-speed computation, communications, and signal processing that we have available today are made possible by advanced integrated circuit technologies with names like CMOS, BiCMOS, ECL, and GaAs. Testing devices built using these technologies requires high-performance instruments that often depend for their performance on the same advanced IC technologies. The HP 8130A and HP 8131A pulse generators are good examples of this bootstrap process. The HP 8131A generates pulses with 200-picosecond rise and fall times at rates up to 500 megahertz. The HP 8130A offers variable rise and fall times at pulse rates up to 300 megahertz. The two instruments share the same architecture and timing circuits, as described in the article on page 64, but use different output systems. A custom high-speed bipolar IC generates all of the timing parameters including pulse rates to 500 megahertz and pulse widths from 500 picoseconds to 99.9 milliseconds. The HP 8131A's output amplifier (page 79) is a thick-film hybrid circuit containing a custom gallium arsenide integrated amplifier circuit. The HP 8130A's fast, variable transitions are produced by a custom gallium arsenide slope generator, and its output amplifier, which requires high linearity because of the variable output amplitude and slopes, is a custom high-speed bipolar circuit (page 85).

R. P. Dolan
Editor

Cover

The picture shows an automated workcell with robots and controllers at the General Motors Corporation facility in Oshawa, Canada. Providing the communication links between the components in the workcell is a typical application of the Manufacturing Automation Protocol 3.0 (MAP 3.0).

What's Ahead

The cover subject of the October issue will be the HP 11974 Series of millimeter-wave spectrum analyzer RF sections, which have magnetically tunable barium ferrite preselection filters and operate in the 26.5-to-75-GHz range. Other articles will describe the HP Interactive Visual Interface, a toolkit for developing graphical user interfaces, and the HP Device Interface System, a tool for developing interfaces between computers and factory-floor devices such as robots and programmable controllers. We'll also have research reports on measurements of R, L, and C in VLSI packages, simulation of air flow in computer cabinets, and statistical simulation of circuit performance distributions in production.

HP Manufacturing Automation Protocol 3.0

The Manufacturing Automation Protocol (MAP) is an intervendor program that addresses the problems that have plagued factory automation in the past. HP's MAP 3.0 product provides international standard network services and protocols and a multivendor MAP programmatic interface.

by Collin Y.W. Park and Bruce J. Talley

COMPUTER INTEGRATED MANUFACTURING (CIM) has traditionally meant equipment from different vendors residing in isolated "islands of automation" because the proprietary components used by the various vendors could not communicate with each other. In many cases, they could not even share the same cable. As a result, cabling was a nightmare, as was the effort of porting applications across different vendors' equipment. Training for these different networks was difficult to provide for users and support staff, and the cost of the various kinds of test equipment to keep it all running was also high. Data throughput was typically low, and centralized management and control of the network became virtually impossible.

The alternative to this collection of incompatible technology was to use only equipment from a single vendor, thus locking the user into a future dominated by that vendor, regardless of price-performance or functionality benefits provided by other equipment vendors. This was really no alternative at all, since no single vendor could meet all the equipment requirements on the factory floor.

The Manufacturing Automation Protocol (MAP) specification was written to address these problems. Started by General Motors Corporation and involving many other large and small manufacturers as well as leading equipment vendors in the industry, MAP is the product of years of networking and manufacturing experience.

MAP specifies a set of standard communication services for factory automation. The specifications provide compatibility from the application layer (vendor X's computer can download to vendor Y's robot) to the physical layer (all MAP devices connect to the same high-speed cable, using the same standard connector). MAP's application program interface (API) allows users to port their applications across vendors, and the standardized communication technology and protocols cut training and maintenance costs.

The MAP services, and the data communication protocols that provide them, are derived from the Open Systems Interconnection (OSI) model as defined by the International Organization for Standardization (ISO) (see box on page 8).

HP and MAP

HP's involvement with the MAP program began in 1984 with MAP 1.0. An implementation of MAP 1.0 was done on the HP 1000 computer system and demonstrated at the National Computer Conference of 1984. The MAP Users Group was created that same year and HP was a founding member. In 1985, HP demonstrated MAP 2.1 at Autofact, a factory automation forum. Also in that year, HP joined the European MAP Users Group and completed the first functional installation of a MAP system at General Motors Corporation in Detroit, Michigan. In 1987 HP introduced MAP 2.1 on the HP 9000 Series 800 computer system. And in 1989 HP released the current version of MAP 3.0 for volume shipments on the HP 9000 Series 800 computers.

The HP MAP 3.0 products provide the MAP-specified services on HP 9000 Series 800 computers. The MAP API is included in the product, so that customers can write CIM applications, such as factory device monitoring and control programs. Efficient user interfaces are provided for configuration, verification, and troubleshooting.

OSI Services Provided in HP MAP 3.0

Three OSI services are currently provided in HP MAP 3.0: the ISO File Transfer, Access, and Management (FTAM) services, the ISO Manufacturing Message Specification (MMS), and the ISO/CCITT X.500 directory services. The HP FTAM and MMS products are called HP MAP 3.0 FTAM/800 and MMS/800 respectively. Only the services provided by FTAM and MMS can be directly accessed by the user. The X.500 directory services are accessed indirectly through FTAM and MMS.

FTAM provides users with the ability to manipulate files on remote systems in an internationally standardized way. The programmatic interface includes subroutines that read and write individual records in files and subroutines to delete files and copy files between systems or on the same system. FTAM is described in more detail in the article on page 24.

MMS is designed for use with manufacturing devices such as robots. Various control and monitoring operations are provided. For example, there are subroutines for reading and setting variables such as counters and alarm condition thresholds. A bulk data transfer facility (e.g., for downloading a program) is also provided. MMS is described in more detail in the article on page 31.

Both HP FTAM/800 and HP MMS/800 use the OSI directory service X.500 to find the address of an application or resources on another system or device. The HP X.500 directory services are described on page 15.

Architectural Overview

In HP's MAP 3.0 product, the application services are implemented in software residing on the HP 9000 Series 800 computer system. The Application Control Service Element (ACSE) protocol and most of the OSI lower layer functions (from the presentation layer* to the IEEE 802.4 physical layer protocols) are implemented on the HP OSI Express card,[1] which is a microprocessor-based I/O card that plugs into the backplane of the host computer. Fig. 1 shows the overall architecture of HP's MAP 3.0 product.

User Application Programs. These are programs written by end users or independent software vendors that may use FTAM for file access and/or MMS for monitoring factory floor devices. These applications use the MAP application program interfaces to access the HP MAP 3.0 services. The application program interfaces are libraries that are linked into the user's program to enable communication with the various service provider processes.

Service Provider Process (SPP). The service provider processes contain the modules that provide the MAP services (FTAM and MMS) and the asynchronous operation capabilities specified by the MAP protocol.

*The presentation layer's syntax transformation function is performed on the host system.

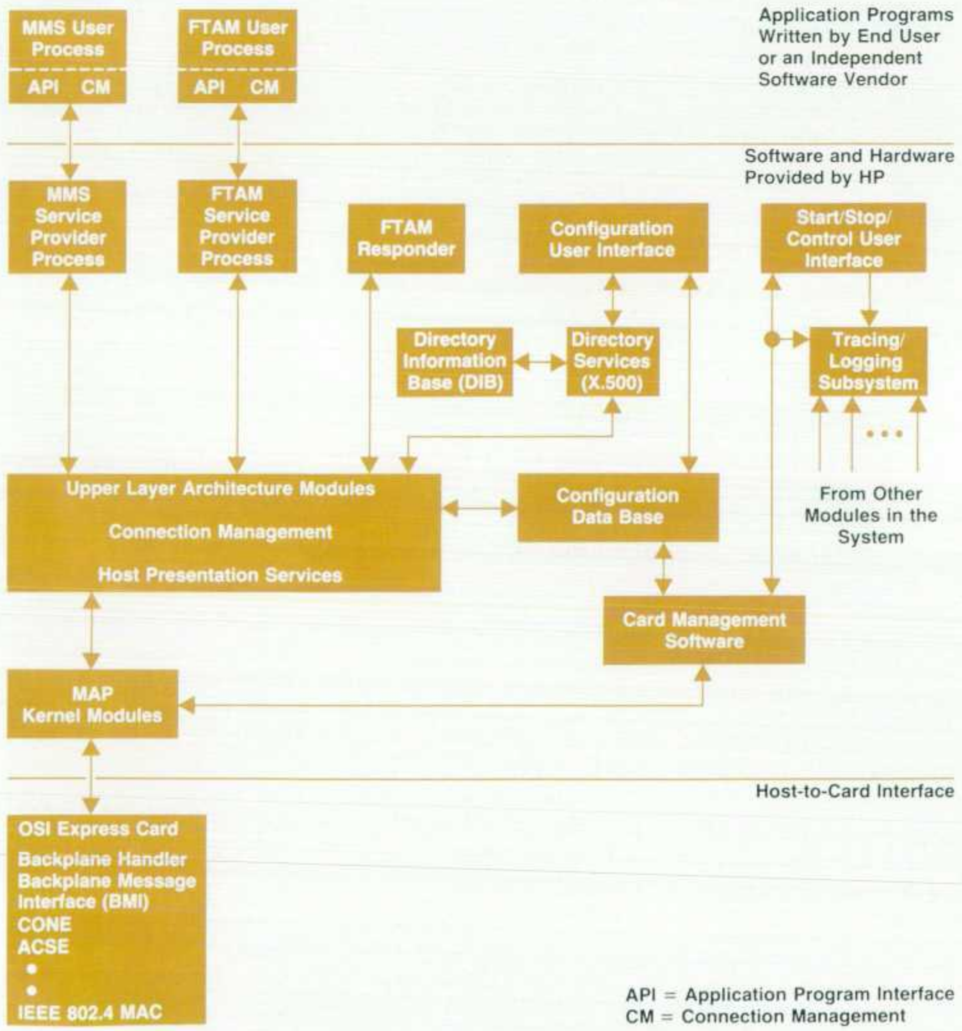
FTAM Responder. The FTAM responder is an FTAM process that handles requests for file manipulation activities on the local machine. Typically these requests are initiated by remote users. If a local user wants to access some remote file, first the local FTAM SPP is used. When the request reaches the remote machine, the FTAM responder on that machine handles the request.

Upper Layer Architecture Modules. The upper layer architecture modules provide connection management and syntax transformation services to the SPPs and the FTAM responder. The upper layer architecture modules also provide a platform for the development of OSI services. The article on page 11 describes the upper layer architecture in more detail.

MAP Kernel Modules. The MAP kernel modules, which are software modules embedded in the HP-UX kernel, are responsible for communicating with the HP OSI Express card and hence to other systems or devices on the network.

Configuration Data Base. This data base contains information that is pertinent to the local system, such as the default levels of tracing and logging and the addresses of the applications on the local host.

Configuration User Interface. This user interface allows users to enter data into the local configuration data base. It can also be used to update information on a (possibly



MMS User Process
API CM
FTAM User Process
API CM
Application Programs Written by End User or an Independent Software Vendor
Software and Hardware Provided by HP
MMS Service Provider Process
FTAM Service Provider Process
FTAM Responder
Configuration User Interface
Start/Stop/ Control User Interface
Directory Information Base (DIB)
Directory Services (X.500)
Tracing/ Logging Subsystem
From Other Modules in the System
Upper Layer Architecture Modules
Connection Management
Host Presentation Services
Configuration Data Base
Card Management Software
MAP Kernel Modules
Host-to-Card Interface
OSI Express Card
Backplane Handler
Backplane Message Interface (BMI)
CONE
ACSE
IEEE 802.4 MAC
API = Application Program Interface
CM = Connection Management

Fig. 1. *The main components of the HP MAP 3.0 architecture. HP supplies the software modules in the middle section and the hardware module in the bottom section.*

Overview of the OSI Reference Model

Fig. 1 shows the seven layers of the OSI reference model. The application layer provides the communication services required by the end user. Each OSI application contains the Association Control Service Element (ACSE)[1] and at least one other application service element (ASE). ACSE controls the establishment and destruction of the association (i.e., the cooperative relationship) between the application and its peer. The other ASEs provide the communication functions to support the specific OSI application. For example, the FTAM ASE contains a module called the FTAM protocol machine, which creates the proper sequences of operations to manage the communication and the transfer of file names and file data between systems. The application, not the ASE, is responsible for requesting the opening of files and reading from or writing to files.

The presentation layer performs negotiation and transformation of syntax. For example, if one system uses EBCDIC and another uses ASCII, then a character set understandable by both systems must be chosen if they are to communicate. The systems must agree to the character set to be used for communication (syntax negotiation), and at least one system will need to do character set translation (syntax transformation).

The session layer provides dialogue control and synchronization. Some applications need facilities to keep track of who's going to talk next, like a radio operator saying, "Over," to signal the person at the other end to start talking.

The transport layer ensures that data sent from one user is communicated to the other user in order, without loss or duplication. This may involve error recovery. For example, if transient errors cause one data packet to be garbled, the transport layer may retransmit that packet. Multiplexing may also be done to allow several conversations to be held using the same wire or the same connection.

Routing is the primary function of the network layer. If two systems are not on the same physical medium, then data packets may have to be forwarded through a gateway, which is represented in Fig. 1 by the box labeled intermediate system. The gateway or intermediate system may in fact have its own applications running at the same time, but the figure shows only the part of the intermediate system that is used for communicating between the two end systems. The network entity in the source end system is responsible for picking an intermediate system that will eventually be able to deliver the data packets to the destination. The network entity in each intermediate system that is part of the route is responsible for selecting the next intermediate system in the route. The last intermediate system is responsible for sending the packet to the destination.

The function of the data link layer is to ensure data integrity between two physically connected systems. Functions such as framing and checksumming are found in data link protocols. Flow control and error recovery procedures are also sometimes used.

The physical layer ensures the physical (e.g., electrical and mechanical) compatibility between end systems, and the integrity of physical media used for data transmission.

References

1. *Information Processing Systems - Open Systems Interconnection - Service Definition for the Association Control Service Element*, ISO 8649: 1987 (E).

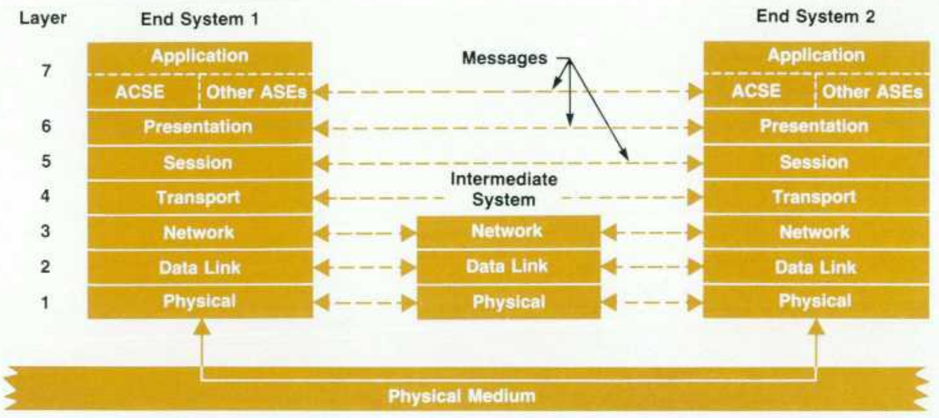


Fig. 1. *The OSI reference model.*

remote) directory system.

Directory Services. The directory services in MAP are based on the OSI X.500 standard. This software provides facilities for determining the locations of applications and resources in a network.

Directory Information Base (DIB). The DIB is the data base that implements the structure of the X.500 directory. It contains the names, addresses, and other information required to find the locations of remote systems and/or applications. The DIB may reside on the local system or a remote system.

Start/Stop/Control User Interface. This interface allows users to start and stop MAP networking, and to monitor and control the level of tracing and logging. Tracing and logging information is used to facilitate troubleshooting the network.

Card Management Software. This software is invoked by the start/stop/control user interface to download the HP OSI Express card's software, and to control the card's operations.

Tracing/Logging Subsystem. This subsystem is responsible for taking the traces and event logs from all of the other HP MAP 3.0 software modules. Each module in the system has a connection to the tracing/logging subsystem. This uniform tracing and logging scheme was designed into HP's MAP product from the beginning to ensure consistent tracing and logging policies and consistent formatting of the tracing and logging information.[2]

Scenarios

The following scenarios illustrate how the components of the HP MAP 3.0 architecture work together.

Configuration and Startup. Once the software and hardware are installed, the system administrator, with the assistance of HP field personnel, typically invokes the command mapconf to configure and start HP MAP 3.0. The system administrator or user can use other commands besides mapconf for configuring and administering the MAP subsystem. Procedures are provided for updating the configuration of a system already running MAP 3.0. There is also a fast startup and shutdown facility.

Mapconf invokes the screen-based configuration user interface, and the system administrator updates the configuration data base with the name of the card and the network layer address to be used. The level of tracing and logging can also be configured at this time. Next, mapconf invokes the start/stop/control user interface and, playing the part of the user, instructs the card management software to download the card-resident software to the HP OSI Express card. Once this is accomplished, the tracing/logging subsystem is started based on information stored in the configuration data base.

Mapconf's next step is to invoke the configuration user interface to allow applications to be configured. First, access to the OSI directory service must be ensured. If the directory information base (DIB) is located on the system, the system is configured as a directory system agent (DSA). The directory system agent is an X.500 process that handles communication with a user requesting access to the DIB. If the DIB is not located on the system, the system is configured as a directory user agent (DUA). A directory user agent is an X.500 process that communicates with the DSA on behalf of the user. The system administrator will also configure the name and address of the FTAM responder. Configuration limits, such as maximum number of simultaneous remotely initiated file transfers, may also be configured at this time.

At this point, the system administrator can enter the FTAM responder's name-to-address mapping into the DIB using the X.500 directory services. If the local system is a DUA, then this mapping must be recorded in the DIB via a remote DSA. The directory services software is invoked and uses the connection management module to establish a connection to the remote directory system agent. The host presentation services module performs the syntax transformations that allow this DUA-to-DSA communication. Other applications are configured similarly.

Once the configuration process is complete, mapconf starts the FTAM responder and then invokes a screen-based verification program that allows the system administrator to see whether the system can indeed communicate with others on the network. The startup process is then complete.

File Transfer. To see how the architecture carries out a user's requests, consider a remotely initiated file transfer. Suppose application A on a remote system (not necessarily an HP system) wants to transfer a file to the local system. Application A will first determine the address of the local FTAM responder, perhaps by using the OSI directory service. Application A then sends a connect request on the IEEE 802.4 token bus, and the request is received by the local HP OSI Express card. This event is passed through the MAP kernel modules to the FTAM responder, which has been waiting for just such an event. The local FTAM responder calls the connection management module in the upper layer architecture modules to ensure that no configured limits are about to be exceeded. If everything is satisfactory, the FTAM responder continues processing.

The FTAM responder on the local system must process the connect indication (it was called a request on the initiating side but it's an indication on the receiving side), and the presentation services module is invoked to perform the required syntax transformations. Subsequent data exchanges tell the FTAM responder that the remote system wants to send a file, and, if security checks pass, the FTAM responder will write the data into the appropriate file.

Programmatic Interface. The HP MAP 3.0 application program interface allows the user to write FTAM and MMS applications that automatically use the parameters in the configuration data base. The application programmer configures the user application before running the program. When it is run, the user's program calls the connection management module in the user process, which checks to make sure that all conditions are within configured limits (e.g., maximum number of simultaneous connections) so that a user application can have access to MAP resources.

Once the API is satisfied that the program has the right to use MAP resources, the program can make high-level calls (e.g., transfer file A from system B to device C) or low-level calls (e.g., read next record). Calls can also be synchronous (e.g., return when the operation is complete) or asynchronous (e.g., read a record or transfer a file in the background).

To provide the asynchronous operations offered by MAP, the HP MAP 3.0 product causes a service provider process to be spawned whenever a user program calls the connection manager's activate function. Communication is established between the user's process and the SPP using a local interprocess communication mechanism. Asynchronous requests can thus be handled, and high-level calls like file transfer can be executed with almost no performance penalty because the process switching overhead is amortized over the entire high-level operation.

Conclusion

The result of many years of manufacturing and networking experience, MAP is an intervendor program that addresses the problems that have plagued factory automation in the past. HP's MAP 3.0 product combines international standard services and protocols with the multivendor MAP programmatic interface. Screen-based utilities simplify configuration and verification. A uniform tracing/logging policy and uniformly formatted output facilitate diagnostic and troubleshooting tasks. The MAP programmatic interface provides enforcement of configured limits, and allows asynchronous and high-level operations.

Acknowledgments

HP's MAP 3.0 products are the results of a cooperative joint development effort among HP's Colorado Networks Division, Information Networks Division, and Roseville Networks Division. Over the course of the hardware and software development, many managers and engineers worked on the project at these three divisions. A special thanks goes to the engineers responsible for the HP MAP 3.0 support and configuration tools, and for system testing. The supportability team consisted of Peter Chang, Kwang-Wei Yao, Don Goodnature, Mike Ganley, Jinn Su, Kathryn Vandiver, Randy Stille, Chye-Guan Chew, Doug Chan, Dave Woods, Pradeep Verma, Greg Gilliom, and Joan Lawler. The configuration team consisted of Brad Wilson, Eric Henderson, Ellen Chu, and Greg Buchanan. System testing of the products in IND was performed by Andy Lai's team of Rajiv Batra, Murali Subbarao, Paul Melmon, and Abraham Lui. Final testing was done in HP's HP-UX test technology center by Anita Mundkur's team of Joe Algieri, Silesh Bijj, Janice Gee, and Yori Huynh. Also deserving a special thanks are Doug Boliere, Todd Reece, Atul Bhatnagar, and Don Tiller for the vision and leadership they supplied to this challenging program.

References

1. *Hewlett-Packard Journal*, Vol. 41, no. 1, February 1990, pp. 6-77.
2. J.K. Shah and C.L. Hamer, "Support Features of the HP OSI Express Card," *Hewlett-Packard Journal*, Vol. 41, no. 1, February 1990, pp. 67-72.

Upper Layer Architecture for HP MAP 3.0 OSI Services

Based on the OSI standard for the application layer, the HP MAP 3.0 upper layer architecture provides a standardized structure that allows network application developers to focus on the services provided by their applications rather than the architecture necessary to interface to network protocols.

by Sanjay B. Chikarmane

TO FACILITATE THE DEVELOPMENT of the OSI (Open Systems Interconnection) application layer networking services, ISO (International Organization for Standardization) architects developed a framework consisting of terminology, concepts, and a structure for developing these services. Within this framework, called the Application Layer Structure (ALS),[1] the architectural model for all OSI application layer services is specified. In defining this model, the ISO architects refrained from using real implementation terms such as processes and libraries to avoid biasing the model toward a system offered by a specific vendor.

This article describes the HP MAP 3.0 upper layer architecture, which is an implementation of the OSI application layer specification.

Application Layer Structure

In ALS terminology, an application using the OSI services is called an *application process*. Note that the word process in this context is not necessarily the same as in an operating system context. In ALS, an application process is simply an abstraction of a real application. The ALS specification defines a structure for an application process that promotes a toolkit approach in which components of the application process perform well-defined functions. These components can be employed in different combinations to provide different services.

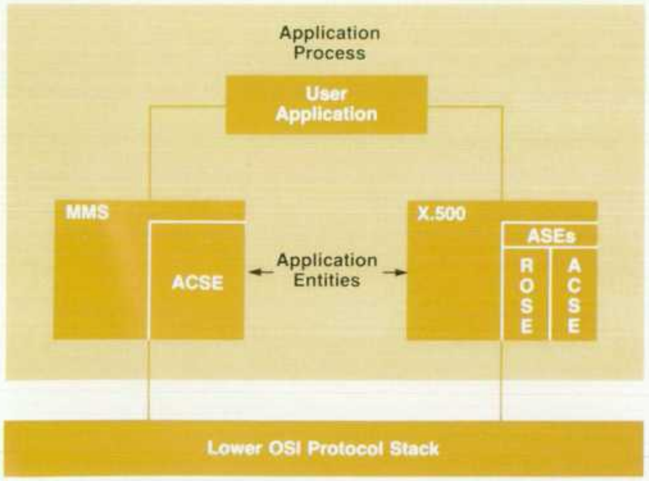


Fig. 1. *An example of the relationship between the upper layer networking services components and the OSI Application Layer Structure terminology applied to these components.*

An application process is composed of one or more components called *application entities* (see Fig. 1). An application entity can be looked upon as a networking service such as a file transfer facility that employs one or more networking protocols. The networking protocols are called *application service elements* (ASEs). A factory-floor status monitoring application is an example of an application process. It may use the services provided by the application entities MMS (Manufacturing Message Specification) and X.500. MMS is used for reading status variables from factory-floor machines and X.500 is used for accessing a directory data base containing information about various factory-floor applications. The ASEs, or networking protocols, are represented by MMS and ACSE (Application Control Service Element) for the manufacturing messaging service, and by X.500, ACSE, and ROSE (Remote Operation Service Entity) for the X.500 directory service. ACSE is an application layer protocol that is used to establish and terminate an association between applications which can be on the same system or different systems. ROSE is a generic OSI service that allows applications to invoke request/reply interactions with applications on remote systems. MMS and X.500 are discussed in detail on pages 31 and 15.

The toolkit approach now becomes apparent. Once well-defined ASEs with consistent interfaces are available, they can be combined to produce an application entity that provides a specific network service. And one or more such application entities can be used to build an application process.

Implementation Architecture

The HP MAP 3.0 upper layer architecture is based on the ALS specification. In this architecture an application process is implemented as one or more HP-UX processes working together. The upper layer architecture distinguishes between two types of processes: the service provider process and the user process. The service provider process is a process that represents an application entity. It is composed of modules representing ASEs and other protocols and functions. The user process is a process that contains an application program that uses the services provided by one or more service provider processes. The upper layer architecture is shown in Fig. 2.

The user starts a user process that issues primitives to activate one or more service provider processes. Each primitive call results in the HP-UX fork and exec operations on a service provider process.

User Process. The five modules in the user process provide the end user with a transparent interface to the network services.

- Application Program. This is the program that the end user uses to access the underlying networking services. For MAP 3.0, application programs are MAP applications used for factory automation.
- Application Program Interface. This is the interface through which the application program obtains access to specific networking services, such as file transfer (FTAM), manufacturing messaging (MMS), and the X.500 directory services. For MAP 3.0, these interfaces are specified by the MAP 3.0 subcommittees.[2]
- Event Manager. This module tracks pending requests made by the application program to the application program interface. The application program uses the event manager to synchronize the completion of requests.
- Connection Manager. This module is responsible for managing the establishment and termination of all connections for the application program.
- Messaging Routines. Interactions between the application program and the service provider processes are in the form of messages over HP-UX domain sockets. The messaging routines implement these messaging services, hiding the specific mechanisms used from the application program.

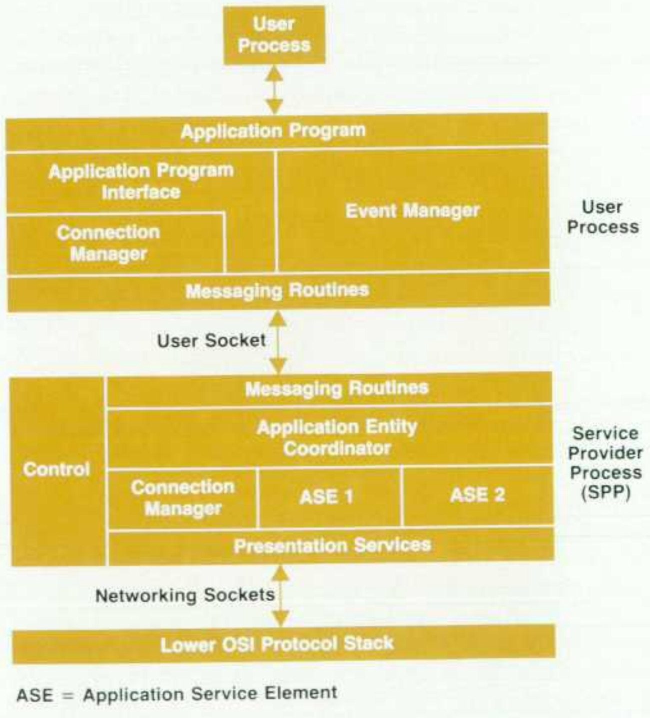


Fig. 2. *The upper layer architecture for HP MAP 3.0 networking services.*

When an application program activates one or more service provider processes, it can make a request through the application program interface to establish a connection with an application on a remote machine. Each request is passed as a message to the appropriate service provider process via the messaging routines. It is then processed by the service provider process in accordance with the protocol elements for that request. Within the service provider process, connections to remote applications and the connection to the application program are handled by sockets. The socket concept is identical to that used in the Berkeley UNIX* 4.2BSD operating system. We will refer to the socket used for a connection to the application program as the user socket, and the sockets used for connections to remote applications as networking sockets.

Service Provider Process. The service provider process, or SPP as it will also be referred to from here on, is an application entity in OSI terminology. The modules in an SPP implement a particular network service.

- Control. This is the main controlling module in an SPP. Its function is to wait in a loop and use the HP-UX select call to query for events on the SPP's sockets. When an event is detected on the user socket, the control module invokes the application entity coordinator. When an event is detected on a networking socket, the control module invokes the presentation services. Based on where an event occurs, the application entity coordinator or the presentation services will process the event, and if necessary, invoke other SPP modules.
- Messaging Routines. Like its counterpart in the user process, this module implements services for message based communication between the application program and the SPP.
- Connection Management. This module performs core functions associated with SPP initialization and connection setup and termination. For connection setup, the connection manager performs a directory lookup to obtain the address of the remote application. Once this is done, it coordinates with the ASEs to obtain their protocol data unit information, and then invokes the presentation services to send the connection request. Should the connection request fail, the connection manager allows the application program to retry the connection setup periodically. The connection manager also allows the application program to listen for connection requests initiated by a remote application The connection manager also coordinates with the ASEs to terminate the connection, obtaining protocol data unit information from each ASE for the purpose.
- Application Service Element (ASE). The SPP uses one or more network protocols, or ASEs, to implement a particular network service. The service provided by an ASE and the interface used to access this service are specified by the OSI standards. For example, MMS, X.500, FTAM, ROSE, and ACSE are ASEs with corresponding OSI specifications.
- Application Entity Coordinator. This module contains

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

the value added functions that go beyond the functions specified by the OSI standard. For example, the HP MAP 3.0 FTAM application interface specifies a primitive to copy a file. The application entity coordinator provides this high-level service by using several lower-level primitives specified by the OSI FTAM service specification. The application entity coordinator also manages the operation of the ASEs.

- Presentation Services. The presentation services provide the services specified for the OSI presentation layer[3,4] and some additional features, such as negotiating and handling presentation contexts, PDU (protocol data unit) encoding and decoding, and PDU dispatching and data streaming.
- Utilities. This module (not shown in Fig. 2) provides services to other SPP modules such as tracking memory use, managing shared memory, and signal handling.

Connection and PDU Transmission

When a connection is set up, each ASE defines the possible types of data that it might send. A set of such types is called the presentation context. The presentation services negotiate these contexts with the remote application on behalf of the ASEs. Once the negotiation is complete, the presentation services ensure that any data sent or received is within the set of contexts negotiated.

When an ASE sends data to a remote application, the data needs to be encoded into PDUs. Similarly, when a PDU arrives from a remote application, it must be decoded and then dispatched to the ASE to whose context the data belongs. These functions constitute the encoding, decoding, and PDU dispatching functions. If a PDU that is to be sent or received is very large, it may not be possible to send or receive it all at once and it will have to be streamed. The presentation services fragment such PDUs before they are sent and reassemble the fragments as they are received.

Example Scenario

The following example shows how the upper layer architectural components interact with each other. In this example only one ASE is used. We assume that before this scenario begins, the end user has started the user process, which in turn has forked a service provider process. At the point where this scenario begins, the user's application program wants to request a connection to a remote application. Fig. 3 shows the sequence of events that takes place with this request.

1. The application requests a connection at the application program interface with application-specific parameters.
2. An application program interface primitive calls the event manager to register an event.
3. The application program interface calls the user process connection manager to initiate the connection.
4. The connection manager sends a message to the SPP to initiate the connection.
5. A select call posted by the control module returns. Remember that the control module is a continuous loop until it detects an event.
6. Having detected an event at a user socket, the control module invokes the application entity coordinator to handle it.
7. The application entity coordinator uses the messaging interface to read the message sent by the user process.
8. Seeing that the message is a connection request, the application entity requests the SPP connection manager to set up a connection.
9. The connection manager calls each ASE (only one is shown) to obtain the PDUs that need to be sent on the connection request.
10. The ASE defines its presentation contexts and then encodes its PDU by invoking the presentation services. When these activities are done the ASE returns to the connection manager.
11. Having obtained each ASE's protocol data unit, the connection manager requests the presentation services to send the connect request.
12. The presentation services send the connection request by using the socket connect primitives to send the protocol data unit. This results in a new socket being created corresponding to the new connection.
13. The presentation services inform the control module about the new socket, which must be added to the sockets that the control module is already monitoring. When this is done, return is made to the connection manager, then to the application entity coordinator, and finally back to the control module.
14. Back in its continuous loop, the control module again issues a select call, waiting for the connect confirmation to arrive from the remote application.

Encoding and Decoding

Protocol data units for ASEs and the presentation layer are specified in the OSI standards in a format called Abstract Syntax Notation One (or ASN.1).[5,6,7] It is a BNF-like notation for specifying the structure and contents of

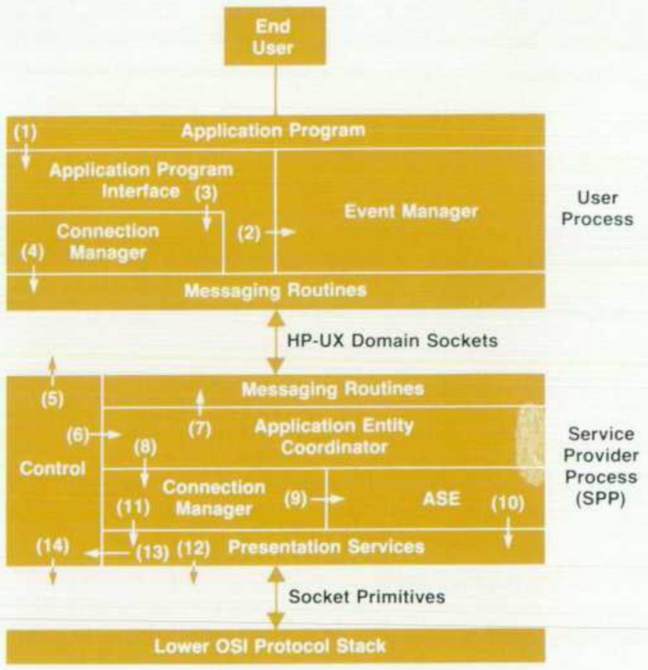


Fig. 3. *The steps involved and the interaction between upper layer architecture components when an application initiates a connection to a remote application.*

PDUs. Included in the notation are several primitive types, such as Boolean, integer, octet string,* and so on. Also defined are the constructed types, which are composed of primitive and other constructed types. This process can be applied recursively to define arbitrarily complex types. For instance, a PDU using the constructed type Sequence can be defined as:

```
AssociatePDU        ::=     SEQUENCE {
      calledAETitle          ApplicationTitle,
      callingAETitle         ApplicationTitle,
      protocolVersion        INTEGER,
      userData               [UNIVERSAL 6]
                             IMPLICIT OCTET STRING}
Application Title :: = OCTET STRING
```

ASEs need to manipulate the PDUs they send or receive. The PDUs can be complex, and writing code to manipulate an ASN.1-encoded PDU can be a formidable task. Locating a particular field within a PDU involves scanning the PDU to find the desired field, the precise position of which depends on the length and the number of previous fields. For our implementation, this task is aided by the use of an ASN.1 compiler. This approach takes advantage of the fact that the ASN.1 notation is a formal language and is well-suited for automated processing.

The ASN.1 specification for an ASE is input directly to a compiler that generates a set of C data structures and encoding and decoding routines (C functions). The C data structures are included in the ASE code. PDU data is stored in these data structures before encoding or after decoding. Encoded PDUs are placed in a chain of data buffers. The encoding and decoding routines are linked into the ASE by the presentation services. Also linked in are some generic functions called the run-time library, which are used to manipulate the primitive types. When the ASEs invoke the presentation services to encode and decode their PDUs, the presentation services determine the correct routine to use based on the presentation context. Fig. 4 illustrates this facility.

Data Handling

We can now take a closer look at how data exchange between two connected application programs is handled.

*A multiple of eight bits.

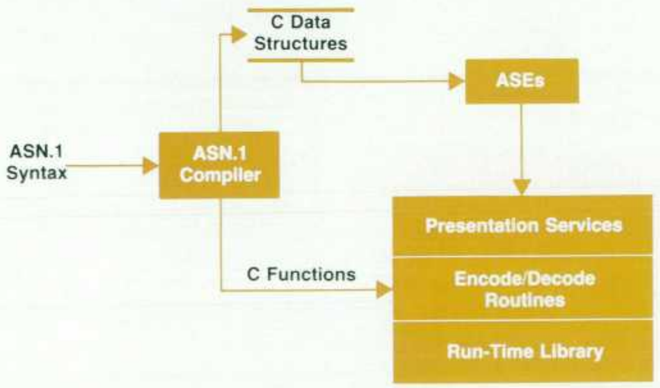


Fig. 4. *The ASN.1 encoding and decoding facility.*

For outbound data the user sends the data to the SPP in a message that is read by the application entity and handed to the ASE. At this point the data is in C data structures and not yet encoded. The ASE then invokes the presentation services to encode the data. This results in a buffer chain containing the application PDU. The ASE hands this buffer chain to the presentation services to be sent to the remote application. Just before sending the PDU, the presentation services encode this data once more, this time to add their own protocol control information. They then invoke the socket primitives to send the PDU.

For the inbound case, the scenario is reversed. When an encoded PDU is received, the presentation services do a first-level decode to remove presentation protocol control information. Since each PDU can contain data with different presentation contexts, this process also identifies the presentation contexts of the PDU. This involves establishing references (pointers) to the locations in the PDU buffer chain for each different presentation context, and keeping a reference count of the number of contexts. Once this is done, the presentation services send the PDU to the appropriate ASEs. Each ASE decodes its portion of the PDU into C structures and processes the data. As each ASE finishes processing its data, the reference count is decremented, and when the count reaches zero, the presentation services release the entire PDU buffer chain.

Conclusion

The upper layer architecture has been used to develop three OSI networking services in the HP-UX 7.0 release of the HP MAP 3.0 product—FTAM, MMS, and X.500. By identifying architectural components and internally standardizing their interfaces, developers can use this as the platform for OSI development. The core components can be leveraged for future services development, allowing developers to focus on the specific ASEs and their application interfaces.

Acknowledgments

Many people contributed to the development of the upper layer architecture for the HP MAP 3.0 product. Wan-Yen Hsu, Doug Gregory, Suhas Badve, and Jeff Mendonca contributed to it during the early specification stages. Wan-Yen and Doug also went on to develop portions of the code. Brian O'Keefe made several useful enhancements and Jeff Kunz did a comprehensive performance characterization of the architecture. Thanks are also due the then project managers Jean Yao and Atul Bhatnagar for providing the environment and support for the successful completion of this project.

References

1. *Information Processing Systems - Open Systems Interconnection - Application Layer Structure*, ISO/DP 9545, ISO/TC97/SC21/N1743, July, 1987. Revised November 1987.
2. *MAP 3.0 Application Interface Specification*, MAP Technical Subcommittee, 1988.
3. *Information Processing Systems - Open Systems Interconnection - Connection Oriented Presentation Service Specification*, ISO 8822: 1988 (ISO/IEC JTC1/SC21 N2335).
4. *Information Processing Systems - Open Systems Interconnection - Connection Oriented Presentation Protocol Specification*,

ISO 8822: 1988 (ISO/IEC JTC1/SC21 N2336).
5. *Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)*, ISO 8824: 1987 (E).
6. *Information Processing Systems - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*, ISO 8825: 1987 (E).
7. K. K. Banker and M. A. Ellis, "The Upper Layers of the HP OSI Express Card Stack," *Hewlett-Packard Journal*, Vol. 41, no. 1, February 1990, pp. 28-36.

Directory Services in the HP MAP 3.0 Environment

To provide a standardized implementation of a directory service for locating resources in the HP MAP 3.0 environment, the ISO X.500 directory standard is used.

by Beth E. Cooke, Colleen S. Fettig, Paul B. Koski, Darrell O. Swope, and Roy M. Vandoorn

AS COMPUTER NETWORKS BECOME larger and more complex, applications are being distributed throughout a network for maximum efficiency in the use of the computer resources. Determining the location of these applications is critical to the success of the network from the user's perspective. A facility called a directory service can be used to determine the location of applications and other resources in a network.

This article describes the general characteristics of a directory service and gives an overview of the ISO/CCITT X.500 model, the directory service in HP MAP 3.0.

Directories

Directories have been around for many years, from the telephone directory in the home to the card catalog at the local library. With the size and complexity of the modern computer network, automated directories are being introduced to perform a function in the network that is similar to the role a telephone directory plays in the home.

Let's consider the world of manufacturing, where the typical operation uses a number of different and isolated computer systems. There is a mainframe application that manages the facility's entire inventory data base, a group of engineering workstations for product design, robots for controlling systems on the manufacturing floor, and personal computers that are used as terminals or low-level workstations. The workstations, controllers, and personal computers can be connected to other computers of the same type, but they are usually isolated from the other systems. For instance, the robot cell controllers may not communicate with each other, the CAD workstations only talk to each other, and personal computers are often completely isolated. These systems are classic examples of islands of information.

To implement just-in-time manufacturing, which is aimed at dramatically increasing manufacturing productivity and flexibility, these islands have to be coalesced into a unified system. More specifically, floor cell controllers should be able to update the inventory data base as they use parts to build the products, and the design engineers at the CAD workstations should be able to access the inventory data base to determine the immediate availability of components that they may want to specify in their designs. In fact, it may be necessary for them to be able to access their supplier's inventory data base, which may be remote and controlled by a different company.

In forming this complete system, there are many technical problems that arise, only some of which a directory service can address. One of the fundamental difficulties is that the new system will be very large and complex, and therefore difficult for anyone to understand completely. Another attribute of such a large system is that it will be dynamic—new components will be added to it and taken away almost daily. Applications and subnetworks will enter and leave the network. Paths between portions of the network will also change frequently. Additionally, the addresses, availability, and physical locations of individual applications and network resources may change. Thus, the changes to the state of the network can be viewed as asynchronous events and applications cannot be required to have prior knowledge of them. Finally, the expected useful lifetime of an individual application can be relatively long, and it will interact and communicate with other applications more often than it will change its availability, address, or other associated information.

Directory Service Characteristics

A directory service can help isolate applications from changes in the network by providing a globally accessible repository of information about the different objects in the network. In directory service terminology objects are items such as printers, servers, workstations, files, directories, gateways, users, and processes (applications). As new applications enter the system they can be registered in the directory so that other applications can retrieve information about them, such as their current address and/or a description of the services that they provide.

For the user's and application's benefit this large complex system should be as transparent as possible. One of the primary goals in designing a transparent network is to provide users with a simple and intuitive view of the network. The directory service can help with this problem by providing a facility through which network objects can be named in an abstract and consistent manner. In general, it is much easier for users to use abstract names than descriptions. For instance, individuals are addressed by a name like "Joe," rather than a description like, "tall 12-year-old with dark hair that lives across the street." This naming facility also helps with the second general problem—changes. If Joe moves, or gets older, he is still going to be Joe.

Although users and hence applications generally prefer to identify objects with relatively simple, English-like names, processes on a network might need to have more information about an object. If the object is a file, they may need to know which machine it resides on, the directory it resides in, what type of file it is, and the filename. This conflicts with the type of simple naming that users would prefer. One goal of a directory service is to provide a naming mechanism for users that does not require them to provide system dependent knowledge about objects.

Directory Service Functions

As mentioned earlier, a version of a directory service that has been used for years is the telephone directory. Humans are not particularly fond of having to memorize phone numbers to use the system. However, humans are fairly good at remembering names or places of business. In the United States a telephone directory is divided into two sections: the white pages and the yellow pages. The white pages are used to retrieve information, such as a person's phone number, address, and middle initial. In terms of the directory service, this is known as name-to-property binding. A user-friendly name can be used as a key to retrieve information associated with an object or properties of an object, such as its network address.

The yellow pages section is used to find the information needed to communicate with some set of people, normally businesses. When the process is examined a little closer, what actually happens is that a property, say Hewlett-Packard dealership, is used to restrict retrieval to only those businesses that are Hewlett-Packard dealers. In directory service nomenclature this is property-to-name binding. In a network environment, a client may wish to see a list of all printers in the building that support a particular font, or all of the workstations that have a certain memory configuration.

Unlike the common phone directory the directory service allows clients to use aliases or nicknames to identify an object. This is classified as providing name-to-name binding. All of the names in a particular set or equivalence class map to the same object. For example, the mail address /USA/HP/IND/Cupertino/Fred, could simply be addressed as Fred and the directory service would take care of mapping Fred to the full mail address.

Similarly, the directory service provides a name-to-list-of-names binding service that returns a list of objects given a single object name. This is a facility that can be used for mailing to groups of users (distribution list), or in defining levels of access to network resources.

Inherent in the requirements for the four types of binding (name-to-pointer, pointer-to-name, name-to-name, and name-to-list) is the responsibility for storing information. The data store contains a set of names and a set of properties that are associated with these names. The combination of a name and a set of properties defines an object in the data store. The directory service also contains the ability to differentiate, in a simple form, between different classes of data. For instance, it can distinguish between the logon name "foo" and the presentation level address "foo." It can also tell that the printer named "Tom" is different from the person with the same first name. In a very fundamental sense, the directory service uses some semantic information as well as syntactical data.

Named objects must be organized in a hierarchical tree structure that enables unambiguous, and possibly context sensitive naming. The purpose of this tree is to provide network administrators and users with logical groupings of the objects they are likely to reference. This tree, or directory, may be organized along some logical grouping such as an organizational structure or geographical locations.

The directory service acts as a globally available register of system information about the objects that are in the

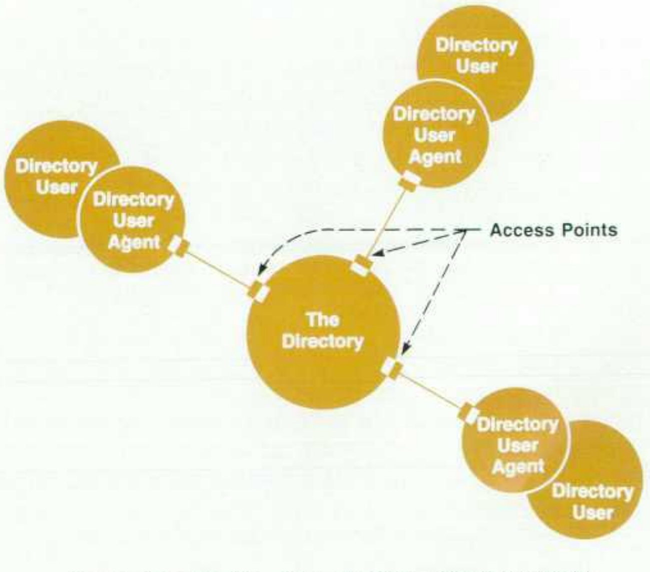


Fig. 1. *The function of the directory user agent (DUA).*

network. When requested, it retrieves specific portions of this data for clients. This implies that the directory service provides a dynamic binding capability for its clients. For the clients to rely upon the directory service, it must contain current and accurate information about the state of the system. If it does, it will serve to increase the network's overall reliability because information about network resources can be registered with one service that is responsible for sharing this information with processes on a dynamic basis. This feature gives the system a self-configuring capability that would not be feasible if the information had to be statically given to each client that might need the data. This way, changes to the network have a minimal impact on network operations and applications can view the network as a relatively stable and abstract system.

Specifically, a directory service is an administrative name management facility for a transparent, distributed processing system. It provides a global naming service and manages information associated with each named network object. With a directory service, users and applications can refer to objects in an intuitive manner. Administrators can limit access to and manage information on these objects through facilities provided in the directory service.

A Directory Service Standard

It is possible to conceive of different ways to implement a directory service, and with multiple vendors implementing directory services it is unlikely that these implementations will be able to communicate or interoperate unless there is a directory standard. Therefore, to ensure interoperability between the different implementations of a directory service, two international standards organizations—ISO (International Organization for Standardization) and CCITT (International Telegraph and Telephone Consultative Committee)—began a collaborative effort to standardize directory services. This work resulted in the CCITT 1988 X.500 standard and ISO IS 9594. These two standards are identical. For simplicity, the directory standard will be referred to as X.500 throughout the remainder of this article.

What is X.500?

To provide a standardized directory, three areas must be defined: the architecture, the information, and the operations. X.500 defines all three. The architecture, called the directory model, defines two agents: the directory user agent and the directory system agent. The directory itself is defined as a repository of information about objects. The information, called the informational model, defines the storage of the different types of data and the rules to which it must conform. The operations, or directory services, are the operations that can be performed on the directory. Together, these two models and the operations form the model for the X.500 directory standard.

X.500 Directory Model

There are four objects in the directory model: the directory user, the directory, the directory user agent (DUA) and the directory system agent (DSA). The directory user can be either a user or an application. For a directory user to

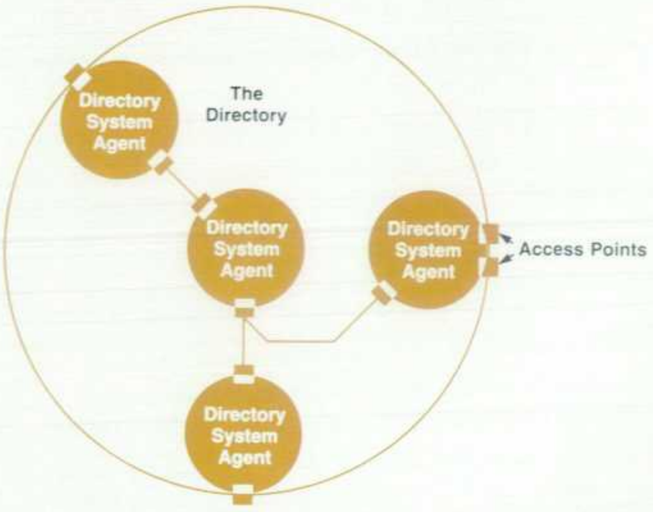


Fig. 2. *The function of the directory system agent (DSA) in the directory model.*

operate on directory information, a connection must be established with the directory (see Fig. 1). This is the job of the DUA. The DUA represents the user to the directory. Interaction between the DUA and the directory occurs at an access point. The access point is the conceptual point at which an abstract service is obtained.

The directory is made up of one or more DSAs (see Fig. 2). The role of a DSA is to communicate with the DUA representing the user, either directly or indirectly via other DSAs, and to provide access to the directory information it is responsible for storing. When the directory consists of a single directory system agent, all directory information is contained within it. When the directory consists of multiple DSAs, directory information is distributed between the DSAs, which must share the information through a network. This is referred to as a distributed directory. The data contained within the directory is collectively known as the directory information base.

The overall X.500 model is shown in Fig. 3. Here, multiple directory users can access the directory through their own DUA. Users can reside on the same system, or across multiple systems. The DUA does not have to reside on the same system as the DSA with which it communicates. The directory itself may be spread across multiple systems and information is shared through interfaces between the DSAs. Each DUA communicates user requests to and receives results from the directory through a DSA's access point. The information received may reside in the DSA that the DUA is accessing, or in another DSA.

X.500 Informational Model

The informational model defines four areas: directory information base, directory entries, names, and directory schema.

Directory Information Base. The directory information base is the collection of information stored in the directory. This information represents objects. An object is something that is identifiable (can be named) and is of interest. When an object is stored in the directory it is known as a directory entry. For entries to be found after they are stored in the

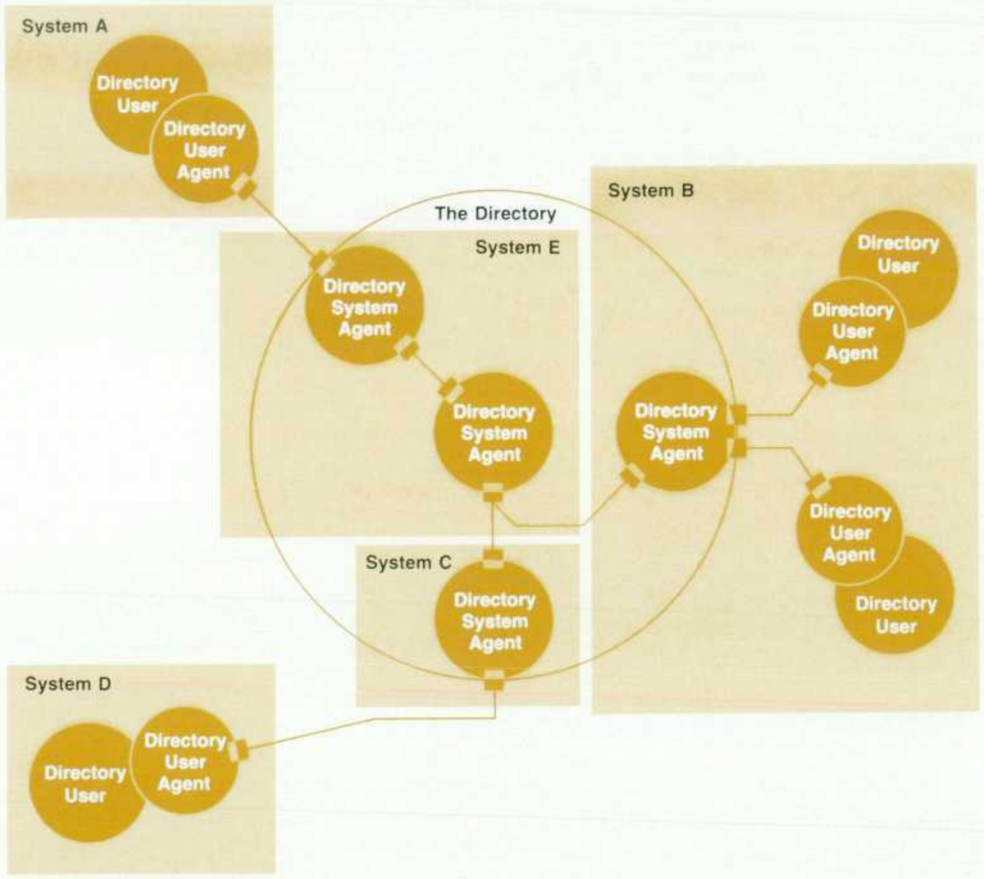


Fig. 3. *The overall X.500 model.*

directory, the entries are logically arranged in a tree structure with each entry representing a vertex (see Fig. 4). This structure shows the parent/child relationship between objects in the directory.

Directory Entries. The information stored in an individual entry is divided into attributes. Examples of attributes are phone number, surname, and country. Many of the directory entries represent similar objects that are grouped into an object class (e.g., software applications, devices, countries). The entry's object class determines the attributes an entry may and must contain. Each attribute consists of an attribute type and an attribute value. The attribute type identifies the attribute as a predefined type, such as organization name, locality, or presentation address.

Names. To identify a particular entry, the entry requires a name. This name is referred to as either a distinguished name or a directory distinguished name. Each distinguished name is unambiguous and represents just one entry in the directory. In the X.500 standard, the distinguished name is a path down the tree to the entry. Therefore, a particular distinguished name consists of the parent's name plus the information on how to proceed down to the last branch. To identify the path to the last branch in the directory tree, each entry has one or more naming attributes. Each naming attribute has exactly one attribute value called the distinguished attribute value, which is used for naming (see Fig. 5). Each child is uniquely identifiable and distinguishable from its siblings by its naming attributes.

Directory Schema. The directory schema is the set of rules, definitions, and constraints that define the conformance requirements for all directory entries and their related information. These requirements govern four areas: the tree structure of the directory information base, the entries, the attributes, and the attribute values.

The rules governing the tree structure of the directory information base define which entry can be the child of another entry, as well as which attributes can be used to distinguish one entry from another (naming attributes). The rules governing the parent/child relationship are based on object classes and are necessary for maintaining a level of organization in the directory information base. For example, it would make no sense for an entry defining a person to be the child of an entry defining an application. It does make sense, however, for the person's entry to be the child of an entry defining an organization.

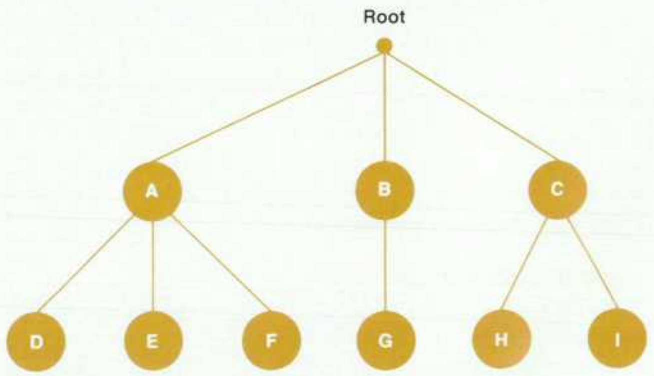


Fig. 4. *The directory information base structure.*

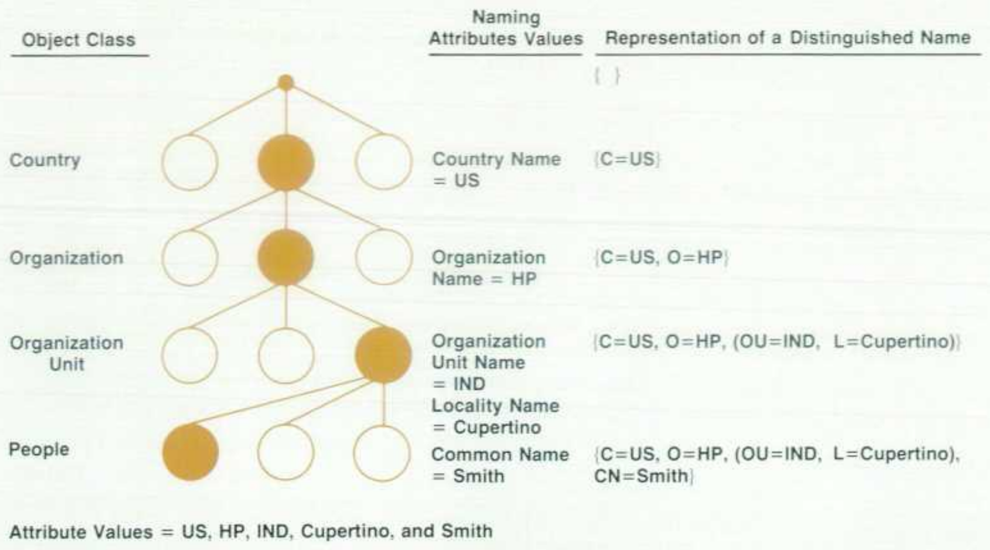

Object Class
Naming Attributes Values
Representation of a Distinguished Name
{ }
Country
Country Name = US
{C=US}
Organization
Organization Name = HP
{C=US, O=HP}
Organization Unit
Organization Unit Name = IND
Locality Name = Cupertino
{C=US, O=HP, (OU=IND, L=Cupertino)}
People
Common Name = Smith
{C=US, O=HP, (OU=IND, L=Cupertino), CN=Smith}
Attribute Values = US, HP, IND, Cupertino, and Smith

Fig. 5. *The structure of a distinguished name.*

The rules governing the naming attributes are necessary to allow a directory user to estimate easily the name of an entry. For example, the naming attribute for an entry of the object class organization would be the organization name. Using an attribute such as a locality name would make naming ambiguous.

Each entry must conform to rules inherited from its object class. As discussed above, the object class dictates when attributes may and must be part of the entry. Other than those dictated, no additional attributes can be included in the entry.

Each attribute must conform to a set of rules defining its identifier, the syntax of its value, and whether the attribute can contain one or more values. These rules allow the directory user to interpret the data retrieved from the directory. For example, the value of a presentation address attribute is defined as a single occurrence of a specialized structure, whereas the value of an organization name is defined as one or more occurrences of a character string.

The final set of rules governs the syntax of attribute values. An attribute value's syntax identifies the rules used for matching a value of a particular type. For example, it is necessary to use a matching rule during a search, when the user has specified some properties to be matched.

X.500 Directory Services

Three types of operations are defined in the X.500 standard:

- User authorization: bind, unbind
- Data retrieval: read, compare, list, and search
- Data maintenance: add, modify, rename, and remove

The user authorization operations can be thought of as logging into and logging out of the directory. The bind operation is used to initiate a directory session when the user provides identification. The unbind operation is used to terminate the directory session.

The data retrieval operations are used to obtain information from the directory. The most common operation is read. The read operation is used to perform name-to-property, name-to-name, and name-to-list bindings, and returns the information stored in the entry. The compare operation is used to verify the value of an attribute. The list and search operations are used for property-to-name bindings, and returns entries that meet certain criteria.

The data maintenance operations are used to add, modify, rename, and delete the information in the directory.

HP MAP 3.0 Directory Service

The directory's name-to-property binding operation is used in HP MAP 3.0 by two application level services called File Transfer, Access, and Management (FTAM) and Manufacturing Message Specification (MMS). These two applications are described in more detail in the articles on pages 24 and 31.

The FTAM standard specifies a virtual file store, a set of services to manipulate that file store, and the protocols that define how to use the services. If a user program wishes to use any FTAM service, it sends a request to the FTAM initiator. The FTAM initiator then sends the user program's request to an FTAM responder that will service it. Fig. 6 shows how this process works.

Directory services are used by the FTAM initiator when it needs to find the network address of the FTAM responder that will service the request. The FTAM initiator sends a distinguished name to the directory via a DUA. The DUA will return the corresponding network address of the FTAM responder to the FTAM initiator. After the initiator has the address, it can establish a connection with the responder.

The Manufacturing Message Specification (MMS), which is the other HP MAP 3.0 application that uses directory services, provides data handling between a variety of machines in a manufacturing environment. The services of MMS are provided to a client by an MMS service provider process (see Fig. 7). When a client issues a request, MMS takes the request and sends it to the appropriate machine to service the request. When the client MMS service provider needs to establish a connection to the machine (server) that will service the request, the directory service is used to get the network address of the server. Once the client has the network address, it can then establish a connection to the desired server.

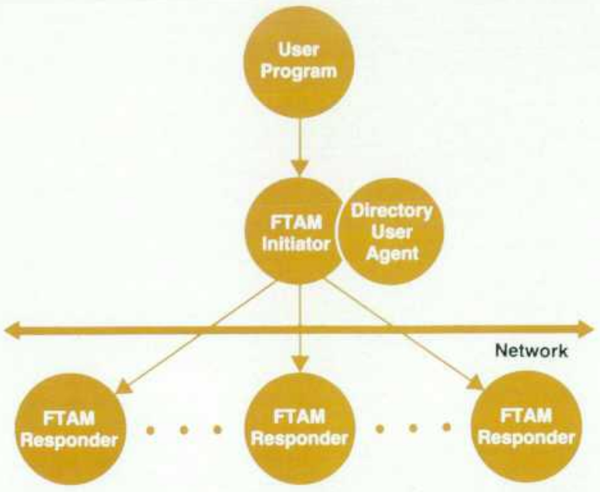


Fig. 6. *Process of communication in FTAM.*

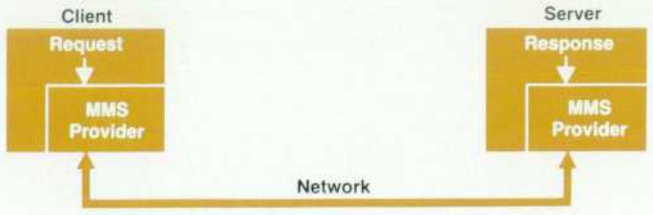


Fig. 7. *Process of communication in MMS.*

The architecture for the directory service for HP MAP 3.0 is an X.500 implementation. Within the flexibility of the X.500 standard, certain liberties were taken for performance reasons and ease of implementation.

Design Decisions

A centralized directory information base was selected for HP MAP 3.0 because it was easier to implement and it satisfied our schedule constraints. With all the directory information centralized, it is only necessary to have a single DSA. This eliminated the need for communication between DSAs and the need to implement the X.500 directory system protocol, which defines the protocol between DSAs.

The DUA represents the user to the directory. There are two ways to implement the DUA. One way is to have one DUA per user and the other is to have one DUA per system. The latter alternative was chosen because of the typical use of X.500 in the HP MAP 3.0 environment. Each process would normally obtain a single address from the directory and then perform its task. It was determined that creating (forking) a process each time an address was needed would be too expensive in terms of performance. Linking the entire directory user agent into the user process was not done either because of process size concerns.

The final decision was to combine the DUA and the DSA into one process on the machine containing the directory information base. There needed to be a DSA daemon process to service directory requests from DUAs on other systems. At a minimum, the DSA daemon needs the ability to receive local interprocess messages for administration operations. It was not a large extension to fold the DUA agent functionality into the DSA process.

Architecture

An overview of the architecture for the directory service in HP MAP 3.0 is shown Fig. 8. There are three processes that make up the directory services: the user process, the DUA, and the coresident directory user agent/directory sys-

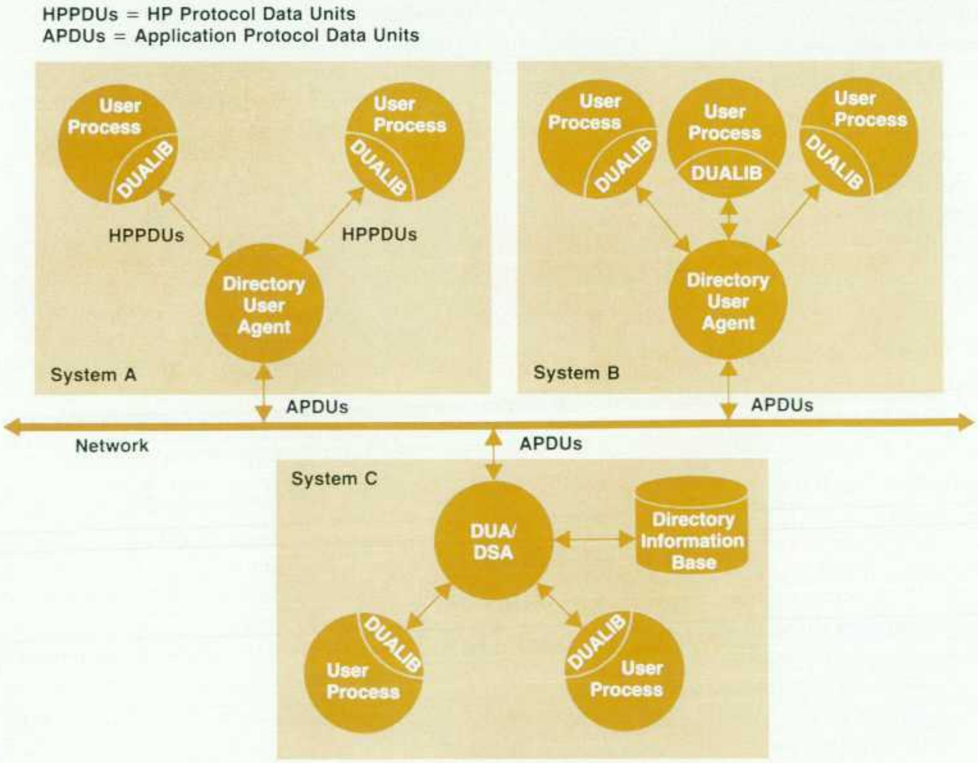


Fig. 8. *X.500 high-level architecture in HP MAP 3.0.*

tem agent (DUA/DSA). The user process is the process that needs to use directory data. This process links in the DUA library (DUALIB) to gain access to functions that allow a user process to access the directory. The DUA resides on the machines that do not contain the directory information base. The DUA receives the requested directory operation from the user process and forwards the request to the machine that contains the directory information base. The DUA/DSA process is located on this machine. This process services the directory operations and receives requests from DUAs on other machines and from user processes on the local machine.

User Process. The architecture of a user process is shown in Fig. 9. The modules that make up the user process are the user code, DUALIB, and messaging. The user code is the application that operates on data stored in the directory. This application accesses the directory through an application program interface. The functions called through the application program interface are all located in DUALIB. DUALIB performs input parameter checking, encoding of the requested directory operation, sending the directory operation to the DUA, receiving the results from the DUA, decoding the results, and returning them to the user code. The final module in the user process is messaging. Messaging is an interprocess communications module that is built on top of sockets, which are identical to the sockets used in the Berkeley UNIX 4.2BSD operating system. Communication between the user process and the directory user agent (or DUA/DSA process) is accomplished by using proprietary protocol data units known as HP protocol data units (HPPDUs).

There are three different types of HPPDUs: initiator, result, and error. A request for a directory operation is sent from the user process to the DUA as an initiator HPPDU. The DUA decodes this HPPDU and takes the appropriate actions. Once the directory operation has completed, either a result or an error HPPDU is returned to the user process. DUALIB decodes this HPPDU and returns the result or error to the user code. The decoding of the HPPDU is done with a lex/yacc-generated parser.

The interface between the user process and the DUA is not an X.500 defined interface. Therefore, it was decided to use a proprietary PDU (protocol data unit) at this interface rather than leveraging one of the X.500 standard defined PDUs for initial troubleshooting and supportability. The HPPDUs are human readable, which aided in the original troubleshooting of the product and is expected to be a benefit during the support cycle of the product. In addition to communicating the directory operations, the HPPDUs are also used for administration operations. For example, during initialization an HPPDU is sent to the DUA with configuration information. When the DUA daemon is to be shut down, a shutdown HPPDU is sent. This provides consistent access to the DUA daemon processes.

Directory User Agent Process. The structure of the DUA process is shown in Fig. 10. The DUA process is divided into seven modules: messaging, control, client coordinator, connection management, ROSE (Remote Operations Service Entity), ASEs (application service elements), and presentation services.

- Messaging Module. This is the same module that exists in the user process and is used for communication between the directory user agent and the user process.
- Control Module. This is the main module that reacts to messages being received either by the messaging or presentation services modules.
- Client Coordinator Module. This is the controlling and coordinating module for X.500-specific tasks. When a message is received from the user process, the control module calls the client coordinator. It is the client coordinator that determines what action the user process wishes to perform. The directory operation comes in as an HPPDU, which the client coordinator parses using an analogous lex/yacc-generated parser. Depending on the operation, the appropriate modules are called. The client coordinator maintains all the connection information. Since the DUA communicates asynchronously with the DSA, the client coordinator needs to remember the operation requested by each process. When an operation completes, the client coordinator encodes the result and sends it back to the appropriate user process.

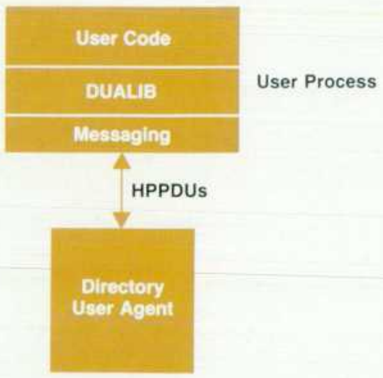


Fig. 9. *User process structure.*

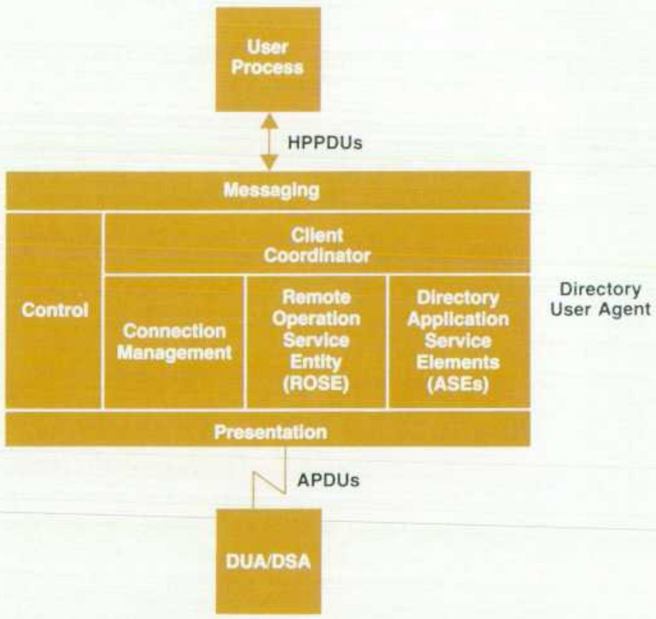


Fig. 10. *Directory user agent process.*

- Connection Management Module. This is the HP MAP 3.0 implementation of the Association Control Service Element (ACSE).[1] This module handles the setting up and breaking down of remote connections between application entities.
- Application Service Elements (ASEs). The DUA ASEs take operation arguments stored in C data structures and create an octet stream called an application protocol data unit (APDU). Likewise, the ASEs convert returning APDUs into C data structures that contain result or error information related to the requested operation. There are two ASEs in the HP MAP 3.0 implementation of the DUA, one for read operations and one for add and remove operations. In addition, a portion of the ASEs handles directory bind operations.

  The process of creating APDUs from C data structures and the reverse is accomplished with ASN.1 encoding.[2] ASN.1 (Abstract Syntax Notation One) is an OSI standard notation for arbitrary data structures. This notation is used like a programming language in the X.500 standard to specify all of the arguments, results, and error information used by any X.500 implementation. An accompanying set of encoding rules called BER (Basic Encoding Rules) determines how data structures described in ASN.1 map into a flat stream of octets. See the article on page 11 for more information about ASN.1 in HP MAP 3.0.
- Remote Operations Service Entity (ROSE). ROSE is a generic OSI service that allows applications to make remote procedure calls to applications on other systems. ROSE also allows the remote applications to return result or error information in response to the requested operation. The X.500 DUA uses ROSE to execute all operations on a remote DSA. Essentially, the ASEs tell ROSE which X.500 operation to invoke on the remote DSA, and gives arguments in the form of an ASN.1-encoded APDU. ROSE then asynchronously invokes the operation on the remote server. When results or error information are returned from the DSA, ROSE calls the appropriate ASE and gives it the response APDU.
- Presentation Services Module. This module is an interface to the presentation layer of the OSI stack. This is the module that the ASEs and ROSE use to encode their APDUs. At initialization, both the ASEs and ROSE register with the presentation services their ASN.1 encoding and decoding routines. When the ASEs or ROSE are ready to access these routines, they provide a symbolic name for the entry point and pass the presentation services module the application data to be encoded or decoded. Encoded data is kept in the presentation services data structures to facilitate its transmission.

DUA/DSA Process. As mentioned earlier, the decision was made to combine the DUA and DSA into a single process for the HP MAP 3.0 implementation. This eliminated the overhead of communicating between processes. The structure of this process is shown in Fig. 11.

The modules that make up the DUA/DSA process include: messaging, control, local coordinator, remote coordinator, data base access module, connection management, ROSE, ASEs, and presentation services. A number of these modules are exactly the same as in the DUA process. They are messaging, control, connection management, ROSE, and presentation services. The ASEs decode the APDUs using the same ASN.1 mechanism described earlier. In addition, any results or errors that are sent to remote DUAs in response to operations are encoded by the ASEs in the DSA process and decoded by the ASEs in the DUA. Note that user requests made on the DSA's machine will not reach the ASEs. Only remote operations that make use of the OSI stack require the interaction of the ASEs and ROSE.

- Local Coordinator Module. The local coordinator module is very similar to the client coordinator module in the DUA process. However, instead of interfacing to the ASEs, the local coordinator interfaces directly with the data base access module. Remember that in this implementation the directory information tree is centralized, making it unnecessary to go to the network to perform the directory operation. The local coordinator is called by the control module when the messaging module receives an HPPDU from one of the local user processes.
- Remote Coordinator Module. The remote coordinator module controls and coordinates remote directory operations. These are operations requested by a user process on a remote machine. The remote coordinator receives the operation from one of the ASEs and then calls the data base access module to operate on the information stored in the directory information base.
- Data Base Access Module. The data base access module interfaces with the physical data base that is used to store the directory information base. All the data base operations have been localized in this module of the DUA/DSA process. This module is responsible for locating the entry of interest, accessing and manipulating entries, ensuring that the schema rules are met, and checking the input parameters of the requested directory operations.

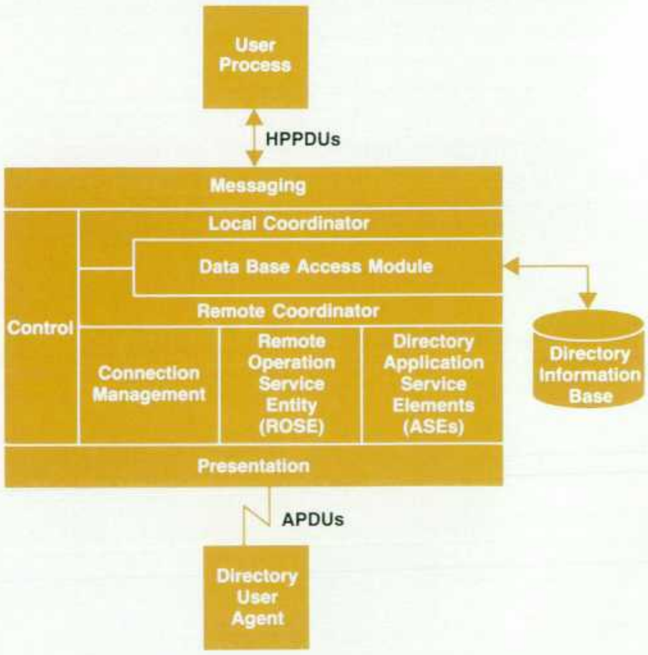


Fig. 11. *Directory user agent/directory system agent (DUA/DSA) process.*

The Data Base

As can be seen from the role of the data base access module, the data base stores more than just the directory information base. Recall that the directory information base is defined as the complete set of information to which the directory provides access. In addition to data for the directory information base, the data base also contains the schema rules and proprietary information such as user capability levels.

The data base is implemented as a relational data base with an SQL (Structured Query Language) interface. Data stored in a relational data base is organized into tables. The directory information base is divided between three tables. The first table stores the directory information tree. This is done by representing the parent-child relationships of the entries. The other two tables store the data. The data is divided between two tables to separate the naming and the nonnaming attributes. This was done to optimize the speed of name resolution.

Retrieving information from an entry involves two steps. First, the distinguished name of the entry is used to navigate through the tree to the entry desired. The table storing the directory information tree, that is, the table that stores the parent-child relationships, is the table used for this navigation. These relationships are used to step through each level of the tree until the specific entry is located. The table storing the naming attributes is used to verify the relationship at each level of the tree.

Since traversing the tree to locate the entry specified by the distinguished name is time-consuming, two techniques are used to optimize this task. The first is the use of a key to limit the searches performed at each level of the tree. At each entry in the path, which is represented by the distinguished name, a search is made for all children of that entry until the child that matches the given naming attributes is located. This has been simplified by creating a key from these naming attributes. This key is stored in the table with other directory information tree information. Rather than searching all the children of a given entry, this scheme limits the search to only those children whose key matches the key derived from the given naming attributes. With this optimization, the searches required at each level of the tree are limited to a subset of the children, which is quite often a single child.

The second optimization actually bypasses the tree traversal by using a cache of commonly used distinguished names. Each distinguished name has a unique identifier associated with it. This identifier is assigned by the data base and is used for internal searches. The result of locating an entry through tree traversal is this identifier. The cache simply maps the distinguished name to this identifier without the overhead required by traversing the tree. In addition, the attribute that contains the network address is also stored in the cache. This optimizes the most commonly used function, the retrieval of an address for a given entry.

Once the entry specified by an operation is located, the operation is then performed on the entry. In the case of a read operation, the requested attributes are retrieved from another table that holds all entry information. The data base access module then returns the requested attributes to the coordinator module and these attributes are ultimately returned to the user process that made the original request.

Conclusion

As applications become more distributed, directory services are going to play an ever increasing role. In HP MAP 3.0, the directory is used to determine the network addresses of applications. HP MAP 3.0 is the first user of this implementation of the X.500 directory.

Acknowledgments

The authors would like the recognize the efforts of the other members of the X.500 product team and thank them for their contributions in making directory services for HP MAP 3.0 a reality. We especially wish to thank Kevin Montgomery, not only for his ROSE efforts, but for his overall contribution to the product.

References

1. K. K. Kimball and M.A. Ellis, "The Upper Layers of the HP OSI Express Card Stack," *Hewlett-Packard Journal*, Vol. 41, no. 1, February 1990, pp. 28-36.
2. *Ibid*, pp. 32-34.

HP MAP 3.0 File Transfer, Access, and Management/800

File Transfer, Access, and Management, or FTAM, is an OSI standard that defines the framework upon which layer seven file transfer services can be built for accessing and managing files across open systems.

by Steven W. Manweiler

THE BASIC DISTRIBUTED file system capability for OSI communication is defined by the file service standard called File Transfer, Access, and Management (FTAM). FTAM defines facilities for the transfer of files between open systems and provides a framework for accessing and managing files across open systems.

While FTAM provides functionality similar to conventional file transfer services, FTAM is not strictly a file transfer service. It also provides the ability to perform functions such as reading and writing parts of remote files, and obtaining or modifying attribute information associated with a remote file. FTAM should not be thought of only as a file transfer service, but rather as a framework on which services such as file transfer can be built. In fact, a service such as a distributed file system could be built using a suitable FTAM implementation. However, the main strength of FTAM is that it is a system independent file service. Any two vendors with FTAM implementations can communicate with each other using FTAM regardless of the types of file systems supported by those vendors.

FTAM is defined in the International Standards Organization (ISO) document ISO 8571.[1] This document defines the file service, the file protocol, and a common model of a file system. However, it does not define a user interface. Various committees are currently working on defining interfaces for FTAM, or have already done so. One example of an existing programmatic interface for FTAM was defined by the North American MAP/TOP (Manufacturing Automation Protocol/Technical Office Protocol) Users Group.[2]

Another standards organization involved with FTAM is the U.S.A. National Institute of Standards and Technology (NIST).[3] Four times a year NIST hosts a workshop in which vendors meet to develop implementation agreements for existing OSI standards.

This article describes the basic concepts of FTAM, the functionality of FTAM in HP MAP 3.0, and some aspects of HP's design and development of FTAM.

Overview of FTAM

The FTAM standard (ISO 8571) defines FTAM in three parts: the virtual file store (VFS), the services, and the protocol. The virtual file store defines how to handle different files structures, the services define the functionality that can be requested by the user, and the protocol defines the order in which the services can be used.

Virtual File Store

The virtual file store, or VFS, is the most important aspect of FTAM. Because of the wide variety of file systems and file structures available on different systems, a common representation of a file system is needed to facilitate the transfer and access of files between dissimilar systems. The VFS is this common (virtual) file system. It defines a set of actions (operations) that can be performed on virtual files, a set of attributes associated with virtual files, and the structure (model) of virtual files. Each FTAM implementation is responsible for mapping the features of the VFS onto the mechanisms provided by its file system.

VFS File Actions. The actions that can be performed on VFS files are similar to those provided by most file systems. Users can create and delete a file, open and close a file, read or change the attributes of a file, read or write all or parts of a file, and erase or move to various locations in a file.

VFS File Attributes. A set of attributes is defined for each file in the VFS. These attributes are properties that, aside from the data in the file, completely describe the file. A file has basic attributes such as its name, its type, and the actions that may be applied to that file, along with attributes that specify accounting information, history, user access restrictions, security restrictions (e.g., passwords), and concurrent access restrictions.

The set of attributes supported by the VFS is large enough to have a logical mapping to any file system. In fact, most real file systems support a much smaller set of attributes than that defined by the VFS. This set of attributes contains mandatory and optional attributes. All FTAM implementations must support the set of mandatory attributes, and may choose to support an optional attribute if, for example, that attribute has a natural mapping onto that implementation's file system.

An example of a mandatory attribute is the permitted actions attribute. FTAM defines a set of actions that can be performed on an existing file. These actions include read, insert, replace, extend, erase, read attributes, change attributes, and delete file. The permitted actions attribute for a file defines a subset of these actions that can be applied to that file. Any FTAM user is capable of performing this subset of actions on the file. FTAM defines an optional attribute called access control that is used when a finer granularity of access restriction is needed for a file. For example, it allows the creator of a file to designate the users that can access the file, and by setting the permitted actions for each user, designate how each user can access the file. If this attribute is not used, any user can access the file.

Another attribute of the virtual file store worth mentioning is its file locking mechanism, which is called concurrency control. When accessing a file, an FTAM user can specify a separate lock for each file access action. For example, a user can specify a shared lock for the read action and exclusive locks for the write actions such as insert, replace, extend, and erase.

Virtual File Store File Model

The most important feature of the virtual file store is its file model. This file model specifies the file data types, the file structure, the file access operations, and the encoding of file data for transfer.

The VFS uses a very general hierarchical model to describe the structure of all files (future addenda to ISO 8571 may include the definitions of other file models such as a network model or relational data bases). In this model, a virtual file is an ordered tree of any depth. Associated with each node in the tree is an access point in the file called a file access data unit (FADU). These access points may or may not have user data associated with them. The user data is structured into groups called data units. Fig. 1 shows this file model.

It is possible to access all or part of the information stored in the tree. When accessing a specific node in the tree, all data contained in the subtree of a node is available to the user. Thus, to read a complete file, the user can either traverse the entire tree (preorder traversal is used), or the user can read all data associated with the root node of the tree in one operation.

Abstract Syntax Notation One (ASN.1) is used for specifying the types of information in a file and how this information is encoded for transfer. ASN.1 provides a common way to specify the format of the data that is to be transferred by FTAM. See the article on page 11 for more about ASN.1.

Most of the file structures of existing file systems are more restricted than that specified by the VFS file model. For this reason the VFS defines the notion of a document type. A document type defines practical file types by imposing restrictions on:

- The structure of the file
- The types of the data in the file and the amount of structuring information present in the file
- The actions that may be performed and when they may be performed
- The mechanism of encoding data for transfer.

Fig. 2 shows the structure of a document type that is called FTAM-1 in the FTAM standard. FTAM-1 models simple text files such as an HP-UX text file, which is simply a sequence of bytes without any structure. This type of file consists of one FADU that contains one data unit. The data unit contains the contents of the entire file. The FTAM-1 document type restricts the file's contents to text (i.e., the file may not contain integers or floating-point numbers). The document type definition also places restrictions on how files may be transferred and accessed. For example, data in FTAM-1 files can only be written at the end of the file, and it is not possible to read parts of an FTAM-1 file, that is, the entire file must be read.

Fig. 3 shows another document type called FTAM-2. This document type models record-oriented files. In this model, each record is represented by a FADU and each FADU contains one data unit that represents the contents of the record. In addition, a root node without an associated data unit is included in the structure. This gives the user the ability to address the entire file in one operation. This is useful for situations in which the user wishes to read the entire file. If there were no root node, the user would have to read the file one record at a time in separate operations. Again, the contents of data units for files of this document type are restricted to text.

The FTAM-2 document type definition also places restrictions on how various operations can be applied to files. For example, the erase operation can only be applied to the root FADU (i.e., only the whole file may be erased). Furthermore, new records can only be added to the end of the file and existing records cannot be changed or updated (i.e., only read access is allowed on existing records).

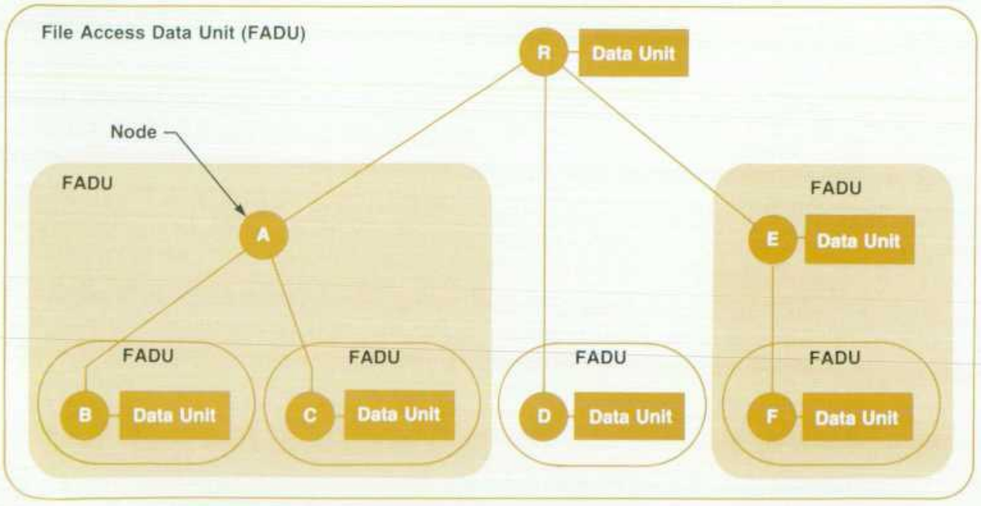


Fig. 1. *General FTAM file model.*

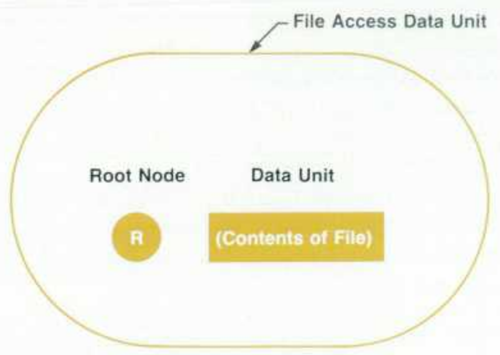


Fig. 2. *Structure of an FTAM-1 file.*

Each FTAM implementation supports a certain set of document types. Developers will usually choose to support document types that closely model the file types supported by their operating systems plus document types that model files of other vendors for interoperability purposes. For example, the document types FTAM-1 and FTAM-3 map naturally onto HP-UX file systems, while the document type FTAM-2 maps naturally onto record-oriented file systems. FTAM-3 is identical to FTAM-1 except that FTAM-3 files contain only binary data. HP's MAP 3.0 FTAM/800 supports all three of these document types.

Services and the Protocol

FTAM defines a set of services that can be requested by the user. The services available in FTAM allow users to establish, release, and abort connections, create and delete files, open and close files, read and change the attributes of a file, read, write, and erase a file, and move to different locations in a file. The order in which these services can be used is also defined and represents the FTAM protocol. The protocol is strictly connection-oriented and follows the requestor/server model (see Fig. 4). In this model, a requestor (initiator) makes requests to a specific server (responder). The responder can handle requests simultaneously from more than one initiator. In FTAM, the initiator is the user and the responder acts as the entity manipulating the (possibly remote) virtual file store.

During protocol exchanges, the initiator and responder build a series of nested stages or regimes. There are four such regimes that can be constructed during the course of an FTAM dialogue. These regimes are:

- The FTAM regime (a connection is made between initiator and responder)
- The file selection regime (a specific file is selected or created)
- The file open regime (the selected file is opened)
- The data transfer regime (data is transferred between the initiator and responder).

FTAM Regime. The FTAM regime is entered when an initiator establishes a connection with a responder. When establishing a connection, the initiator and responder negotiate what can be done on the connection. This negotiation is necessary because implementations are not required to support all of the FTAM functionality. This negotiation takes place in two steps. First, the initiator sends a connect request that contains information concerning what operations the initiator wishes to perform on the connection. Next, the responder issues a connect response back to the initiator indicating which of the requested operations the responder supports.

The initiator may terminate the FTAM regime (i.e., close the connection) using the FTAM terminate request. A connection may be terminated at any time using the FTAM abort service. This is an unorderly method of terminating a connection. For example, if a connection is aborted during a data transfer, the file being accessed may be left in an undefined state.

File Selection Regime. Once the FTAM regime has been established, the next step is to select or create a file. All operations in subsequent regimes are applied implicitly to this file. When the initiator is finished with this file, a new file can be specified by exiting and reentering the file selection regime. Once the file selection regime is established, the initiator can request file management operations without entering any other regimes. For instance, the initiator may read the attributes or modify the attributes of the currently selected file. The file selection regime can be terminated by the initiator by deselecting or deleting the file using the FTAM deselect or FTAM delete service. The FTAM protocol then reenters the FTAM regime.

File Open Regime. Before any data transfer can take place, the selected file must first be opened. This is the purpose of the file open regime. When a file is being opened, the initiator specifies the data transfer operations to be performed on the file. Once the file open regime is established, two data access operations may be performed prior to the establishment of the data transfer regime. The initiator may either erase all or part of the file or move to a specific location in the file. When all data access and data transfer operations are completed for the currently opened file, the file open regime can be terminated by the initiator using

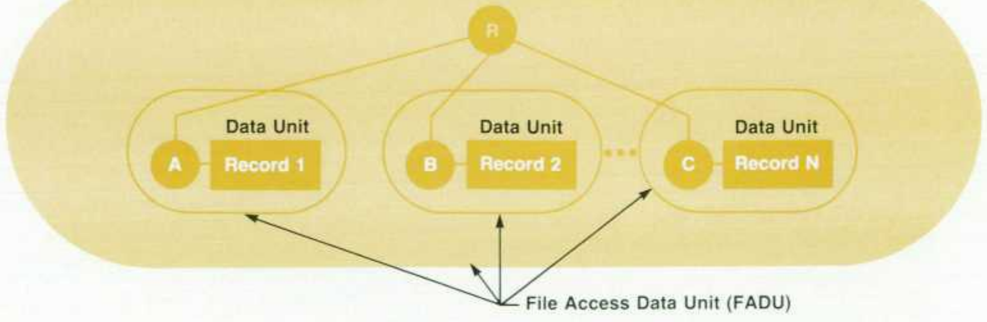


Fig. 3. *Structure of an FTAM-2 file.*

the FTAM file close service. The FTAM protocol then reenters the file selection regime.

Data Transfer Regime. The innermost regime of the FTAM protocol is the data transfer regime. In this regime, all or part of a file is transferred unidirectionally between the initiator and responder. There are three steps involved in this regime. First, the initiator specifies the direction of the data transfer (i.e., reading or writing) and the parts of the file to be transferred. Next, the specified data is transferred. Finally, an orderly end-of-data transfer is performed. This entire process may be performed more than once without having to close the file. Once the data transfer regime is completed, the FTAM protocol reenters the file open regime.

FTAM defines an optional error recovery protocol that facilitates recovery from errors such as a broken connection or the failure and restart of a remote host.

HP MAP 3.0 and FTAM

HP's MAP 3.0 FTAM/800 product is completely compliant with the FTAM standard ISO 8571 and the NIST Phase II Implemention Agreements.[3] HP's virtual file store supports a complete implementation of three document types: FTAM-1 (unstructured text files), FTAM-3 (unstructured binary files), and FTAM-2 (record-oriented text files). The VFS also supports the full functionality of concurrency control, the majority of the access control functionality, and many other attributes. The protocol implementation is also relatively complete. All major components of the protocol are implemented with the exception of restart and recovery.

HP MAP 3.0 FTAM/800 consists of three different processes: a user process, a service provider process, and a responder process (see Fig. 5). This implementation follows

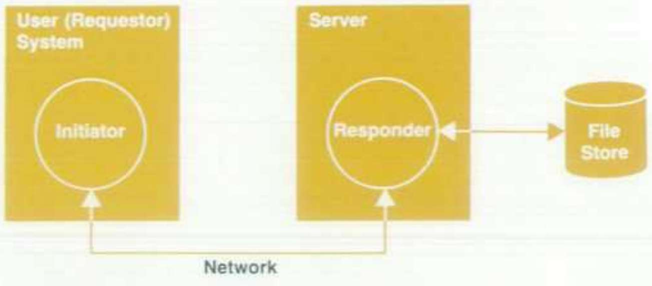


Fig. 4. *FTAM requestor/server model.*

the HP MAP 3.0 upper layer architecture described on page 11.

User Process

The user process contains the user's application program along with the MAP FTAM library. The MAP FTAM library is a set of C functions callable by a user's C program. The functions in the library provide the MAP FTAM application program interface for user applications. The library validates parameters passed in calls to its functions and translates these calls into requests to the service provider process. The service provider process and the user process communicate using HP-UX domain sockets.

The MAP FTAM interface provides two types of calls: low-level calls and high-level calls. Low-level calls perform one FTAM service request while high-level calls perform a sequence of FTAM service requests. An example of a low-level call is ft_create(), which creates an FTAM file, and an example of a high-level call is ft_fcopy(), which performs all of the FTAM service requests necessary to copy a file from one location to another.

To establish a connection to a remote responder using

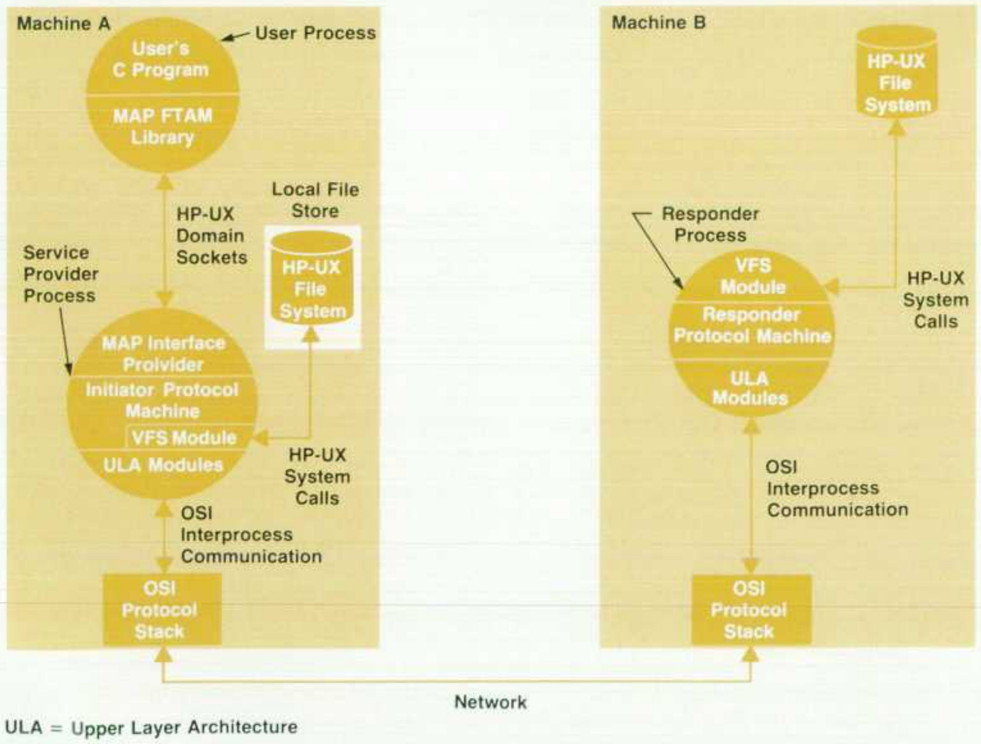


Fig. 5. *FTAM implementation for HP MAP 3.0.*

the MAP FTAM interface, the user must specify the address of that responder. The MAP FTAM interface supports two forms of addressing: directory distinguished names and presentation addresses. If the user supplies a directory distinguished name, the MAP FTAM library issues a request to the X.500 directory service, which resolves the name, and returns the appropriate presentation address. The MAP FTAM library can then issue the connect request to the lower layers of the OSI stack. If the user provides a presentation address, the X.500 service is not used. See the article on page 15 for more information about X.500.

Each call has two modes of operation: synchronous and asynchronous. In synchronous mode, the request completes entirely before control is returned back to the caller. In asynchronous mode, the call returns once the request is sent to the initiator protocol machine. Another call is provided so the user can check the status of outstanding requests and receive the results of requests when they complete.

Service Provider Process

The service provider process, which is forked by the user process, contains three modules: the MAP interface provider, the initiator protocol machine, and the VFS module. The function of the MAP interface provider is to translate MAP FTAM requests into requests to the initiator protocol machine. For low-level MAP requests, this translation is trivial since the low-level MAP FTAM requests are modeled after the initiator protocol machine services. However, the high-level MAP FTAM requests like ft_fcopy must be issued in a series of calls to the initiator protocol machine.

The initiator protocol machine accepts FTAM service requests from the MAP interface provider and validates the requests to ensure that they are made in valid protocol sequence and that the parameters are compliant with ISO 8571 and the *NIST Phase II Implemention Agreements*. After validation, one of two actions takes place. If the request involves a file on a remote system, the request is encoded using ASN.1 and sent to the responder operating on the remote system. If the request involves a local file, a request is issued to the VFS module, which accesses the file from the local file store. Note that requests for local files could have been handled in the same manner as requests for remote files. This different method of local file access was added as a performance enhancement.

Responder Process

The responder process, which runs as a daemon (server) process, contains the responder protocol machine and the VFS modules. It waits for connect requests from an initiator and when one is received, it forks the child process that handles all operations on that connection. When the connection is terminated, the child process exits.

The VFS modules are a library of functions that are linked with both the service provider process and the responder process. The implementation is straightforward with a few exceptions. The exceptions occur in the areas of concurrency control and attribute support. Since HP-UX does not provide a file locking mechanism sufficient to handle FTAM's concurrency control features, shared memory was used to implement concurrency control. Because FTAM requires the support of attributes that do not map naturally to HP-UX file systems, shadow files are used to store various FTAM file attributes. One shadow file exists for each file accessible with FTAM. A shadow file resides in the same directory as the file it describes and makes use of a "_" prefix on the filename to make it transparent on an HP-UX ls (list contents of a directory) command.

File Copy Scenarios

Three different scenarios exist when copying a file using FTAM. The first and most common scenario occurs when a file is copied from a remote node to the local node. In this scenario, one user process, one service provider process, and one responder process exist. The VFS in the service provider process handles all access to the local HP-UX file system while the VFS in the responder process on the remote node handles all access to the remote HP-UX file system. The data is transferred between the service provider process and the responder process using the lower layers of the OSI protocol stack.

The second scenario occurs when a file is copied from one location in the local file store to another location in that same local file store. In this case, there is one user process and one service provider process, but no responder process. The VFS in the service provider process handles all access to the local file system.

The third and final scenario is the situation in which a file is copied from one remote file store to another (or the same) remote file store. In this case, there is one user process, one service provider, and two responder processes. The responder on the source node transfers the contents of the source file to the service provider process. The service provider process receives the data from the responder process on the source node and sends the data to the responder process on the destination node. The responder process on the destination node receives the data from the service provider process and writes the data to the destination file.

A more efficient method of transferring the data would be to have the two remote responder processes communicate directly with each other. This method of communication is not possible with the FTAM protocol.

FTAM Design and Development

The development of the FTAM product involved project teams from HP's Colorado Networks Division (CND) in Fort Collins, Colorado and Information Networks Division (IND) in Cupertino, California. There was also coordination with HP's Roseville Networks Division (RND) in Roseville, California, where the HP OSI Express card was developed. The FTAM initiator and responder protocol machines and the VFS module were developed at IND at the same time the MAP FTAM interface was being developed at CND. Since some of the interface code resides in the same physical process as the protocol machines, a great deal of communication was necessary to ensure the compatibility of the module interfaces. The need for close communication between geographically separate divisions, coupled with the large code size of FTAM (75 KNCSS), created a complex engineering problem that forced the project teams to follow several "best practices" and to create a few of their own.

MAP FTAM Interface Development

Because the design effort and FTAM expertise were distributed over two organizations, the design of the MAP FTAM interface for FTAM was difficult. At the time, the protocol machines and the VFS modules were already in the coding phase. Working from the MAP FTAM interface specification and the external specifications of the FTAM protocol machine modules, the interface designers used structured design methods for the initial design. Structure charts were used to document the high-level design, and proved to be a valuable tool for communicating the design to IND FTAM engineers at a high-level design review. This review and the tools helped to uncover many design issues that would have gone unnoticed until the integration and testing phase, which might have jeopardized the timely release of the product.

The next step was the low-level design. In this step, pseudocode for the interface modules was developed that contained the major logic in the design. Again, a design review was held to review this pseudocode and more defects were uncovered.

Next, the code was developed using a technique called sliced implementation. In this approach, complete vertical slices of functionality are implemented in several stages. This approach worked well with the MAP FTAM interface since this interface contains about thirty calls. These calls were grouped into four disjoint sets and implemented in four slices. The first slice contained the connection management related calls, the second slice contained the low-level file management calls (e.g., create, open, close), the third slice contained the low-level data transfer calls, and the last slice contained the high-level calls.

The development and module testing of each slice were performed in an efficient, pipeline fashion. When the developers finished a slice, the code was handed to another engineer who was in charge of module testing each slice. While this slice was being module tested, the developers worked on the next slice.

Module Testing

The testing of the interface modules also posed an interesting problem for several reasons. First, as the slices were completed, they could not be integrated with the protocol and VFS modules since those modules were not finished with their early testing phases. Second, the interface modules reside in two different processes (user process and service provider process), and when the initial slices were being completed the utility routines for such functions as interprocess communication (messaging) and managing connections with remote machines (connection management) were not finished. Finally, the amount of code in the MAP FTAM interface modules alone is about 26 KNCSS and their interfaces with other modules are numerous and complex.

Given these problems, we had to develop a special method of testing these modules. Writing C code to simulate the nonexistent modules would have been too time-consuming because each stub routine would have been too general, or numerous sets of stubs would have been written generating numerous executable files. Also, because of the amount and complexity of the parameters at each module interface, validation of results would have been difficult.

To aid in this testing effort, a general module testing tool was developed. This tool facilitates the rapid development of module tests by alleviating some of the problems mentioned above. Testing using this tool involves two steps. First, the tester describes the relevant interfaces between the module being tested and the nonexistent modules. Given this interface description, the tool generates an executable file that contains the code of the module to be tested, the code for an interpreter for executing module test scripts, and code for transferring control between the module interfaces and the interpreter.

The second step is to write test scripts that are executed by the interpreter mentioned above. These test scripts are written in a language designed specifically for module testing. The language is similar to C but is scenario-oriented and contains facilities that simplify parameter validation. Because the language is interpreted, errors in test scripts can be quickly changed and rerun without having to recompile and relink. Using this language, the tester can specify the order of events at the interface of the module being tested. These events include calls to routines in the module and calls from the module to routines in nonexistent modules. Each occurrence of a call to a nonexistent routine is handled separately in the scenario. This simplifies the task of writing stubs yet allows a great deal of flexibility since the script code for each call can be tailored for that specific instance. By forcing the tester to specify the exact scenario for a test, a more rigid test environment is created where spurious calls by the module or calls made out of order cannot go undetected.

At strategic points in a test script, the tester can place statements to validate various parameters and other data values. These statements are assertions or print statements. The printing facility provided by the tool automatically formats the output according to the type of value printed. It can even print an entire recursive data structure.

While a script is running, the tool prints out information about the events that have occurred. When any failure occurs, the tool reports the failure and exits. The exact location of the failure in the scenario is known immediately by the tester. This information helps in the debugging process. If the information printed by the tool is insufficient to solve the problem, the script can be reexecuted in any conventional debugger such as xdb.

Product Test Suite

The test suite for the integrated FTAM product was designed and developed as if it were an actual product. This means that in the early stages of design, much consideration was given to creating a simple but flexible test suite that had all the characteristics of an HP product such as usability and supportability.

Typically, there are multiple test suites for a product. For example, a product may have a functional test suite, a reliability test suite, a stress test suite, a performance test suite, and so on. Instead of having completely independent test suites for each type of testing, the FTAM test suite encompasses all types of tests. These suites were developed under a common umbrella, sharing a common architecture, interface and output format. Among the advantages gained by using this philosophy are that a large amount of common code can be shared across tests and the learning curve for supporting and using the tests is reduced considerably.

The FTAM test suite features one uniform command line interface for all of the different types of tests. The interface contains many options that give the user a great deal of flexibility in controlling the degree of testing. Options exist for specifying the test or set of test cases to run, the number of iterations for each test, how long to run the tests, the number of connections to use during testing, the names of the nodes (machines) to be used during testing, and so on. For example, the user can specify that a set of reliability tests run for 72 hours using a given set of nodes running under a certain level of stress. There is also a single configuration file that can be used to specify the values of various parameters needed during testing.

The test suite can be run manually or automatically using an automated test tool such as the internal HP tool known as the scaffold.[4] The output printed by the test suite contains only simple passed or failed messages. This allows a novice user to determine the results of a test easily. If a test fails, the parameter information of the call that failed is printed to aid in solving the problem. The output printed by the test suite is uniform (and deterministic) across all tests.

Each test in the suite is written in a uniform manner so that all tests have some similar characteristics such as the same architecture, interface, and naming conventions. Once the design of one test is understood, the design of all tests for the product is also understood. A large number of library routines exist that were designed to be as test case independent as possible. These two factors greatly simplify the addition, modification, and deletion of tests or test cases.

The time spent designing and developing the test suite has been worth the cost. The test suite has proven to be very usable and all members of the FTAM team have become expert users of the suite within a short time. Tests have been added and modified by different team members with very few problems. Other project teams have already adopted the architecture and methodology of this test suite in the test suites for their products. Finally, a point worth mentioning is that the amount of code for the test suite is actually smaller than that of the product. This is a noteworthy achievement given the complex nature of the MAP FTAM interface.

Performance

Once the initial product was integrated, performance tests were run to determine FTAM's file transfer throughput rate. We discovered that FTAM could only transfer a file at a rate of 3 kbytes/s on an HP 9000 Series 815 computer. For this reason, a performance task force was organized consisting of engineers representing all areas of the OSI architecture. This task force identified major improvement areas by taking performance numbers at different layers and then analyzing execution traces to determine the most significant areas for performance improvement.

The areas selected included the addition of the VFS to the service provider process in the initiator, modification of transport configuration parameters, and reduction in the use of heap memory. With these changes, FTAM's throughput increased from its initial value to 60 kbytes/s on two HP 9000 Series 815 machines, and to 225 kbytes/s on two HP 9000 Series 835 machines.

Acknowledgments

Many individuals contributed to the design and development of the FTAM product. The engineers and managers involved with FTAM were Suhas Badve, Rajiv Batra, Atul Bhatnagar, Alan Burke, Jeff Conrad, Dave Hendricks, Lisa Kozlowski-Owen, Paul Melmon, John Smith, and Don Tiller.

References

1. *Information Processing Systems - Open Systems Interconnection - File Transfer, Access and Management,* ISO 8571 (Parts 1-4), International Organization for Standardization, 1988.
2. *Manufacturing Automation Protocol Specification,* Version 3.0, General Motors Corporation, 1988.
3. *Stable Implementation Agreements for Open Systems Interconnection Protocols,* NIST Special Publication 500-162, National Institute of Standards and Technology, December, 1988.
4. C. D. Fuget and B. J. Scott, "Tools for Automating Software Test Package Execution," *Hewlett-Packard Journal,* Vol. 37, no. 3, March 1986, pp. 24-28.

HP MAP 3.0 Manufacturing Message Specification/800

The first release of HP's implementation of the MMS standard offers powerful communication tools for monitoring and controlling robots, PLCs, and other factory-floor devices in the manufacturing environment.

by Peter A. Lagoni, Christopher Crall, and Thomas G. Bartz

THE MANUFACTURING FLOOR of the 1990s is a highly competitive and complex environment. Sophisticated factory-floor devices such as sensors, logical controllers, and robots are used to manufacture a variety of products. These factory-floor devices require powerful communication links to create a fully automated manufacturing environment.

To establish an environment such as this is not an easy undertaking, and requires some critical components. First, the tools and devices must be sufficiently sophisticated to be applied to the task. Second, a means for controlling and coordinating all of these tools in the environment must exist. By some well-defined method, distributed or hierarchical control must be established and communicated among the computers and devices.

Before the advent of the Manufacturing Automation Protocol (MAP), many of these components already existed. Developers of factory-floor devices and controllers have for some time provided mechanisms for controlling and monitoring their products. However, these devices have lacked another critical component—standardization—which provides reduced development and implementation costs. When each robot or device requires its own proprietary control language, the work required to develop a fully automatic manufacturing environment is complicated greatly. If a designer consciously decides to reduce this complexity by limiting the number of proprietary control languages to be used, a corresponding limitation of available devices results.

The Manufacturing Message Specification (MMS) standard was developed to meet this need. Its success hinges not only on its standardization, but also on its sophistication. To be a viable alternative to existing methods, it must offer minimally the same level of control previously available through proprietary means. We believe MMS achieves this goal, and we will devote much of this article to describing the powerful functionality that MMS delivers and how we've implemented much of it in HP's first MMS product offering, HP MMS/800. Before exploring this functionality in greater detail, the development of MMS and the value it provides will be discussed.

History of MMS

In 1980 General Motors Corporation formed the GM MAP Task Force to identify or create standards for multivendor factory-floor communication. The MAP architecture developed by this task force is based on the International Organization for Standardization (ISO) *Reference Model for Open Systems Interconnection* (OSI). It uses standards or subsets of standards that are appropriate for manufacturing at each layer of the OSI reference model. Because there was no existing standard or even work in progress on a standard that addressed the application layer (layer 7) communication needs between cell controllers and the devices that they controlled, the MAP Task Force developed the Manufacturing Message Format Standard (MMFS, pronounced "Memphis") to meet this need. MMFS was part of the MAP 2.1 specification, which was published in 1985 and became the first generally useful version of MAP.

While MMFS provided a description of an application layer protocol that could be used for factory-floor communication, an international standard in this area was still needed. Even before publication of the MAP 2.1 specification, the Electronic Industries Association (EIA) agreed to adopt all of MMFS and to reissue it in a form that would be acceptable as an ISO standard. This new version became known as the Manufacturing Message Specification (MMS). MMS is identified by the document numbers issued by both EIA and ISO. The EIA number is RS-511 and the ISO number is ISO 9506. While these numbers refer to documents maintained by two different organizations, the documents are identical.

While MMS was initially based on MMFS, the current form of the specification shows little resemblance to its predecessor. One of the reasons for this is that the documentation format required by ISO is different from the one used for MMFS. Another is that ISO application layer specifications are designed to be implemented above an ISO presentation layer. MAP 2.1 had a null presentation layer. A third reason for this change is that the MMS specification was developed by a committee made up of employees of many companies, not just General Motors, and input from these different sources resulted in the specification's being expanded to meet newly identified needs.

MMS went through many drafts as it progressed through the standardization process. By the time MAP 3.0 was issued, MMS had progressed to an ISO draft international standard (DIS). It is this version that is included in the MAP 3.0 specification and implemented in the HP MMS/800 product.

An ISO DIS is one level below the most stable designation given by ISO, which is ISO international standard (IS). A specification is almost always changed as it moves from DIS to IS status, and MMS is no exception. Since MAP 3.0 was published, MMS has become an international standard, and there are differences between the version of MMS specified in MAP 3.0 and the version of MMS specified by ISO IS 9506. While these differences are not major, they do make MAP 3.0 implementations incompatible with IS-based implementations.

After the MAP 3.0 specification was published, General Motors turned over the responsibility for continued work on MAP to the MAP/TOP Users Group. The MAP/TOP Users Group sponsors an MMS technical committee to address MMS technical questions. This committee is currently addressing the MMS incompatibility between MAP 3.0 and IS 9506 and should have a solution in the near future.

While it is necessary to have a well-defined standard to allow multivendor communication, a standard cannot address all of the implementation details. No matter how carefully standards are written, there can be differences in their interpretation. To address these problems, implementation agreements are necessary. Before publication of the MAP 3.0 specification, an MMS special interest group was created by the Industrial Technology Institute in Ann Arbor, Michigan. This group developed implementation agreements that were used to demonstrate the feasibility of MAP at the Enterprise Network Event held in Baltimore, Maryland in June 1988. Many of these implementation agreements are also included in the MAP 3.0 specification.

Following the move of MMS from DIS to IS, MMS was included in the U.S. National Institute of Science and Technology (NIST) OSI workshop. It is expected that future versions of the MAP specification will include the implementation agreements from this workshop.

MMS Interface

MMS addresses the content of the messages sent over the network. It does not, however, address the application program interface provided by an implementation of MMS for use by an application developer. An application program interface is the set of library calls provided by an MMS implementation to the MMS user. These library calls are used both to send and to receive MMS messages. The MAP 3.0 specification includes a definition for the MMS interface (MMSI). The definition specifies the function calls, their parameters, and how each function call will be used. This interface allows application developers to write applications that can easily be ported to other vendor's products that support the MMSI. This interface was developed by the MAP application programming interface committee. Following the publication of the MAP 3.0 specification, this committee moved to the MAP/TOP Users Group. The HP MMS/800 product is the first MMS product that supports this interface.

During the time that HP MMS/800 was being developed, the committees working on the MMS standard, the MMSI standard, and the implementation agreements were very active. Two of the members of HP's development team participated, and continue to participate, in meetings pertaining to MMS. This active involvement in these committees allowed the development team to track changes and incorporate them into the product in a timely manner. It also allowed Hewlett-Packard to have a voice in the decisions made by these committees and to contribute to the success of both MMS and MAP 3.0.

Now that MMS has reached international standard status, most of the future MMS work will involve the creation of companion standards to work with MMS to meet the special communication needs of the various types of devices that use MMS. Work is currently progressing on companion standards for robots, programmable logic controllers, numerical controllers, and production management (or cell controllers). Companion standards may also be used to address the special needs of certain industries. The process control industry is currently creating a companion standard for its use. Implementation agreements will be needed for these companion standards as they become stable enough to implement. The MMS special interest group at the NIST OSI workshop will create these agreements.

The Problem MMS Solves

Within the OSI networking reference model, MMS is an application layer (layer 7) protocol. A layer 7 entity is not an application; instead, it provides tools and programmatic access to facilitate application development. Manufacturing software design engineers develop their own applications directly above the MMS services to solve a specific manufacturing problem.

MMS functionality is organized into ten service sets, each of which targets a specific set of tasks or areas of control, such as connection establishment and maintenance, device status monitoring, uploading and downloading of data and executable images, remote execution management, and variable creation and access.

Before delving into the specifics of these and other service sets, it is helpful to develop a basic understanding of the manufacturing environment and its layout. Fig. 1 shows a classic layout of the manufacturing floor from the networking point of view. There are four levels of hierarchy: plant manager, area manager, cell controller, and factory-floor devices.

Primary plant control and information assimilation are coordinated by the plant manager. Area managers are responsible for controlling what is to be manufactured at any time by specifying this information and coordinating it among the various cells on the factory floor. The cells are linked together according to the chain of steps required to assemble a given product. Each cell consists of a cell controller and a number of devices. These devices may be programmable logic controllers, numerical controllers, robots, or sensors. The cell controller directs the activities of its devices based on its assigned responsibilities. The area manager monitors progress as it is made and is alerted by cell controllers of any problems that may arise. As this information is collected, it is made available to other sites such as the corporate offices via wide area networks or dedicated links.

Communication between the area manager and the cell controllers within its jurisdiction is normally accomplished through a file transfer protocol, such as FTAM

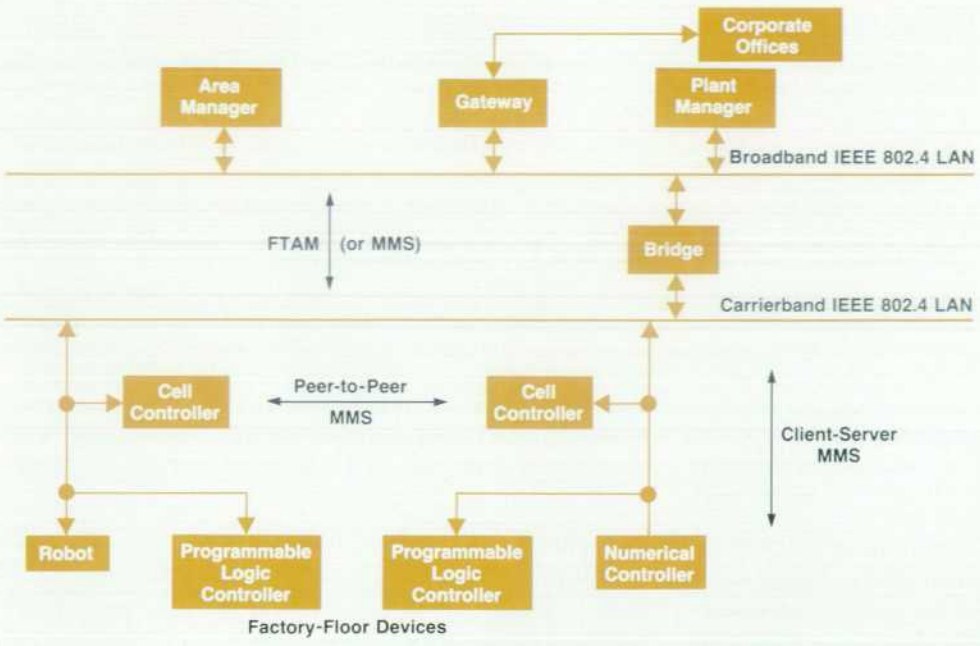


Fig. 1. *The manufacturing floor from the networking point of view.*

(File Transfer, Access, and Management—see article, page 24). The information exchange often consists of instructions (binary executables) and tables of data (e.g., Car X1, 2-door, color blue, convertible; Car X2, 4-door, color brown, sedan; etc.) that dictate what is to be manufactured and in what order. Data tables can also consist of event logs (for diagnostic or inventory purposes) collected by the cell controllers and showing what was indeed manufactured and the times at which corresponding events took place. MMS also provides file transfer capability that can be used in this capacity, but it is less efficient than FTAM.

Once a cell controller has received a set of instructions, it is responsible for carrying them out. Doing so will normally consist of establishing a connection to each device in its cell and downloading executable and data images over these connections. MMS remote process management can then be used over this connection to start the executable images. At this point, specific MMS variables can be created and initialized at the devices for control and monitoring purposes. The cell controller will likely establish connections with neighboring cell controllers for peer-to-peer MMS communication. Variables, semaphores, and event monitors associated with these peer-to-peer connections will then be created to provide synchronization of the activities on the assembly line (e.g., as a car passes from one cell to the next, each cell must be notified of its arrival.)

This example is one manner in which MMS can be used to facilitate and coordinate the production process. There are certainly many other approaches to organizing a factory floor and incorporating MMS.

We are now ready for a more detailed explanation of the MMS services.

MMS Services

There are ten service sets in the MMS standard, each of which defines a number of unique MMS messages. These services sets are as follows:

Context Management. The context management services offer the ability to administer associations or connections and the environment in which those connections are established. This includes opening, closing, and aborting connections, as well as notifying applications of aborted connections and protocol errors as they occur. Also included are utilities to prepare the environment before any connections can be initiated or received. These utilities spawn any necessary processes and initialize global variables, tables, and lists.

Connections have associated with them a certain context for communication, which is negotiated at the time of establishment. This context dictates what can be communicated on that connection and ensures that both ends understand the contents of the messages to be exchanged. For example, a context for MMS will differ significantly from an FTAM context because the message structures are very different. MMS and its companion standards have their own contexts. The context to be used is specified by the initiator of the connection. If the responder cannot support the specified context, it is forced to deny the connection request.

VMD Support. VMD stands for *virtual manufacturing device*. It refers to the model used and described by the MMS documents to characterize the behavior of a manufacturing device. A VMD can be thought of as the name MMS uses to refer to a device under its control. The VMD support services are used to obtain information about a device, such as its identity (manufacturer and model number), its status, what MMS objects it has defined (variables, semaphores, journals, etc.), and what resources it has available.

Domain Management. A *domain* is an abstract object defined within the VMD model. It represents a subset of a VMD's resources. Domains are analogous to object files and can be grouped together to form an executable program. Examples of VMD resources gathered within a domain are data files, object code, and memory. The domain management services are used to create domains by reserving a set of VMD capabilities (or resources), associating a name with them, and downloading the corresponding binary image data. Furthermore, domains can be uploaded and stored for future use, deleted, or interrogated to determine their attributes. Some manufacturers provide static domains with their devices; these require no downloading.

Program Invocation Management. These services are used to define a program invocation as a collection of domains, and then control its execution. When creating a program invocation, a list of existing or downloaded domains is sent to the VMD and a name is given that binds them together. Once created, the invocation can be controlled by issuing start, stop, resume, reset, kill, or delete commands. The domains that make up the program invocation will usually consist of executable device code and the data the device uses to perform its assigned task.

Variable Access. The variable access services make it possible to define remote variables and their types, and to read and write those variables as needed. The variables may be simple types, complex structures, or lists of other variables. Named variables are defined by associating a name and a previously defined type with a given memory address at the VMD. Once defined, variables are easily accessed with read and write messages.

Semaphore Management. The semaphore management services allow applications to create and access semaphores for synchronization, control and coordination of shared resources. Two types of semaphores can be managed by MMS to provide this. *Pool semaphores* have a one-to-one correspondence with acquirable, logical, or physical entities at the device. By taking control of a pool semaphore, an application locks the corresponding resource for exclusive use. *Token semaphores* are provided for synchronization. If a given application has control of a token semaphore, subsequent requests by other applications to control the semaphore are queued.

Operator Communication. These services provide communication with remote operator stations. Remote operator stations are simple input/display devices that provide for human interaction with the devices and cell controllers. The input service is used to request entry data from a device such as a bar code reader, keyboard, or optical sensor. A device such as a monitor or printer may be used as an operator station to display data from the output service.

Event Management. Event management allows a client MMS user to define and manage events at a VMD and to be notified when those events occur. This is accomplished by explicitly defining event conditions and event actions. These are then tied together through event enrollments at the VMD. An event enrollment dictates what event action (MMS service) is to be taken by the VMD when a given event condition occurs. Events can be triggered normally, as they occur, or artificially by the MMS user. Additional MMS services are provided to obtain current status summaries of event conditions and enrollments.

Journal Management. Journal management offers a facility for recording and retrieving chronologically ordered information such as events, variables corresponding to those events, or general textual comments. These services allow the MMS user to create, delete, initialize, read, and write journals and obtain status information about them.

File Management. These services offer simple file transfer and file management capabilities. MMS services are provided to open, read, and close files. When used together, these three services are used to transfer a file from the remote to the local system. To transfer a file to a remote system, the obtain-file service is used. This tells the remote system to obtain a file from a specified file server. The requesting system may be that file server. Files can also be deleted or renamed, and directory listings can be obtained through a file directory service.

All of these MMS services are either confirmed services or unconfirmed services. Both confirmed and unconfirmed services are initiated by an MMS user issuing a service request. This request is sent out on the network and is received by the peer MMS user as a service indication. For an unconfirmed service, the service has been completed at this point. In the case of a confirmed service, the MMS user that received the indication must send a response back to the requesting MMS user, as shown in Fig. 2. This response is called a confirm when it is received by the MMS user that issued the original request. At this point the confirmed service is complete. Most of the services defined for MMS are confirmed services.

HP MMS/800 Definition

The services provided by MMS allow a wide range of applications. Unfortunately, it was not feasible for the MMS team to provide all eighty-six MMS services in the HP MMS/800 product. A narrower focus had to be developed. In determining this focus, the typical HP MMS/800 application was evaluated. This typical application consists of an HP-UX computer running MMS as a cell controller in the automotive, aerospace, and electronics industries. The factory-floor devices in this type of application include programmable logic controllers (PLCs), numerical controllers (NC or CNC machines), and robots.

To control these types of devices, a cell controller needs to support the context management, VMD support, domain management, program invocation, variable access, operator communication, and file services. The variable access and file services used in peer-to-peer communication are also included. The peer in this case may be another cell controller or an area manager.

The remaining services—semaphores, journals, and events—are mainly used in the process control industry. MMS/800 is not designed for this type of application at this time.

HP MMS/800 Services

Careful consideration was given to the set of services to be implemented. This set had to be large enough to provide the functionality needed for a cell controller, but small enough to allow the services to be implemented and tested in time to meet the HP-UX 7.0 operating system release

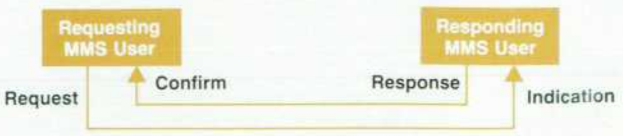


Fig. 2. *MMS services are either confirmed or unconfirmed. The response shown is omitted for unconfirmed services.*

schedule.

The major source of information used in choosing the set of functionality to implement was the MAP 3.0 specification. The specification defines seven classes of devices for which MMS will be used. The classes are:

- MAP1. Used for NC machines
- MAP2. Used for PLCs (simple)
- MAP3. Used for PLCs (complex)
- MAP4. Used for robots
- MAP5. Used for process control applications (simple)
- MAP6. Used for process control applications (medium)
- MAP7. Used for process control applications (complex).

A product considered MAP 3.0-conformant must implement all services defined in at least one of these classes. The set of services chosen for the first release of HP's MMS satisfies classes MAP1, MAP2, and MAP3. In addition, over 86% of the services in MAP4 are included in the first release. The box on page 38 lists the services provided in the first release.

HP MMS Implementation Model

As explained in the article on page 11 describing the upper layer architecture, the HP MAP 3.0 product is based on a two-process model. The two processes are the user process and the service provider process.

In the MMS version of the model (Fig. 3), the application program interface is the standard MMS interface (MMSI) introduced earlier in this article. The application entity coordinator is known as the high-level service provider and the application service element is known as the MMS protocol machine. Each of these MMS elements is described in more detail below.

The MMS User Process

The purpose of the MMSI is to allow the application program to exchange MMS messages with devices or peer applications. Therefore, much of the MMSI code simply

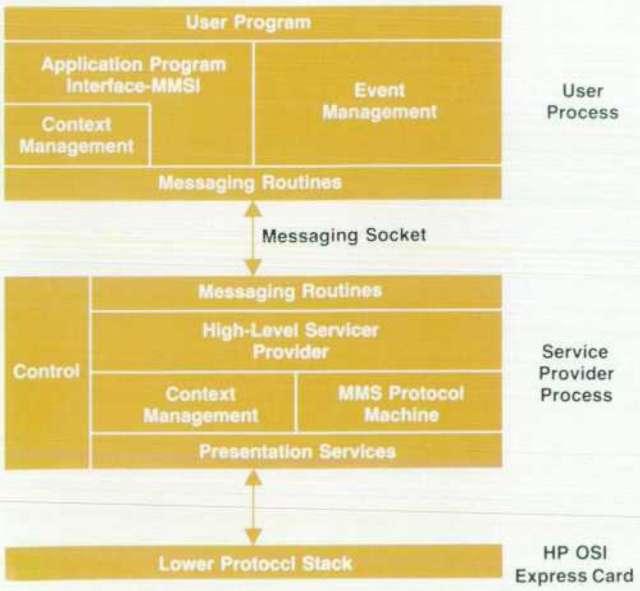


Fig. 3. *The HP MMS/800 implementation of the HP MAP 3.0 two-process model.*

takes input from the user in the form of parameters and input buffers and validates it. This data is then sent to the service provider process through the messaging socket. The service provider process expects the data in a format compatible with its data structures, which are derived from the Abstract Syntax Notation One (ASN.1) standard.[1] The MMSI performs the translation from MMSI standard structures to the service provider process's ASN.1-derived structures. The interface is also responsible for translating inbound data from the service provider process into the user's MMSI data structures.

The following algorithm describes the processing for most of the MMS request and response functions.

```
1. Validate the status of the connection and the input data
2. Translate the data from MMSI format to service provider process
   ASN.1-derived structures
3. Place the data in a standard message format to be sent to the service
   provider process
4. Add the event to event management
5. Send the message to the service provider process
6. If the event is synchronous
   {
     wait for a reply
     copy the data to the user's output buffer
   }
7. Return control to the user.
```

After control returns to the user for asynchronous events, the user can call the event management wait function. This function waits for the next completed event. At this time the interface copies the data for the completed event into the user's output buffer.

To receive indications (requests from the remote device), the application must explicitly ask for them. This is accomplished via the indication receive request lReceive. The lReceive function sends a message to the local service provider process indicating that the user is willing to receive one indication. It is the responsibility of the service provider process to pair the lReceive requests with indications received from the network. The service provider process will queue any indications it has received for which the user process has not issued an lReceive. The service provider process will also queue lReceive requests for which indications have not yet been received. When the service provider process is able to pair an indication and an lReceive request, the indication is passed to the user process. The indication is then placed in the user's lReceive output buffer where it can be processed by the application program.

The scenarios above for requests, responses, indications, and confirms describe the general processing that takes place for most of the VMD support, domain management, program invocation, operator communication, and file services. However, the context management and variable access services require the interface to perform additional work and store complex information.

The context management services require the interface to keep track of local VMDs, application entities, and connections. Local VMDs are used when the application process wants to act as a server for the variable access services. To be a server, the local side must be viewed as a VMD by the remote client. These local VMDs are created by executing a function call indicating which application entities will belong to the VMD. An application entity is the part of an application that uses networking services. In an ISO environment, these application entities have addresses that the interface is required to store to facilitate the communication with the remote devices. In our implementation, an application entity corresponds to a service provider process.

The interface keeps track of the VMDs, application entities, and all connections to perform validity checking on operations requested by the user. The information for each VMD, application entity, or connection is stored in a separate data structure that is placed in the interface's VMD table. A simplified version of the VMD table is shown in Fig. 4. Hash functions are used to provide quick access to particular entries in the VMD table and doubly linked lists allow easy additions and deletions.

The VMD table also provides a means for storing MMS objects defined by the user. In our current implementation, variable and type objects are the only user-definable MMS objects supported. These MMS objects have an associated scope that dictates a range over which they can be used. For example, a connection-specific variable can be accessed only on the connection for which it was defined, whereas a variable with a VMD-specific scope can be accessed on any connection to the VMD. Since we support both connection and VMD-specific variable access, the objects for variables and types are stored under the structure corresponding to the particular connection or VMD.

Variable Access

The variable access services are one of the most powerful and complicated portions of MMS. The purpose of these services is to allow the user to read and write variables on the remote device. However, this is much more than a simple interprocess communication mechanism. Once the user has performed the necessary setup, the interface can access the user's program variables directly to place data received from the remote device or retrieve the data to be sent to the remote device.

To achieve this level of variable access, the interface must know all about the local variable. The interface must know the type of the variable, which is either simple, an array, or a structure. The interface must also know the address of the program variable, its name, and how the variable should look on the network. The MMSI provides functions that allow the user to provide this information to the interface dynamically. Once the information has been supplied, a function call is used to define the variable to the interface. This results in a variable access object structure being added to the VMD table in Fig. 4. The variable access object is placed under the VMD or connection structure defined by the scope of the object.

One final piece of information needed by the interface is knowledge of the C compiler packing rules. This information has been built into the interface. The packing rules, along with the type of the variable and its starting address, allow the interface to access the C program variable directly.

Once the user has defined the local program variable to the interface, reads and writes can be performed. For a read, the interface retrieves the data from the remote device and places it directly into the application program's C variable. On a write, the interface takes the data directly from the application program's C variable and sends it to the remote device. The local C variables in the application program can then be used in normal computations.

The MMS Service Provider Process

The MMS service provider process contains two modules that are specific to MMS. The first is the high-level service provider and the second is the MMS protocol machine. The rest of the modules that make up the MMS service provider process are the same service provider process modules used by the other HP MAP 3.0 components. The service provider process is spawned by the user process as the result of the MMS user issuing an application entity activation function call. The service provider process provides the communication pathway between the user process and the HP OSI Express card[2] and the network. This call must be made before any connections to the remote device can be set up. Once a service provider process has been activated, the MMS user can set up connections and issue and receive MMS messages over those connections.

The high-level service provider has two primary functions. The first is to manage all communication between the service provider process and the user process. This is accomplished with function calls to the service provider process messaging module. One management function is to pack messages being sent to the user process into a form that the user process can interpret and to unpack messages being received from the user process. The routines to pack and unpack MMS messages are discussed later in this article.

The second function of the high-level service provider is to process all incoming indications that are responded to automatically by this product. When the high-level service provider receives an incoming indication it checks to see if it is for an autoresponse service. If it is not an auto-

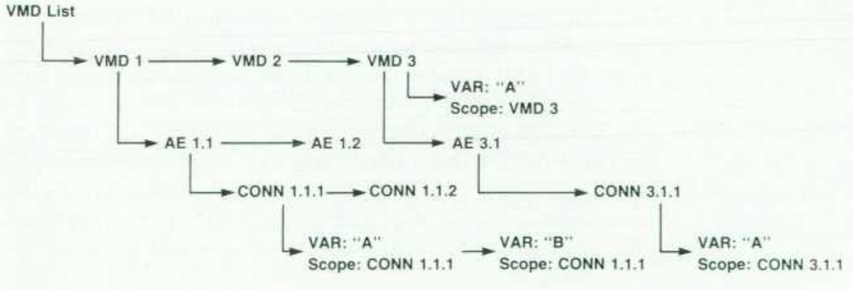


Fig. 4. *Data structure for context management.*

response service, the indication is sent to the user process to be processed by the MMS user. If it is an autoresponse function, the high-level service provider performs the related operation, creates the response, and sends it out on the network without notifying the MMS user. Eight autoresponse services are supported in the HP MMS/800 product. These include the identify service and all seven of the file services.

The primary function of the MMS protocol machine is to enforce the MMS protocol as it is defined in ISO DIS 9506. This entails validating the values of MMS message parameters and verifying that the state diagrams defined by the MMS specification are not violated. In the event that a protocol violation is discovered on an outbound MMS message, an error is returned to the local MMS user. If a protocol violation is discovered on an inbound MMS message, an MMS reject message is sent back to the remote device. In the case of a rejected inbound message, the local MMS user is also notified of a rejected inbound message with an MMS reject indication.

Both the high-level service provider and the MMS protocol machine must maintain information about connections and confirmed services. A common data structure is used to store all of this information. Fig. 5 illustrates the basic design of this data structure. When either a connect request or a connect indication is processed by the service provider process, a connection entry is added to the hash table with its state indicating that the connection is in progress. When a corresponding positive connect response or confirm is processed, the state of the connection entry is updated to show that the connection is valid. If the corresponding response or confirm is negative, the connection attempt failed, and the corresponding connection entry is removed. A similar process occurs when the connection is concluded.

When a confirmed service request or indication is processed by the service provider process, a request entry is added to the request entry list for the corresponding connection entry. When the response or confirm is processed, the request entry is removed.

Doubly linked lists are used for both the connection entries and the request entries. A common set of data structure manipulation routines is used to manage these lists. This includes initializing the data structures, adding new elements to the list, retrieving the next element on the list, and removing elements from the list. These routines are designed to manipulate general doubly linked lists and are used to manage data structures within the service provider process and the user process. Because they were written and tested early in the project, very few errors were discovered in these data structures during product testing.

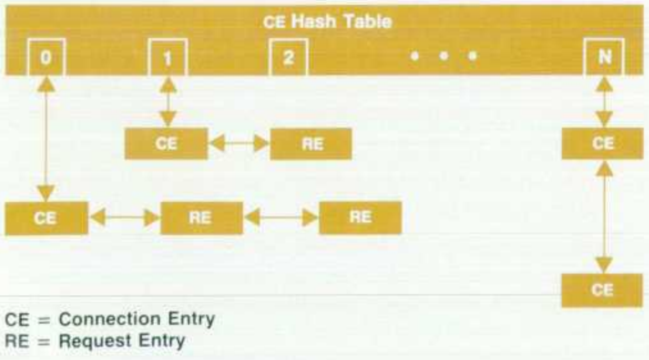


Fig. 5. *Data structure for information about connections and confirmed services in the service provider process.*

ASN.1 Compiler

Many of the modules in the HP MAP 3.0 product rely on the ASN.1 standard. For all of these modules, including HP MMS/800, the ASN.1 compiler described in the article on page 11 was used to generate encoding and decoding routines and data structures. While the purpose of the encoding and decoding routines is to translate MMS messages between the ASN.1-derived data structures and the bit patterns sent and received on the network, the MMS product uses them for an additional purpose: to validate certain MMS protocol requirements.

During connection establishment, the nesting level for variables and the parameter conformance building block are among several parameters negotiated for the connection. The value of the nesting level determines the extent to which structures can exist within other structures for variables defined for the connection. The value of the parameter conformance building block describes which fields within specific MMS messages will be valid for the connection. If the negotiated nesting level is exceeded or if an invalid field is present as determined by the parameter conformance building block, a protocol error occurs. To determine if either of these situations exists, the entire MMS message must be parsed and information must be gathered regarding both of these values. Since the encoding and decoding routines already parse MMS messages, additional code was simply inserted into these routines to collect this information and validate that neither the nesting level nor the parameter conformance building block is violated. Since the creation of the encoding and decoding routines by the ASN.1 compiler is an automated process, the insertion of this additional code was also automated. Use of the encoding and decoding routine saves an additional parse iteration of each MMS message and allows faster processing of each MMS message.

Automatic Generation of Pack and Unpack Routines

The ASN.1-derived data structures are used throughout the service provider process and to send the MMS messages to and from the user process. In the user process the MMS messages are translated between the ASN.1-derived data structures and those used by the MMSI. Since the ASN.1-derived data structures are automatically generated by the ASN.1 compiler, there is a pattern to their design, including the naming convention used for fields within the structures and their corresponding types. The MMS development team took advantage of this to generate the code to pack these data structures into a buffer when sending them across the channel linking the service provider process and the user process. After determining the nature of the data structure patterns, routines were written to process the header file generated by the ASN.1 compiler and locate each pointer field within every structure. Each pointer field represents a new structure within an MMS message which

HP MMS/800 Services

In the client-server model used by MMS, a cell controller is considered to be the client, while the factory-floor device is the server. For peer-to-peer communication, the cell controller must be able to act as the server as well as the client. For each of the services listed below, C or B is used to indicate whether the HP MMS/800 implementation handles the client side or both the client side and the server side. An A indicates that the service provider process (SPP) responds automatically in the server role. When the SPP automatically responds, the application program neither receives the indication nor sends a response.

- Context Management.
  - Initiate (B). Open a connection.
  - Conclude (B). Gracefully close a connection.
  - Cancel (B). Cancel a previously issued service request.
  - Abort (B). Abruptly close a connection.
  - Reject (B). Refuse an MMS message that violates the MMS protocol.
- VMD (Virtual Manufacturing Device) Support
  - Status (C). Request a VMD to provide its status information.
  - Unsolicited Status (C). Received an unsolicited status indication from a VMD.
  - Get Name List (C). Retrieve a list of named objects from the VMD.
  - Get Capability List (C). Retrieve a list of the capabilities of the VMD.
  - Identify (BA). Get basic information about the remote device (e.g., vendor name, model number, version number).
- Domain Management
  - Initiate Download Sequence (C). Instruct the VMD to start downloading a domain from the cell controller.
  - Download Segment (C). Send a segment (piece) of the domain to the VMD.
  - Terminate Download Sequence (C). Terminate the download, usually because the download is finished.
  - Initiate Upload Sequence (C). Instruct the VMD to prepare for uploading a domain to the cell controller.
  - Upload Segment (C). Instruct the VMD to upload the next segment of the domain.
  - Terminate Upload Sequence (C). Terminate the upload, usually because the upload is finished.
  - Request Domain Download (C). The VMD requests the cell controller to initiate a download.
  - Request Domain Upload (C). The VMD requests the cell controller to initiate an upload.
  - Delete Domain (C). Delete a domain from the VMD.
  - Get Domain Attributes (C). Retrieve the attributes of a domain from the VMD.
- Program Invocation
  - Create Program Invocation (C). Link one or more domains on the VMD into one executable program.
  - Delete Program Invocation (C). Delete a program on the VMD.
  - Start (C). Start a program on the VMD.
  - Stop (C). Stop (suspend) a program on the VMD.
  - Resume (C). Resume a stopped program on the VMD.
  - Reset (C). Reset a stopped program on the VMD.
  - Get Program Invocation Attributes (C). Retrieve information about a program on the VMD.
- Variable Access
  - Read (B). Retrieve the value of one or more variables from the remote device. These can be simple variables, arrays, structures, etc.
  - Write (B). Write new values to the remote device's variables.
  - Information Report (B). Send the values of the local machine's variables to the remote device without being asked for them.
  - Get Variable Access Attribute (B). Retrieve the attributes (type information, etc.) of one or more remote variables.
- Operator Communication
  - Input (C). Retrieve data strings from the remote device's console.
  - Output (C). Send data strings to the remote device's console.
- File services
  - Obtain File (BA). Ask the remote device to get a file (MMS does not allow writing of files, so the remote device is asked to read the file).
  - File Open (BA). Open a file on the remote device.
  - File Read (BA). Read part of a file on the remote device.
  - File Close (BA). Close a file on the remote device.
  - File Rename (BA). Rename a file on the remote device.
  - File Delete (BA). Delete a file on the remote device.
  - File Directory (BA). Get a directory listing from the remote device.

requires packing into the buffer. Once these fields were detected, it was easy to derive their pointer types from their names, and, together with some structural information for each data type, generate the packing code.

A two-step process was employed. First, the ASN.1-generated header file was parsed to generate lists containing the information required for packing. The elements on these lists indicated the opening and closing of substructures and unions, union tags, and pointers. Second, a recursive set of routines was written to interrogate these lists and generate the corresponding packing code for each data structure.

With minor modifications, the two-step process was repeated to generate the unpacking code as well. Over ten thousand lines of code were automatically generated using this process, saving an enormous amount of tedious coding and debugging time.

MMS Interoperability

As discussed earlier in this article, the reason for the development of the Manufacturing Message Specification was to provide a mechanism for multivendor communication on the factory floor. To provide this multivendor communication, the HP MMS/800 product must be able to communicate with MMS products from other vendors. HP is actively conducting interoperability testing with the MMS products of other vendors to ensure this communication. At the present time, interoperability has been demonstrated with a wide range of products from several other manufacturers. As new MMS products appear on the market, HP will continue this testing to provide MMS users a wide range of devices to be used with the HP MMS/800 product.

Conclusion

Until the advent of MMS, interconnecting factory-floor devices was a costly and complicated process. Each device and computer vendor had proprietary protocols that required specialized communication hardware and software to connect devices.

MMS and MAP 3.0 solve this problem by providing one common communication standard for all factory-floor devices. MMS provides the necessary functionality for controlling these devices. This allows engineers in the factory to choose the best vendor and device for a task without having to worry about the cost of specialized communication hardware and software to connect this new device.

In addition to providing factory-floor communication with MMS, MAP 3.0 has defined a standard MMS interface (MMSI). This standard interface reduces programmer training costs and allows applications to be ported easily from one vendor's computer to another. This provides the ability to create flexible applications and reduce development costs.

The HP MMS/800 product provides the application programmer with MAP 3.0 MMS functionality through the standard MMS interface. All the necessary functionality for controlling factory-floor devices and peer-to-peer communication with other cell controllers has been included in this product.

Acknowledgments

We would like to thank Steve Booth for his assistance in the code development of the MMSI module and Mike Robinson for helping in the automatic generation of the service provider process pack and unpack routines. Thanks also go to John Beckman, Hugh Mahon, Neal Bauer, and Mickey Gittler for their efforts in the development and testing of this product. We would also like to thank our managers, Don Tiller and Jeff Lindberg, for their support and guidance during this project. Finally, thanks go to the marketing people who helped make this into a real product. Tim Antonsen developed an excellent set of manuals that make MMS understandable. Isabel Garity provided training and sample programs, and Paul Morrison provided product marketing guidance.

References

1. *Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)*, ISO 8824: 1987 (E).
2. W.R. Johnson, "An Overview of the HP OSI Express Card," *Hewlett-Packard Journal*, Vol. 41, no. 1, February 1990, pp. 6-8.

Bibliography

1. A.H. Stacy, *The MAP Book: An Introduction to Industrial Networking*, Industrial Networking Incorporated, 1987.
2. *Manufacturing Automation Protocol Specification Version 2.1*, General Motors Corporation, 1985.
3. *Manufacturing Automation Protocol Specification Version 3.0*, General Motors Corporation, 1988.
4. *Manufacturing Message Specification*, ISO DIS 9506, International Organization for Standardization, 1985.
5. V.C. Jones, *MAP/TOP Networking: A Foundation for Computer-Integrated Manufacturing*, McGraw-Hill Book Company, 1988.

HP-UX Kernel Communications Modules for a Card-Based OSI Protocol Stack

HP MAP 3.0 products are based on the HP OSI Express card, which implements most of the OSI protocol stack on an I/O card. The kernel modules provide reliable data transfer between the host computer and the HP OSI Express card.

by Eric C. Scoredos, Kimberly K. Scott, and Richard H. Van Gaasbeck

THE OVERALL DESIGN of the HP MAP 3.0 product is based on the HP OSI Express card.[1] The HP OSI Express card provides an implementation of the OSI protocol stack on an I/O card for HP 9000 Series 800 computers. The card off-loads much of the network overhead from the host, leaving CPU bandwidth available for processing user applications and MAP services.

To provide the reliable transfer of user data between the host computer and the HP OSI Express card over the HP Precision Bus (HP-PB) backplane, three host-resident modules that run in the HP-UX kernel were developed for the HP MAP 3.0 product. These kernel modules provide an interface that allows users to make or break connections (communication paths) between different hosts on the IEEE 802.4 network and to send and receive data on these connections. Up to 100 separate full-duplex connections can be established between a given host and any other hosts on the network. The kernel modules and portions of the HP OSI Express card are shown in Fig. 1. The three kernel modules are:

- The upper layer interprocess communication module (ULIPC)
- The CONE (common OSI networking environment)[2] interface adapter (CIA)
- The HP OSI Express card driver.

ULIPC Module. This module provides an interface that allows higher-level network services such as FTAM, MMS, and X.500 to create or terminate a networking connection, send or receive data, or perform control functions. ULIPC verifies the caller's parameters and transfers the data between user memory and kernel memory. ULIPC then invokes the CONE interface adapter to carry out the requested operation.

CONE Interface Adapter (CIA). This module is responsible for the communication between the host and the HP OSI Express card. It creates and maintains connections between itself and its peer on the card, the backplane message interface (BMI).[2] These two modules pass messages back and forth regarding the creation or termination of connections and the availability of data to be transferred inbound or outbound. User data is also exchanged between these modules as part of these messages.

OSI Express Card Driver. This module and its peer on the HP OSI Express card, the backplane handler,[3] are responsible for transferring data across the HP-PB backplane. For outbound data transfers, the driver forms the data buffers passed to it into a chain of buffers suitable for DMA, and transfers these buffers via DMA to the card. For inbound transfers, the driver forms empty buffers into a DMA chain and uses DMA to transfer data from the card into the buffers. In addition to its networking interface, the driver also has an interface that handles the downloading of the executable code for the OSI protocols and diagnostic operations on the HP OSI Express card.

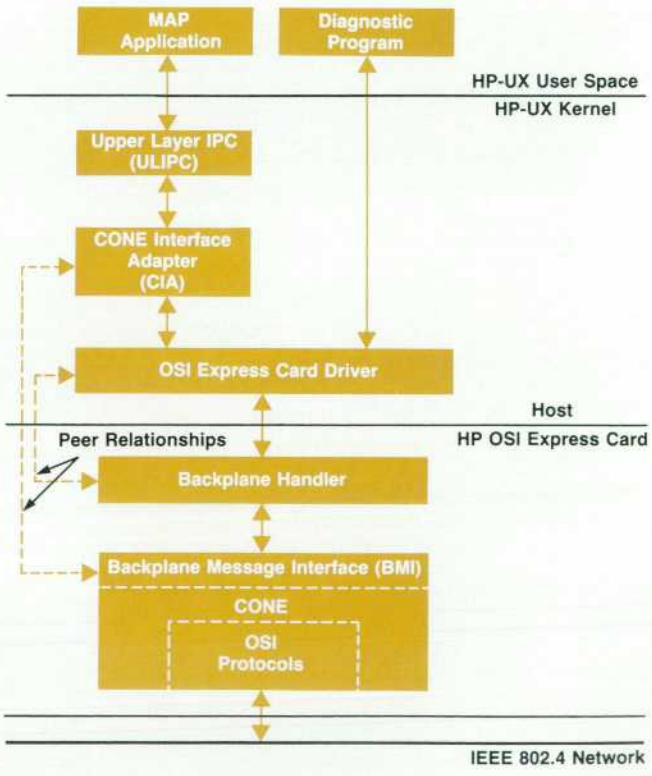


Fig. 1. *HP MAP 3.0 software block diagram showing the HP-UX kernel modules and their peer HP OSI Express card modules.*

HP-UX Networking Implementation Model

The MAP kernel modules are designed to conform with the structure and function of the existing HP-UX 7.0 kernel networking subsystem. Fig. 2 shows a block diagram of the HP-UX kernel networking model. This implementation is based on Berkeley 4.2 BSD[4] with extensions added to support proprietary HP networking interfaces such as NetIPC.[5]

As shown in Fig. 2, HP-UX user applications make either NetIPC, Berkeley IPC, or ULIPC networking system calls. Within the kernel, the IPC system calls are initially handled by code specific to the type of call. This code processes the user request and then makes calls to the socket layer to carry out the requested function.

A socket can be thought of as a communications endpoint. The socket layer provides a set of standard routines for manipulating the kernel data structures that represent a particular socket, as well as routines for sending and receiving data on that socket and performing other operations. To accomplish some of its tasks, the socket layer invokes the services of specific networking protocol modules through a protocol switch table. This table maps generic calls into protocol-specific routines. The protocol to be used for various networking services is generally fixed. In Fig. 2 the protocol switch table is used for handling NetIPC, Berkeley IPC, and ULIPC system calls. NetIPC and Berkeley IPC calls typically use the TCP/IP (Transmission Control Protocol/Internet Protocol) protocols, whereas the ULIPC calls use the CIA, which has the same entry points as the standard protocol modules. For example, a send request made by the socket layer to accomplish a data transfer on behalf of Berkeley IPC invokes the TCP/IP send routine, and a send request made by the socket layer on behalf of ULIPC invokes the CIA's send routine.

Although TCP/IP and CIA occupy similar positions in the HP-UX networking architecture, and are accessed in much the same fashion, their relative roles in accomplishing user data transfer in their respective systems are quite different. TCP and IP are traditional networking protocol modules that perform standard transport and internet layer functions such as flow control, error-free transmission, and routing. Once user data passes through these modules, it is ready for transmission by a LAN card. By contrast, the CIA's role is mainly to act as a conduit for data to and from the OSI Express card, where the transport and internet protocol modules reside. The host and card must, in effect, implement their own private protocol to set up host-card paths and to exchange data before that data can be transmitted between different hosts on the network. Some aspects of this protocol are discussed later in this article.

Upper Layer Interprocess Communication Implementation

The ULIPC provides a system-call interface that is used by higher-level host software (mainly network service modules) to accomplish data transfers. This interface bears some similarity to HP's NetIPC, with extensions added so that additional OSI requirements such as long OSI addresses and user data with connect or disconnect requests can be handled. The ULIPC interface is not exposed to users and is accessible only by the higher-level software in the HP MAP 3.0 implementation. Table I summarizes the ULIPC calls and their functions.

Many of the ULIPC calls provide an option parameter that contains different things depending on the call being made. For example, OSI addresses are passed to the HP OSI Express card using the option parameter in the ulcreate() call. There are also special routines for creating and manipulating the parameter's content. Other uses of the option parameter include:

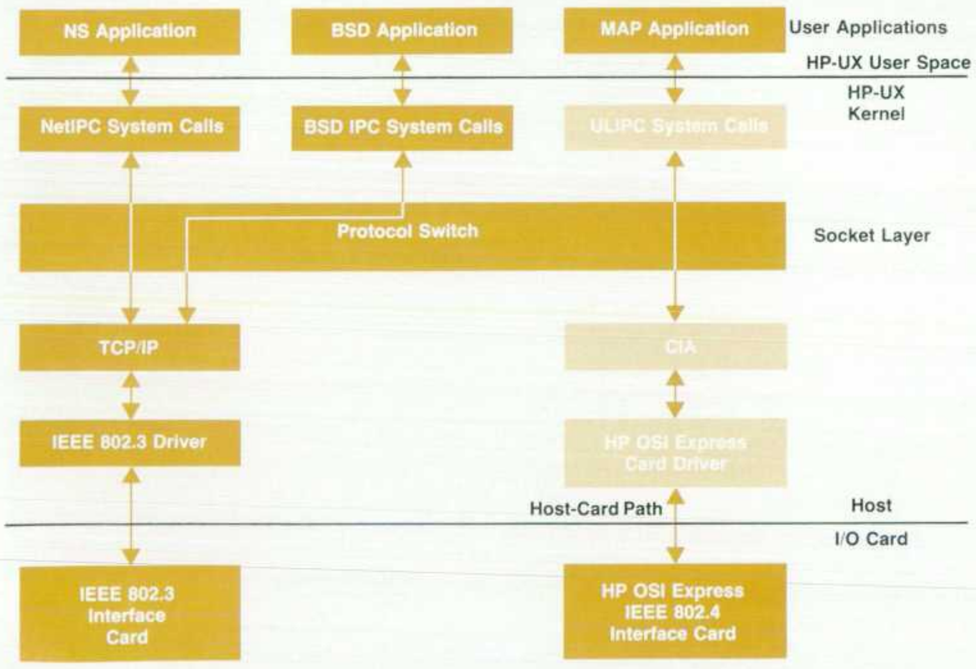


Fig. 2. *HP-UX kernel networking model.*

- Setting send and receive socket buffer sizes (i.e., the amount of data that can be buffered at the socket)
- Setting send and receive thresholds (i.e., the amount of data or space that must be available in a socket before the socket is designated as readable or writable)
- Setting time-out values for synchronous operation (i.e., the amount of time to wait in ulrecv() for data to be read)
- Sending data with a connection request.

OSI Addresses and Connections

There are two concepts that are important in understanding the implementation of ULIPC connections: the representation of an OSI address and the process of establishing a connection in a connection-oriented system. An OSI address is used to identify a given host and application in a network.[2] Fig. 3 shows a schematic representation of an OSI address. As shown, an OSI address consists of multiple subaddresses (selectors in OSI terminology) and each subaddress corresponds to a different OSI protocol layer. The OSI address is supplied to ULIPC calls through the option parameter described earlier. One of the special option parameter routines is used to insert the OSI address into the option parameter before making the call.

Table I
Upper Layer IPC Calls and Their Functions

ULIPC Call	Function
ulcreate	Create a call socket (communication endpoint)
ulconnect	Initiate connection establishment
ulcontrol	Perform various special functions on a socket or connection
uldest	Create a destination OSI address descriptor
ulrecv	Receive data on a connection or OSI connect request
ulrecvcn	Receive the connection indication for a call socket
ulselect	Determine the status of a socket
ulsend	Send data on a connection
ulshutdown	Terminate a connection

OSI applications establish connections with each other by exchanging protocol data units (PDUs) for their layers in a specific sequence. A PDU can be thought of as a message that contains control information and possibly user data for a specific layer. Ignoring implementation details, the basic sequence of events for creating a connection is:

- The application initiates a connection by sending a connect request PDU to the remote application with which it wishes to make a connection.
- The PDU arrives at the remote host and generates a connect indication event, which is processed by the remote application.
- If the remote application wishes to accept the connection, it responds by sending a positive connect response PDU to the initiating application.
- The PDU arrives at the initiating host and generates a connect confirm event, which is processed by the initiating application. The OSI connection is now established between the two applications.

ULIPC Connections

For HP MAP 3.0 an OSI connection between two applications using ULIPC relies on the use of sockets.[6] Sockets are a widely used mechanism in networking interfaces for establishing connections. The ULIPC interface uses three types of sockets: call sockets, virtual circuit sockets, and destination sockets. Call and virtual circuit sockets are the most commonly used socket types. Call sockets are used to establish the addresses of applications, whereas virtual circuit sockets are used to reference individual connection endpoints. Destination sockets are used in special instances to hold address information for remote applications.

The process for establishing an OSI connection using ULIPC is illustrated in Fig. 4. This figure shows two OSI applications, A and B, which might be located on different hosts on the network. While it is possible for HP MAP 3.0 nodes to communicate with MAP nodes from other vendors, for purposes of this example, both A and B are assumed to reside on HP MAP nodes. We will also assume that application A wants to establish a connection with application B.

For application A to be able to connect with application B, both applications must create call sockets using ulcreate(). This is accomplished by both applications inserting their OSI addresses into the option parameters and then calling ulcreate() on their respective systems. The ulcreate() call registers the address of the OSI application on the HP OSI Express card, creates an associated call socket, stores the address in this socket, and returns a descriptor to the caller. The descriptor is an integer value used for subsequent references to the socket. With the address registered on the card, remote applications can now attempt to make connections to the registered application. Fig. 4a shows the action of the ulcreate() calls by both A and B.

The next step is for application A to set up the structure for initiating a connection with application B (see Fig. 4b). To accomplish this, application A uses application B's application entity title, or the name of application B, to determine B's OSI address. Application A can determine the OSI address of B by accessing a directory using the directory services of X.500. X.500 in HP MAP 3.0 is described on page 15. Once application A has B's OSI address, it calls the function uldest(). Uldest() creates a new destination socket

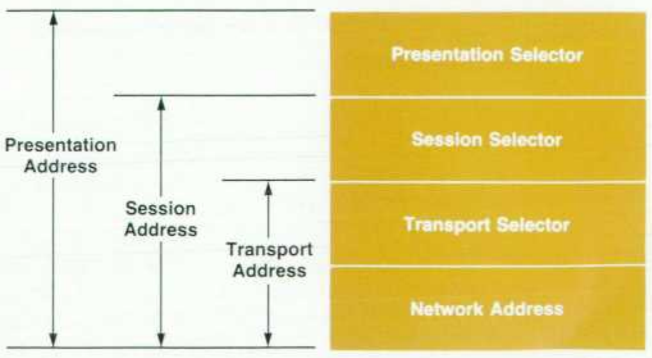


Fig. 3. *The parts of an OSI address.*

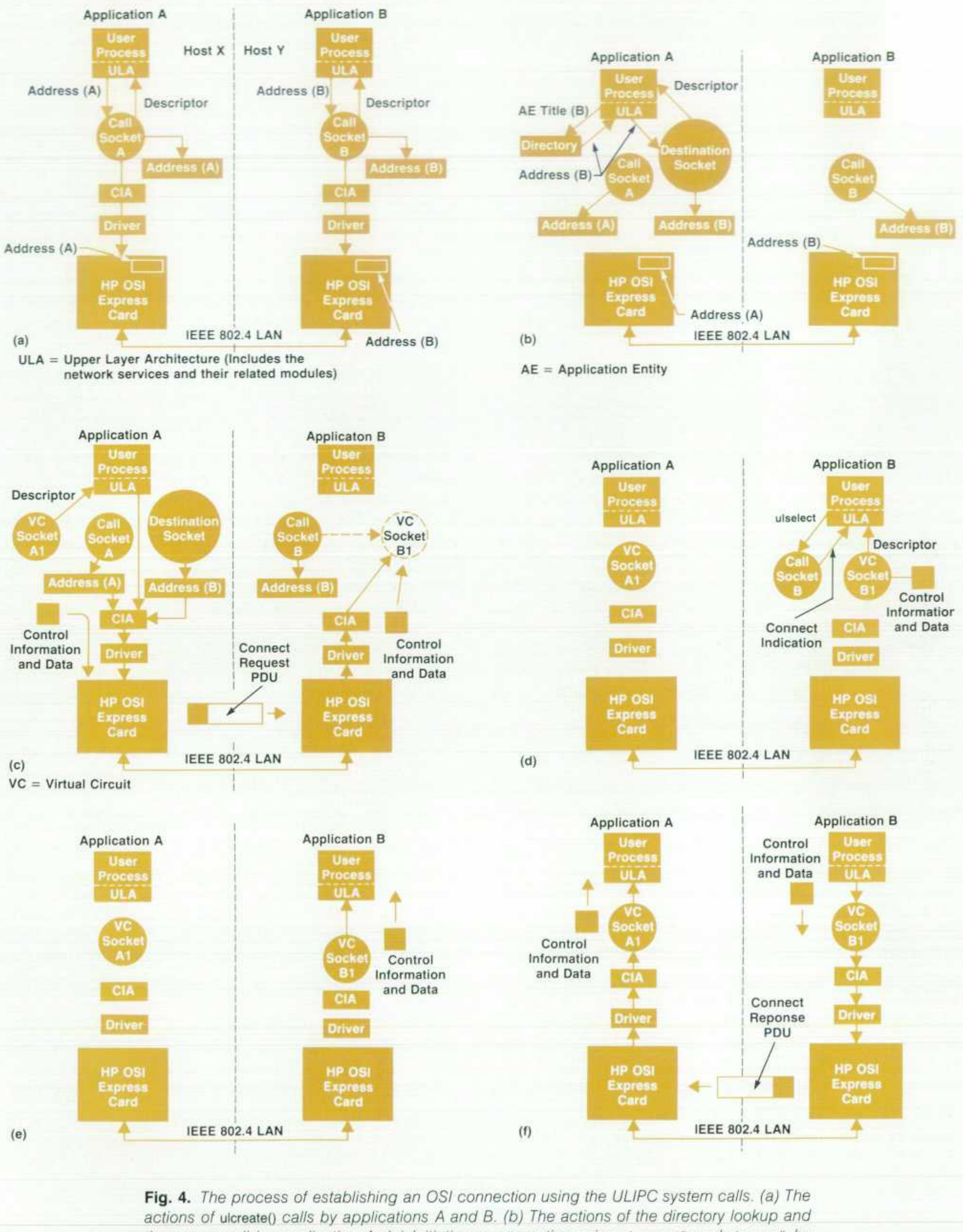


Fig. 4. *The process of establishing an OSI connection using the ULIPC system calls. (a) The actions of* ulcreate() *calls by applications A and B. (b) The actions of the directory lookup and the* uldest() *call by application A. (c) Initiating a connection using* ulconnect() *and* ulrecvcn() *by application A. (d) A fledgling connection is established between A and B with the* ulselect() *and* ulrecvcn() *calls by application B. (e) Application B makes a* ulrecv() *call to receive the control information and data sent from A. (f) A connection is fully established between A and B through a* ulsend() *call by B and a* ulrecv() *call by A.*

structure to hold the address information it is passed, and returns a destination descriptor that references this socket. This descriptor is used when making subsequent ULIPC calls that need to refer to application B.

Now application A initiates the connection to application B (see Fig. 4c). First, A inserts any control information and user data to be sent with the connect request PDU into the option parameter. This includes information that the host-resident protocol layers, including the application, need to exchange to establish a connection. Next A calls ulconnect(), supplying A's call socket descriptor and B's destination descriptor, along with the option parameter. Ulconnect() creates a virtual circuit socket (A1) for the connection, and then returns to its caller a descriptor for this socket. The host-control information and the user data travel through the card-resident protocol layers, with each layer adding its own control information to form a connect request PDU, which is sent to the destination.

Having initiated the connect request, A now calls ulrecv() to await a connect confirmation event indicating that B has accepted the connection. (Asynchronous modes of operation that don't require waiting are also possible).

When the connect request PDU sent by A arrives at B's host, a connect indication event occurs. The control information for the lower-layer protocols is removed from the PDU by the card, leaving the host control information and the user data sent by A. This data is passed to the host, where it is attached to an unreferenced virtual circuit socket (B1) that the CONE interface adapter (CIA) creates for the connection, pending its final acceptance. The virtual circuit sockets in Fig. 4c are labeled with number suffixes (B1 and A1) to signify that the connections these sockets represent could be the first of many connections to be created between the two applications.

To determine if an incoming connection indication event exists at its call socket, B calls ulselect(), which is used to detect the presence of connect indication events at call sockets or data on virtual circuit sockets. If ulselect() indicates the presence of a connect indication at B's call socket, B then calls ulrecvcn(). Ulrecvcn() completes the creation of the virtual circuit socket for the connection on B's host and returns a descriptor for the socket to application B (see Fig. 4d). Note that the host control information and user data have not yet been received by application B. At this point a fledgling connection has been established between A and B (enough so that OSI PDUs can be exchanged between the two applications). However, the full OSI connection is not yet completed until the required PDUs are exchanged between A and B.

For the OSI connection to be completed, B must first do an explicit ulrecv() call to receive the control information and data that were sent by A's ulconnect() call. Any user data sent with the connect request is also received in this call. Fig. 4e shows B receiving these items using ulrecv().

Application B can now determine if it wishes to accept the connection. If B accepts the connection, it sends a positive connect response PDU back to A using ulsend() in the same manner described previously with ulconnect(). When B's positive connect response PDU arrives at A, it causes a connect confirmation event. This event causes application A's ulrecv() call to complete, and the host-control information and user data sent by B are passed to application A. The OSI connection is now fully established.

This example shows that the kernel modules have little OSI-specific knowledge. They are used primarily to establish lower-level paths between the host and the HP OSI Express card over which PDUs can flow to remote applications. They provide mechanisms for sending and receiving PDUs, but do not themselves have any knowledge of the PDU content being sent or received. The exchange of PDUs to establish a connection is orchestrated by higher-level software, in this case the applications in conjunction with the upper-layer network services.

If this example is changed to assume that application B is running on a non-HP MAP node, nothing is changed from the standpoint of application A, which will still issue ULIPC calls in the sequence shown in the example to initiate a connection with B. On the B node, the implementation details of how B handles connect indication events and sends connect response PDUs would be vendor-specific and differ from the example, but the resulting flow of PDUs between the two systems would be the same. The kernel communication modules, along with the HP OSI Express card facilities, allow standard OSI communication between interoperable systems from any vendor.

Once an OSI connection has been established between applications, users can send and receive data on the connection using ulsend() and ulrecv(). The ULIPC interface supports the use of vectored I/O, which lets users pass messages in both directions as an array of pointers to buffers containing parts of the message. This saves users from having to do extra data copies to send OSI messages.

A special ULIPC call, ulcontrol(), is used for performing special functions on a socket or a connection. For example, it can be used to set individual socket parameters such as buffer sizes, thresholds, and time-outs.

CONE Interface Adapter

The CONE interface adapter, or CIA, is responsible for establishing a communication path for a network connection between the host and the HP OSI Express card, and for sending and receiving user data on this path (see Fig. 1). This host-card path must exist to create a full OSI connection between applications. However, it differs from a full OSI connection in that this host-card path exists before the OSI connection is fully established and continues to exist until after the OSI connection is terminated. (For the rest of this section we will ignore this distinction and simply use the term connection to refer to both the host-card path and the complete OSI connection.)

Upper Interface

As we have already seen, the CIA provides an interface that is similar to the interface provided by networking protocol modules such as TCP/IP in HP-UX (see Fig. 2). Interface users such as the socket layer make requests to the CIA on behalf of ULIPC system calls by calling the CIA's upper interface entry point with several parameters. One parameter is the request code that tells the CIA what operation to perform, such as PRU_SEND, which tells the CIA to send data on a certain connection, and PRU_ABORT, which tells the CIA to close the host-card connection. Other parameters are a chain of buffers containing the user's data and a pointer to the connection's associated socket data structure.

CIA-to-BMI Message Communication

The CIA and its HP OSI Express card counterpart, the backplane message interface (BMI), work together to establish communication paths across the host-card boundary. Since these two modules execute in separate environments and cannot communicate via procedure calls or shared data structures, a messaging scheme was designed for communication between them. Example messages that the CIA can send to the BMI and the information they convey include:

- Create. Tells the BMI to expect data on a specified connection in the future.
- Add_SAPS. Registers a call socket so that the BMI will send a connect indication for a certain OSI address to the associated call socket.
- Send. Sends user data to the BMI.
- Destroy. Tells the BMI to break a specified connection.

The BMI can also send messages to the CIA if the card needs to initiate activity in the host, as in the case of an incoming connect indication being received.

Many of the messages that can be exchanged between the CIA and the BMI direct the modules to perform operations that take time to perform. Therefore, most messages have corresponding handshakes that the message sender must wait for before assuming the operation has been performed. This handshaking keeps the operation of the CIA and the BMI in synchronization on both sides of the backplane. The CIA maintains state variables internally to ensure that messages are sent only when appropriate.

Register Sets

The CIA uses specific register sets on the HP OSI Express card for setting up connections and transferring messages or data on connections. A register set is a collection of 16 32-bit registers that can be read from or written to by the host to start a DMA transaction between the host and the card.[3] The host can address several hundred separate register sets independently and start DMA transactions on each concurrently. Therefore, a register set can be thought of as either a host-to-card (outbound) or card-to-host (inbound) path depending on how it is being used. The allocation of register sets for various host-card functions is given below.

Register Set Number	Function
0	Downloading and Diagnostics
1	Transparent Indication Register Set
2,3	Management Register Sets
4,5	Expedited Data Register Sets
6-17	Reserved
18,19	Path Register Set Pair #1
20,21	Path Register Set Pair #2
22,23	Path Register Set Pair #3
.	.
.	.
.	.
216,217	Path Register Set Pair #100

Register Set 0. This register set is used for downloading the HP OSI Express card software and for sending and receiving diagnostic messages and data to or from the card. It is not used for normal networking operations.

Register Set 1. This register set is used to read transparent indications sent from the card to the host. A transparent indication is used to help provide card-to-host (inbound) flow control.

Register Sets 2 and 3. These are management register sets used by the CIA and the BMI to send messages to each other about connections that are not yet established. The even-numbered register set in this and other register set pairs is always used for host-to-card message transfers, and the odd-numbered register set is used for card-to-host transfers. When the CIA and the BMI want to set up a host-card path for a networking connection, they first exchange messages on the management register sets.

Register Sets 4 and 5. These register sets are used to send and receive expedited data between the host and the card. Expedited data is a type of data defined in the OSI protocols whose transmission sometimes takes priority over normal data.

Transparent indications and expedited data are described in more detail later in this article.

The remaining register set pairs are used for host-card paths across which user data flows. Each path register set pair supports one full-duplex network connection. There are 100 such pairs and therefore 100 connections are suppported. The path register set pairs are allocated to connections as connections are established. The fact that each connection maps one-to-one with a register set pair means that connections can be handled independently and one connection cannot block another.

Sending and Receiving Messages and Data

The CIA sends and receives messages (some of which contain user data) across various register sets to and from the BMI using the OSI Express driver. The register set used depends on the situation and the type of message being sent. To start a host-card path for a network connection, the CIA and the BMI exchange create messages and handshakes on the management register sets. These messages instruct both sides as to which path register set pair to use for the connection. When user data is to pass between the card and the host, the CIA and the BMI exchange send messages (which contain user data) between them on the appropriate path register sets. Finally, when a host-card path is to be deallocated because the network connection is being terminated, the CIA and the BMI exchange destroy messages and handshakes on the management register sets to effect the deallocation of the appropriate path register sets and to clean up connection resources in the host and the card.

OSI Express Card Driver

The OSI Express card driver transfers messages and user data to and from the HP OSI Express card across the HP-PB backplane. In addition, the driver can be used to exchange diagnostic information with the card. The driver uses direct memory access (DMA) transfers to move data between the host and the card. A detailed discussion of DMA transfers between the host and the HP OSI Express card is given in reference 3.

To transfer data using DMA, the driver forms the data into a DMA chain. A sample DMA chain is shown in Fig. 5. A basic DMA chain is made up of a linked list of quads (four 32-bit values) to which data buffers can be attached. Each quad except the last (the link quad) contains information about the direction of the DMA transfer to be performed, an attached data buffer, and a byte count of the data. The link quad denotes the end of a DMA chain and references the DMA completion list entry for a transaction. The completion list entry is a memory area that the card updates with the status of the completed transfer. In addition, the link quad can reference more DMA chains.

Networking

To transfer data to the HP OSI Express card, the driver allocates resources for the connection and then sends or receives messages and data using DMA.

Allocation and Deallocation of Register Sets. When a connection is brought into service, the CIA and the BMI allocate path register sets and other resources to the connection. The CIA also calls the driver so that the driver can allocate its own resources for the connection. When this call is made, the CIA passes the driver pointers to a set of procedures called *completers*. These routines, which are associated with each connection, are called by the driver whenever certain events occur on the connection. The three types of completer routines are read completers, transparent indication completers, and error completers. These routines reside in the CIA.

When register sets are deallocated, the CIA, in addition to sending BMI messages, makes appropriate calls to the driver so that the driver can deallocate the resources that had been allocated for the connection.

Sending and Receiving Messages and Data. When the CIA wishes to exchange messages and data with the BMI, it makes read or write requests for the desired register set to the driver. The CIA passes to the driver a list of memory buffers to use for the transfer. These buffers contain user data in the case of writes, and space for inbound data in the case of reads. Also, each buffer contains space for the DMA quad required to form the DMA chain. This allows the driver to avoid the additional effort of allocating memory for quads. The driver forms the memory buffers into a DMA chain similar in structure to that shown in Fig. 5. Then, for an outbound transfer, the driver starts the DMA transfer on the outbound path register set (even-numbered registers) associated with the connection. For an inbound transfer, the driver starts the DMA transfer on the connection's inbound path register set (odd-numbered registers).

The driver allows the CIA the flexibility of starting simultaneous DMA transfers on multiple connections or on connections with transfers in progress without having to wait for the completion of existing requests. If a transaction is in progress, the driver will queue a newly requested transaction for that register set behind it.

When a DMA transfer is completed by the card, the card fills in the completion list memory area pointed to by the link quad with information about the transaction type and its status. It also adds this completion list entry to the host's DMA completion list and generates a host interrupt. A special I/O service routine in the host kernel scans this completion list and invokes the driver for each completed transaction. For completed read transactions, the driver calls the read completer routine associated with the connection. The read completer routine puts the data read on the inbound socket buffer for the connection so that it can be read by ULIPC calls. For completed writes, the driver simply frees the memory buffers used in the transfer and updates the connection's outbound socket buffer.

The transparent indication and its associated completer routines provide a special mechanism for inbound (card-to-host) flow control. A transparent indication is simply a message that tells the host that data is available for reading from the card. The message contains the number of the connection on which the data is available and the quantity of data. Register set 1 is always allocated for receiving transparent indications from the card into the host, and the driver always keeps a small number of read transactions posted on this register set. If the driver receives a transparent indication on the special register set, it calls the transparent indication completer routine for the connection specified in the transparent indication. This completer routine checks whether enough memory exists in the host to read in the data. If so, the CIA calls the driver to read the data; otherwise it waits until memory is available. The

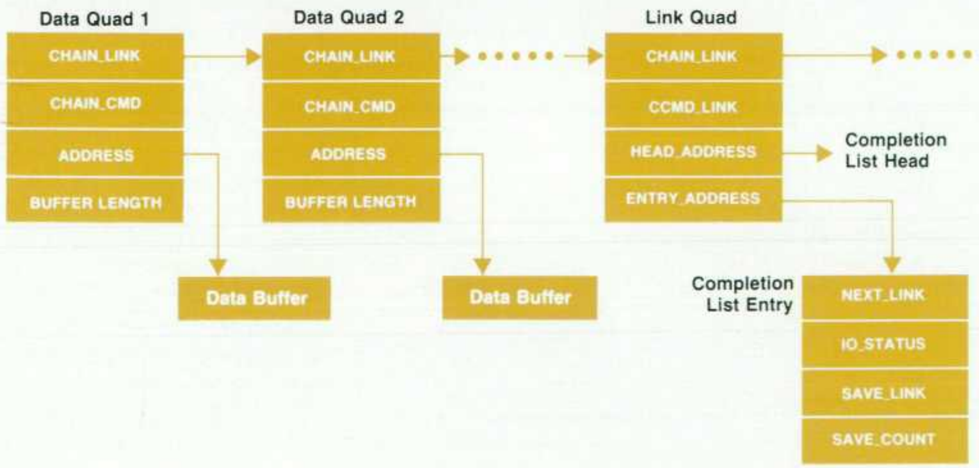


Fig. 5. *A DMA chain.*

card will not send another transparent indication for the same connection until the CIA has called the driver to read the data for the indication already received.

Fig. 6 depicts the situation that could exist between the host and the card if two applications in the host had each established a single connection to an application elsewhere on the network. Application X's connection is using register sets 18 and 19 to transfer data between the host and card (and ultimately between two communicating hosts), with register set 18 being used for outbound traffic and register set 19 for inbound traffic. At the time this situation occurs, the connection has one outbound and one inbound DMA data transfer in progress.

Application Y's connection uses register sets 22 and 23. This connection has multiple outbound data transfers in progress. This could be the case if the remote host is for any reason slow in receiving the data.

Also shown in Fig. 6 are transparent indications being sent by the card to the host to indicate the presence of data to be read on the card. The figure shows the first transparent indication being received on behalf of application Y, and the second by Application X. Presumably, both applications would post subsequent reads to receive this data, which would execute directly or queue after the read transactions depending on the system timing.

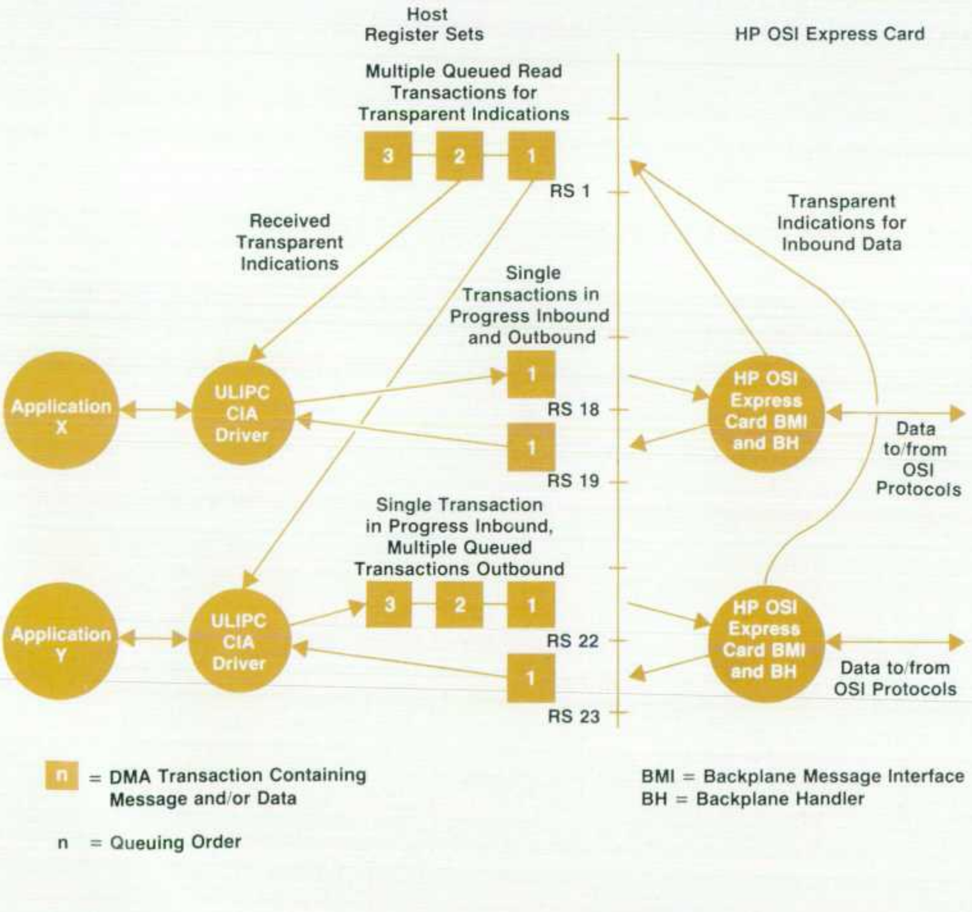
Diagnostic Use

Nodal management and diagnostic tools can call the driver directly from user space to download software to the HP OSI Express card or to exchange diagnostic information with the card. Examples of diagnostic operations the card can perform are self-testing and uploading of memory contents to the host for debugging.[7]

The driver supports the standard HP-UX system calls open(), ioctl(), read(), write(), and close() for diagnostic access to the card. The open() call makes the driver and card available for diagnostic access. The ioctl() call tells the driver which type of diagnostic transfer to initiate on a subsequent read() or write() operation. When the driver's caller invokes read() or write(), the driver builds a DMA chain appropriate to the diagnostic request to be performed, referencing the data buffers to be read or written. The driver then starts the DMA transfer and waits for its completion. After the operations have been performed, the close() call relinquishes the driver for diagnostic access.

Kernel Module Data Structures

Several data structures are used within the kernel modules to maintain the state of the many activities associated with a connection. Fig. 7 shows the key individual connection data structures. The top-level structure is the socket structure. This structure stores socket state information, and also references two queues (socket buffers) for inbound and outbound user data. When an application sends data using ulsend() calls, ULIPC moves data from user space into the socket's outbound socket buffer. Similarly, incoming data for the connection is queued on the socket's inbound socket buffer and can be read from there using ulrecv() calls. The state information and queues are updated as data flows



Host Register Sets
HP OSI Express Card
Multiple Queued Read Transactions for Transparent Indications
3
2
1
RS 1
Received Transparent Indications
Transparent Indications for Inbound Data
Single Transactions in Progress Inbound and Outbound
Application X
ULIPC CIA Driver
1
RS 18
1
RS 19
HP OSI Express Card BMI and BH
Data to/from OSI Protocols
Single Transaction in Progress Inbound, Multiple Queued Transactions Outbound
Application Y
ULIPC CIA Driver
3
2
1
RS 22
1
RS 23
HP OSI Express Card BMI and BH
Data to/from OSI Protocols
n = DMA Transaction Containing Message and/or Data
n = Queuing Order
BMI = Backplane Message Interface
BH = Backplane Handler

Fig. 6. *Transactions in progress between the host and the HP OSI Express card on multiple register sets. In this example, there are two full-duplex connections with each connection in use by a separate application.*

through the connection.

The next data structure is the CIA control block. This data structure is used to hold information about the current state of the host-card path for a connection. Contained in this state information are a socket pointer and a register set value identifying the backplane register sets to be used for the connection. The control block can also reference queues of outbound expedited data and user data from connect requests.

At the driver level, state information is kept for each register set pair in use, as well as for each register set in the pair. The register set pair information contains the completer routine pointers for the register set pair as well as the overall state of the pair (e.g., is the pair allocated or free). For each register set in the pair, the driver stores the head and tail pointers for the DMA chain being transferred.

Special Kernel Module Features

Several features provided by the kernel modules enhance the efficiency of network transfers. Some of these features include the ability for more than one application to use the same call socket, expedited transfer of high-priority data flows, and guaranteed delivery of inbound data.

Call Socket Rendezvous

One of the special features that the ULIPC and the CIA provide is the ability for two or more unrelated user programs to wait for connection indications on the same call socket. This ability makes it easier for server processes to handle several indications at once.

The CIA with help from the ULIPC allows unrelated processes to share the same call socket if they both specify identical OSI addresses. When the first call socket is created, the CIA sends a message (Add_SAPS) to the BMI with the OSI address included. Assuming that the card has enough resources with which to register the address, it will send back a positive response to this message along with a special value that the CIA will associate with the first call socket. When the second call socket is created by the second process and the Add_SAPS message is sent to the BMI with the same OSI address as for the first process, the card software notices that the two addresses are the same and sends the response back with the special value for the first call socket. The CIA realizes from this that the second socket is a duplicate of the first, and tells the ULIPC that the two sockets should be merged. The call socket descriptor provided to the second process will therefore reference the same call socket as that given to the first process.

Expedited Data

In the OSI environment, expedited data is data that sometimes must be sent or received with a higher priority than normal data and not be subject to normal flow control. A message from an application to abort a connection is an example of data that usually receives expedited treatment.

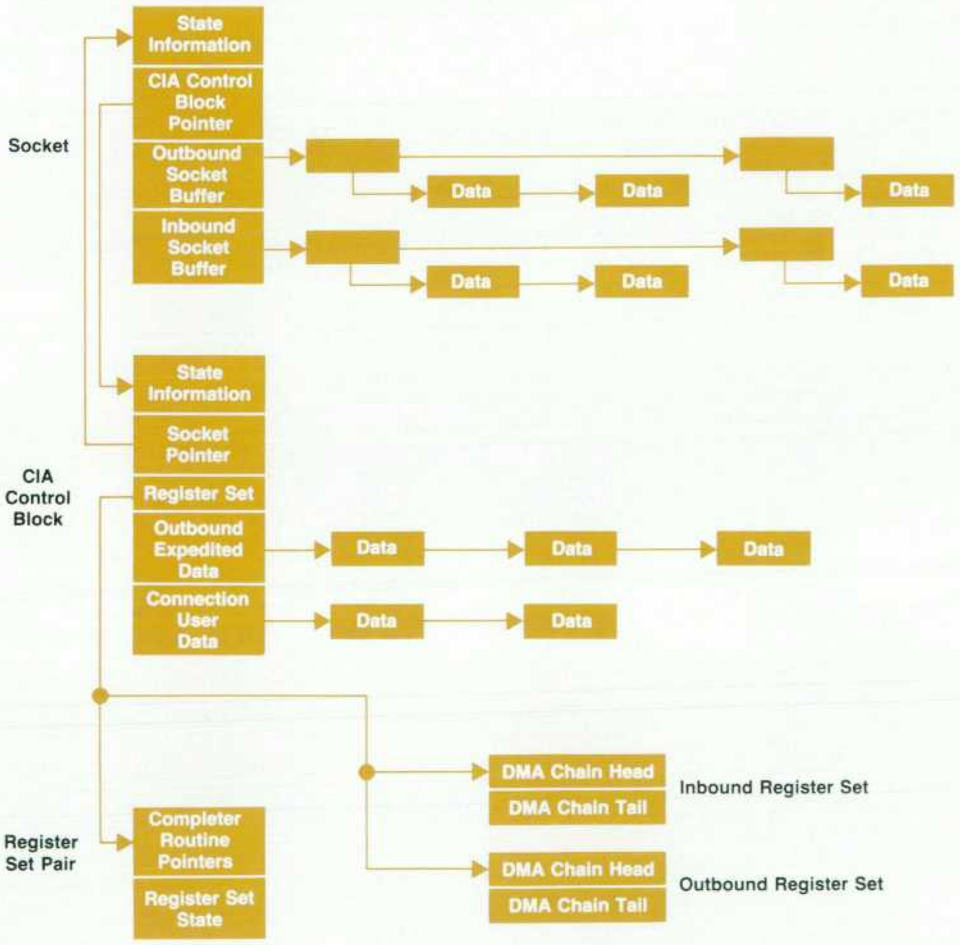


Fig. 7. *Data structures for an individual connection.*

The kernel modules make special provisions for this type of data. First, dedicated register sets are allocated for receipt and transmission of expedited data so that transmission of this data is not impeded by pending transfers of normal data. Then, at the CIA and the ULIPC level, inbound expedited data is marked so that it can be distinguished from normal data. When inbound expedited data arrives, the data on the inbound socket buffer is reordered so that all expedited messages come before normal data. The socket reservations (maximum amounts of data allowed to be buffered at the socket) are also increased so that the flow of expedited data will not be impeded by normal data.

Outbound expedited data bypasses normal data stored on the outbound socket and is either sent immediately or stored (if the BMI is not yet ready to receive it) in the CIA control block for the connection. Therefore, outbound expedited data, as well as inbound data, always bypasses the flow control mechanisms of normal data.

Guaranteed Delivery of Inbound Data

In networking implementations where the transport protocol resides in the host, the transport layer has access to the latest status of resources in the host, particularly in the area of memory availability. If the status is unfavorable for receiving inbound data, the transport layer will know this and can take appropriate steps to compensate by doing such things as discarding inbound data. This assumes that the remote side will likely retransmit the data and that the unfavorable status is only a short-term condition.

However, when the transport protocol resides outside the host, as is the case for this implementation, it must receive data with no particular knowledge of or guarantee about the host's ability to receive this data. If host conditions are unfavorable for receiving the data, the data cannot simply be discarded because it has already been acknowledged to the transmitter as having been received, and will therefore not be retransmitted. Therefore, data received by the card transport protocol must be guaranteed successful passage into the host.

The approach taken to this problem was to examine the possible causes of host inability to receive data, and adjust kernel configuration parameters (in particular, networking memory reservations) for the implementation to ensure that they could not occur under worst-case conditions.

Acknowledgments

Many people contributed to the modules described in this article. At HP's Information Networks Division, Allwyn Sequeira, Paul Gradin, Greg Marino, Doug Chan, Ali Golshan, Collin Park, and Frank Fiduccia all contributed to the design, development, and test of these modules at various stages of the project. Tom Robins did the early driver design and helped with the development of this article. Ron Mok helped characterize ULIPC performance. From HP's Roseville Networks Division, Bill Gilbert, Jim Haagen-Smit, Dave Kumpf, Glenn Talbott, and Mike Wenzel made contributions. Our thanks to everyone above, and to Greg Gilliom, Jean Yao, and Atul Bhatnagar for their management support.

References

1. *Hewlett-Packard Journal*, Vol. 41, no. 1, February 1990, pp. 6-77.
2. S.M. Dean, et al, "CONE: A Software Environment for Network Protocols," *Hewlett-Packard Journal*, Vol. 41, no. 1, February 1990, pp. 18-28.
3. G.F. Talbott, "The HP OSI Express Card Backplane Handler," *Hewlett-Packard Journal*, Vol. 41, no. 1, February 1990, pp. 8-18.
4. S. J. Leffler, et al, *4.2BSD Networking Implementation Notes*, Computer Research Group, Department of Electrical Engineering and Computer Science, University of California, Berkeley, Revised July 1983.
5. R. J. Carlson, et al, "HP AdvanceNet: A Growth-Oriented Computer Networking Architectural Strategy," *Hewlett-Packard Journal*, Vol. 37, no. 10, October 1986, pp. 6-10.
6. S. J. Leffler, et al, *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley, 1989.
7. J. K. Shah and C. L. Hamer, "Support Features of the HP OSI Express Card," *Hewlett-Packard Journal*, Vol. 41, no. 1, February 1990.

Interoperability Testing for HP MAP 3.0

Interoperability testing is used to ensure that HP MAP 3.0 OSI services can communicate with other vendors' systems and to uncover errors both in HP's and other vendors' OSI implementations.

by Jeffrey D. Meyer

ONE OF THE PRIMARY objectives of HP's MAP 3.0 offering is to allow HP systems to communicate with those of other system vendors. To be successful in meeting this objective requires interoperability testing in addition to standard software testing practices such as module, system, and reliability testing. Interoperability testing is the verification of the ability of different network implementations to communicate. This type of testing helps to expose implementation errors in both HP and other vendors' systems, and most important, it helps to ensure that we provide our customers with a truly open system.

Need for Interoperability Testing

The large number of network standards, their relative newness, and the existence of many options within the standards makes the implementation of a full OSI stack a formidable task. Errors can be made both in the selection and enforcement of options and in the general encoding and decoding of the protocol data units that are sent across the network. These types of errors may go undetected during testing between similar systems because they can be canceled. Canceling happens when an error is present in both the sending and receiving code, the net effect being that no error is detected.

Fig. 1 illustrates this problem. In this case implementation A incorrectly requires the presence of optional field Y in a PDU. During the testing of A against itself this error is not detected because A always includes field Y. When A tests with implementation C, which does not include this field, the error is exposed. Fig. 1 also illustrates that interoperability testing is not transitive. A tests with B successfully because B includes the optional field. Also B tests successfully with C because B does not require the optional field Y. But A fails the test with C.

Because of the canceling effect, an OSI product can go through its internal software testing cycle and still contain significant undetected errors. Because of the lack of transitivity we cannot make assumptions about our ability to operate with one vendor based on our experience with another. For these reasons interoperability testing is an important and necessary element of the test cycle for OSI products.

Conformance Testing

Another type of testing that exposes defects in OSI implementations is conformance testing. Conformance testing consists of running a set of communication tests between the system being tested and a reference system. The idea here is that the reference system is a correct implementation of the OSI layers being tested and that successful completion of the conformance tests will indicate that the tested system's implementation is also correct (conformant). Tests for OSI implementations are administered by a national agency. In the United States this is the Corporation for Open Systems (COS), which issues certification marks to systems passing the tests.

Conformance tests may eventually replace the need for most interoperability testing, but this is not the case today. The main problems existing are the availability and stability of tests and the commitment of vendors to undergo testing. Because the FTAM conformance tests are still under development and there are no accepted MMS tests in the United States, we must perform interoperability testing.

HP is taking an active role in participating in the COS's conformance testing. We have completed IEEE 802.4 and OSI Transport Class 4 tests and are working with the COS on FTAM testing. Because the conformance tests are also subject to errors in implementation, the participation of other vendors in conformance tests is essential to their becoming stable. As stable conformance tests become avail-

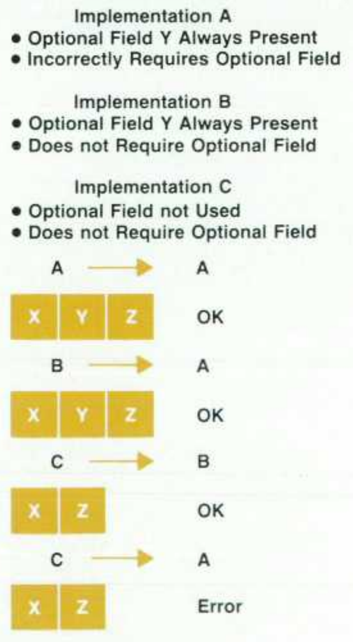

Implementation A
• Optional Field Y Always Present
• Incorrectly Requires Optional Field
Implementation B
• Optional Field Y Always Present
• Does not Require Optional Field
Implementation C
• Optional Field not Used
• Does not Require Optional Field
A → A
X Y Z
OK
B → A
X Y Z
OK
C → B
X Z
OK
C → A
X Z
Error

Fig. 1. *An example of how an error can be canceled between systems.*

able and are widely accepted among vendors, we should see the following benefits.

- The tests will be very rigorous and produce a range of behavior much greater than that generated by a typical implementation. This should enable conformance testing to expose the majority of implementation errors.
- The tests will be improved over time to incorporate tests for implementation errors that might still be found between conformant systems.
- The rigorous and improved tests will mean that a system's conformance mark will provide a high level of confidence in its ability to interoperate with other conformant systems. This will reduce the need to perform controlled interoperability testing between each pair of implementations.

The Interoperability Test Process

Although interoperability testing could be performed by purchasing the other vendor's equipment and implementing and running the tests in-house, experience has shown that our most effective interoperability testing has resulted from cooperation between HP and the other vendor. The cooperative approach speeds the test development cycle and it improves the ability to diagnose problems detected in the other vendor's equipment. If problems are found in the other vendor's equipment, their involvement improves the turnaround time for fixes.

Fig. 2 shows the interoperability testing process that has been established at HP. The process is initiated by an HP field representative after a customer request has been received for an HP OSI network product. The other system vendors on the network are reviewed against the systems we have already tested. If any have not been tested then the testing process begins. The testing process may also be initiated by R&D or marketing for vendors our customers might use in the future.

The first phase in the testing process is determining the availability of personnel within the factory to perform the testing. After identifying the internal team, contact with the other vendor is established for carrying out the testing. In some circumstances the customer may also be involved.

When the teams are established the next step is to exchange information about each vendor's implementation. The information exchanged consists of a document called a *Protocol Implementation Conformance Statement* (PICS) and, if available, network traces showing the protocol data unit encodings produced by the vendor's stack. The PICS describes the services and options supported for a protocol. Each of the ISO protocol standards specifies the information to be provided in the PICS, which usually takes the form of tables to be filled in. The PICS information is used to determine the functionality to be tested. The PICS information can also be used to determine if the functionality requested by the customer is met by the two implementations. The network traces from the other vendor can be compared with those of our implementation to identify differences and to look for known problems.

After this exchange of information takes place, both parties are prepared to decide on the appropriate test cases to be executed and the location where the tests are to take place. HP has a set of abstract test cases for both FTAM and MMS, which are generally proposed as part of the test suite. An abstract test case describes the test purpose, the steps to execute the test, and the expected results.

In addition to testing at one of the vendor's facilities, two other options exist for test location. Testing can be performed over a wide area network by using a router to move data from each vendor's local IEEE 802.4 network to an X.25 public data network. An advantage of this type of testing is that both parties can work out of their respective labs, giving them easy access to the development and diagnostic tools available. Testing may also be performed at the COS's interoperability test lab. COS, the agency that performs conformance testing, has provided floor space in their lab for vendors to leave their equipment. For about a week every quarter vendors are invited to come to the lab to carry out interoperability testing. An advantage of this environment is that testing can be performed with more than one vendor.

Install, configure, and verify are combined and listed as a separate step because when equipment is moved for the purposes of this testing, it is important that its local functioning be verified first. This avoids wasting time diagnosing problems that have nothing to do with interoperability.

After all the preceding activities have been completed, the actual execution and interpretation of the interoperabil-

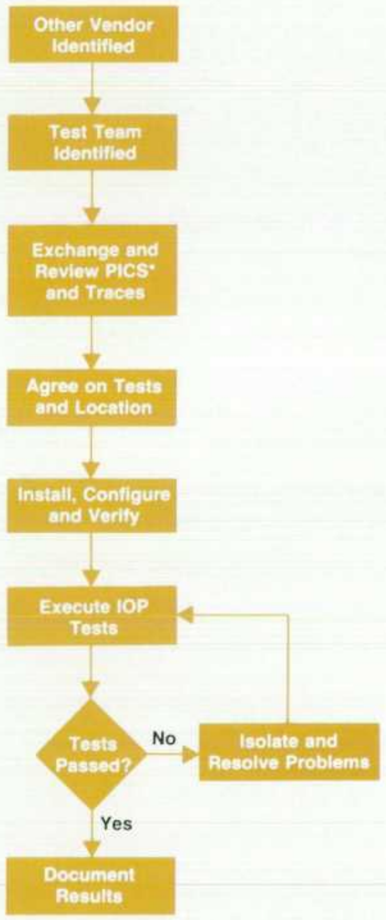

Other Vendor Identified
Test Team Identified
Exchange and Review PICS* and Traces
Agree on Tests and Location
Install, Configure and Verify
Execute IOP Tests
Tests Passed?
No
Isolate and Resolve Problems
Yes
Document Results

Fig. 2. *The interoperability testing process.*

ity tests can take place. The evaluation of the failed tests should involve both vendors. It is useful to gather as much information as possible, including traces of the dialog, error messages reported, errors logged, and configuration data. After diagnosing a problem, ownership is assigned and the problem corrected. If fixes are not readily available, both vendors may agree to proceed with other tests if they are confident that the error exposed will not affect the outcome of the other tests. In fact, it is useful to define tests that are loosely coupled, that is, test each component of the network service with little dependence on other functions. This ensures that testing can proceed in parallel with fixes being developed for exposed problems.

When testing is complete an entry is made for that vendor into an on-line information base. This entry describes the tests that were performed, and for those that failed, the symptoms and type of problem, the owner of the problem, and the fix. In addition to the test results, these entries also give information about the vendor's equipment, the personnel involved in testing, the time and location of testing, and the diagnostic tools available on the other vendor's equipment. This record can then be used for evaluating future field requests for vendor support as well as for tracking trends in interoperability problems.

Interoperability Results

The most encouraging result to come from our interoperability testing is the high level of cooperation we have had from other vendors. One reason for this cooperation is that all parties involved gain from this testing. Both vendors have the opportunity to improve their implementations and their customers get the assurance that the different systems will be able to communicate. We had originally thought that issues of defect ownership might arise frequently, but all the defects we encountered so far have had ownership resolved.

The distribution of errors we have seen so far is shown in Table I. Some quick observations from this table include:

- Interoperability (IOP) testing is necessary because defects were uncovered in the tests with all vendors.
- Defect rates from the session layer on down were low because the various implementations of these layers are quite stable.
- Defect rates should decrease steadily as interoperability testing is done with more vendors.

Table I
IOP Defects Exposed By Layer During IOP Testing with Eight Vendors (HP defect count at left)

Layer	A*	B*	C	D*	E	F*	G	H	Totals
FTAM	0 0	0 0	3 3	0 0	1 7	0 0	0 4	0 2	20
MMS	1 5	2 1	0 0	2 1	0 0	0 2	0 0	0 0	14
ACSE	4 1	2 4	0 1	0 0	0 1	0 1	0 0	0 0	14
Presentation	1 0	1 5	0 0	0 2	0 0	0 1	1 2	0 1	14
Session	0 0	0 0	0 0	0 1	0 0	1 0	0 0	0 0	2
TP4	0 0	0 0	0 0	0 0	0 1	0 0	0 0	0 0	1
CLNP	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0
MAC	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0

* Indicates MMS testing (otherwise FTAM)

The largest class of errors encountered was protocol data unit (PDU) encoding errors, about 20%. It is noteworthy that from the presentation layer on up, the encoding rules for the PDUs change, and this is where the majority of the errors were located. The change is that the encoding rules are no longer explicitly stated in each protocol specification as they are in the lower layer standards. Instead, all the upper layer standards use what is called Abstract Syntax Notation One (ASN.1) to describe the way PDUs are to be constructed. ASN.1 is defined by two standard documents of its own. ASN.1 allows a consistent mechanism for describing the upper layer protocols, but the cost is that the final encoding of the PDU is less clear. ASN.1 also permits lengths of PDUs to be encoded in two different ways, and several vendors, including HP, had some errors in decoding both methods. See the article on page 11 for more about ASN.1.

Within the FTAM errors, half of the errors were the result of the incorrect effect being applied to the file being accessed. For instance, a request to recreate a file with new attributes might actually result in the same file being overwritten and its old attributes kept (i.e., time of creation, owner, access restrictions, etc.). Another large class of errors came from two implementations being overly restrictive on a set of concurrency control flags, which are used to control simultaneous access to files.

Another trouble spot, at the application and presentation layers, had to do with the negotiation of contexts to be used for the two entities to communicate. When a connection is established at the presentation layer the two entities agree what application layer protocols will be used (e.g., ACSE and MMS) and what encoding rules will be used, which is always ASN.1. One problem was that the codes used to indicate the protocols being proposed were still in a state of flux late in the implementation cycle. Other problems resulted from the standard's allowing multiple ways to order the proposed protocols and different methods of encapsulating the data from the upper layers.

One last trouble area worth mentioning resulted from optional fields. Six errors were the result of either an implementation requiring a field that was optional, or not handling a field that could optionally be present.

Useful Practices

The HP 4974A MAP 3.0 protocol analyzer proved to be an indispensable tool during interoperability testing. The analyzer captured all the OSI traffic between the two systems being tested and displayed it in real time on the screen. The traffic was presented in several windows corresponding to the different layers of the stack. This multilayer display proved especially useful for situations when one system encountered an error at an intermediate layer in the stack. We could see in real time the dialog that had occurred and determine which side was the last to transmit. Because the analyzer is able to save the traffic it monitors in files, we were able to keep track of exactly what dialogs took place for each test case.

The analyzer was also useful in identifying problems that are not apparent at the application layer. An example was incorrect disconnect or abort dialogs. Although at the application layer both sides may see the connection successfully released, the analyzer exposed errors in the dialog carried out at lower layers.

Another valuable quality of the analyzer is that it acts as an unbiased observer on the network. There is no doubt that traffic displayed by the analyzer is traffic that is actually on the network. This is more reliable than traces taken by either host.

The best method for isolating encoding errors is to place traces of each system's encoding for a particular PDU side by side and examine those fields that are different. Note that this process can also be carried out before actual testing if PDU traces are exchanged by the vendors as suggested in the interoperability process.

The logging facilities of the HP MAP services and the HP OSI Express card also proved very useful. Because much care had been taken to associate unique log messages with each potential error detected throughout the stack, several errors could be identified immediately from the log message text.

Conclusion

The interoperability testing process we have developed at HP has helped us uncover and correct errors in our OSI implementation as well as several other vendors' implementations. We plan to continue our interoperability testing with more vendors in the future. We now have a better understanding of the types of problems that occur between different implementations and through our records are in a better position to support our OSI offerings in the field.

Acknowledgments

The following people played key roles in the success of our interoperability test effort: interoperability project manager Jim Cunningham, previous manager Greg Gilliom who began this project, OSI Express engineers Mike Ellis, Steve Mueller, and Tom Smith, FTAM engineer John Smith, MMS engineers Pete Lagoni, Hugh Mahon, and Jeff Williams, and fellow interoperability engineer Ken Brown. Thanks must also be extended to the engineers at Allen-Bradley, Computrol, Data General Corporation, GE-Fanuc, Motorola, RETIX Incorporated, and Unisys for their hard work in making open systems a reality.

The HP MAP 3.0 Software Integration Lifecycle

The HP MAP 3.0 program was a large multidivisional effort with project teams spread over different geographic locations and working under different organizations. To manage the integration of the hardware and software components from these different project teams, a generic integration lifecycle was developed for the HP MAP 3.0 product.

by Douglas R. Gregory

THE HP MAP 3.0 PROGRAM was a large software and hardware development project that involved teams from three HP U.S. divisions: Information Networks Division in Cupertino, California, Colorado Networks Division in Fort Collins, Colorado, and Roseville Networks Division in Roseville, California. At the start, the program was faced with the prospect of integrating and releasing as a product software, firmware, and hardware from more than ten project teams. These teams were geographically dispersed and each had their own software development and management methods. Ensuring that the final product met its functional and quality goals was akin to assembling an airplane in which all the parts were designed and built in different locations and at different times, with the parts assembled into larger subassemblies at different locations. As an added complication, the final assembly had to continue despite ongoing changes to the airplane parts.

From the experiences gained during the HP MAP 3.0 program, we have constructed a generic integration lifecycle that describes the various integration tasks and issues a large, multidivision project needs to address and when it needs to address them.

Integration Organization

In large projects integration activities can be accomplished using three different organizational models: distributed, centralized, or a combination of the two (see Fig. 1). Using the distributed integration model entails dividing integration responsibilities among one or more members from each development team. These people are responsible for integrating the team's software component into the overall product, with limited support provided by a dedicated program-wide integration person. While this appears to optimize the needs of each team, in reality the program suffers because there is often a lack of shared plans, accountability, and decision making ability. Problems are compounded in a multidivision environment because different entities have different software lifecycles, terminology, and quality and development objectives. For example, one team may consider their job complete if their software component builds and runs on just the final, customer shippable product, while another may feel it is necessary to ensure development stability by building and testing their portion of the product weekly.

A centralized integration team offers the benefits of more cohesive, program-wide integration planning, more accountability for the success of product integration, and more focused effort to solve integration problems. However, in a multidivision environment it is difficult for a centralized team to meet the different needs of each organization and project team. Care must be taken to ensure that the central integration team is perceived by the rest of the program as a partner and not as an organization constantly in the way (i.e., something to work around).

The HP MAP 3.0 program evolved into a hybrid of the above two organizations. A centralized integration team was set up with the responsibility for ensuring the step-by-step integration of the various software and hardware components for the final product and the final integration and shipment of that product via the HP-UX release process. As time progressed, it became apparent that, in addition to the central integration team, each geographically separate division needed at least one full-time person dedicated to meeting that entity's specific integration needs, plus a member from each project team to serve as the first point of contact for integration planning, problems, and issues (see Fig. 1c).

The HP MAP 3.0 Integration Lifecycle

Integration activities can be divided into three phases: the initial development phase, the initial product integration phase, and the final product integration phase. The initial development phase is characterized by preparatory activities for integration and results in an integration plan. In the initial product integration phase the integration plan from the earlier phase is executed. Finally, the final product integration phase involves making the product run on the customer-release platform. Because these phases occur in parallel with other software lifecycle activities, large, multidivisional programs may find it useful to add an integration activity column to their traditional development lifecycle.

Initial Development Phase

Occurring in parallel with the design, coding, and unit test phases of the development lifecycle, the integration objectives of this phase are to plan the strategy and tactics for assembling the individual software components to create the final product. The output of this phase, an integration plan, should address the following:

- Identifying common interfaces between teams
- Developing a program build strategy
- Implementing and distributing an integration development environment
- Identifying initial module entry criteria
- Developing an integration plan.

Because the success of the specification and design phase of a software product has a strong impact on the success of the implementation and testing phases, the activities performed in this phase of integration greatly influence the integration activities in later phases.

Common Interfaces. Because different organizations have different ways of doing things, it is critical to identify common interfaces through which they can interact. Although there were interfaces in the HP MAP 3.0 program at all levels (product teams, development teams, etc.), the interfaces that were eventually used to manage the integration effort for the program included a program-wide lifecycle and program evaluation and review technique (PERT), a common build technique and use rules, an integration development environment, initial module entry criteria, and a final product system test strategy.

These interfaces defined the way entities such as the divisions and development teams interacted with each other while leaving them the freedom to innovate to meet their specific, local objectives. It is important to note that these interfaces were introduced as needs were identified, and were continuously refined and improved as time passed.

Program Build Process and Strategy. The design and implementation of an efficient, consistent method of building a software product is critical in any program. In a project with geographically dispersed entities, it becomes even more important because of the limited communication paths available to resolve differences and defects found during product integration.

Originally intended to be used only for integration purposes, the HP MAP 3.0 build process was later adopted by virtually all project teams and became an integral part of the development environment. The build process provided a method of compiling, linking, and verifying all or part of the MAP product in a consistent, uniform fashion, while giving each project team as much independence as possible in implementing and testing their portion of the build. Fig. 2 shows the partial output of a build for HP MAP 3.0.

Of equal importance to a program's build process is its build strategy. The types of questions that need to be answered with any build strategy include:

- How often should a product build be done?
- Should a product build be done in one location or should each entity build its own portion?
- How should the build be distributed to development and test teams?
- How should new modules be integrated into the product?

The original HP MAP 3.0 build strategy was to build and distribute the latest version of the software every four to

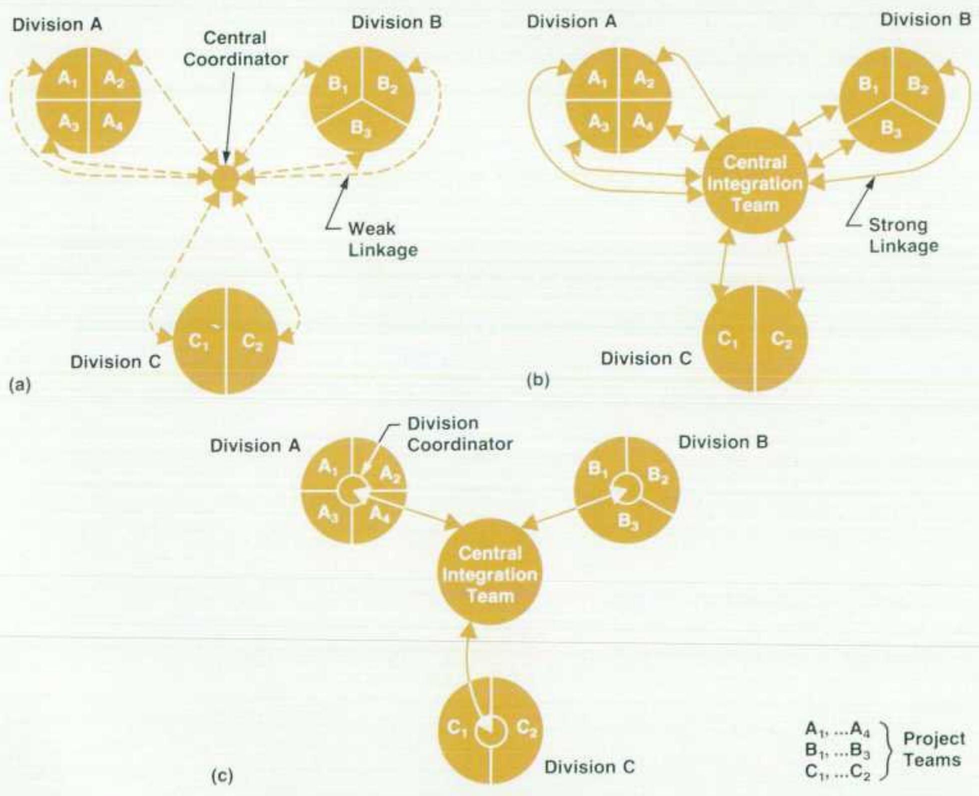


Fig. 1. *Different methods for organizing an integration team. (a) A distributed integration team. (b) A centralized team. (c) A hybrid of the two. For geographical and convenience reasons, HP MAP 3.0 adopted (c) as its model.*

six weeks. Each build, or integration release, contained successively more product functionality. The large time between releases was judged to be necessary because of the instability of the product during the early phases of development. Development teams were responsible for picking up the latest release via disk image or across the internet and using it for continued development, integration, and testing.

However, from the moment developers created their own private version of the build tree, they began to modify it, incorporating "under the table" fixes from other development teams. This meant that by the time it was necessary to build a new release, each development team's private tree was often very customized and very different. While individual modules passed their tests within private trees, six weeks later, when a completely new product build was performed, owners found that either their module or a module that depended on their module had broken because of a code change they had not incorporated in the six week interim. This resulted in integration releases that were taking longer than anticipated to create, test, and distribute.

To alleviate these problems, partway through the program, we adopted a daily build strategy. Each night, an attempt was made to build and test to a predefined level the entire HP MAP 3.0 product using the latest version of the source and test code. If a good build was achieved with a quality level appropriate to a daily build, it was saved and made available to development and test teams. By keeping the daily and other last good builds in separate areas, a failure to achieve a product build on any given day did not impact a team's ability to retrieve the latest good build. Before a major external release (e.g., an HP-UX integration cycle), a designated daily build was set aside in a separate area and tested for functionality and quality levels appropriate to its intended customer (e.g., HP MAP 3.0 system test, beta test sites, and HP MAP 3.0 system integration). Some of the other advantages gained from daily builds included:

- Daily builds provided constant feedback on the state of the product. If the daily build was broken, the HP MAP 3.0 product was broken and not usable.
- Problems could be identified and isolated while they were still fresh in the minds of those who had made the changes.
- Good builds, complete with badly needed defect fixes, could be made available on a much more frequent basis to teams who needed them.

Development Environment. One of the major problems in a large, geographically dispersed program is determining how to distribute product builds to development and test teams in a timely and nondisruptive fashion. To accomplish this task, the HP MAP 3.0 integration team adapted and modified for their use a set of tools that became known as the integrated personal development environment, or IPDE (see the box on page 59). The objectives of the IPDE were to provide:

- A private copy of a build for each developer and tester
- A mechanism in which developers and testers could easily incorporate a new build into their private copy with minimal disturbance to their development and test activities
- A procedure that allowed different teams to pick up build trees with different frequencies
- A flexible toolset modifiable to meet the different needs of different phases of the program.

Using the IPDE, the integration team was able to offer appropriately tested build trees to the different integration user groups (see Fig. 3). For instance, system test teams who were interested in high-quality, stable code, picked up a new build tree only after a major external release. Some developers, on the other hand, wanted to be running the latest code to test the compatibility of their changes, or because they needed a major defect fix.

After a group picked up a new build tree, individual users in that group linked into the new build tree. In that way they were able to begin using in a short period of time a build tree that contained the latest checked-in files from all other teams, while continuing to use private versions of their work files. By simply recompiling the portion of the build they were working on, developers and/or testers could decrease the probability that when they made their work files available for a later build (by checking them into the source control system), they would break another module or another module would break theirs.

Returning to the aircraft analogy used earlier, imagine the chaos that would reign on the assembly line (or the flight line) if a wing designer were to make a change to the wing shape without being able to assess the change's impact on the whole aircraft. Unless this assessment is done, the whole assembly line may have to shut down, or, worse, the aircraft may not even leave the ground. The IPDE provided a way for software developers to make such checks on the HP MAP 3.0 product.

Modules Entry Criteria. As developers finish their initial code and module tests, it is important for them to know the release criteria for their software modules. In other

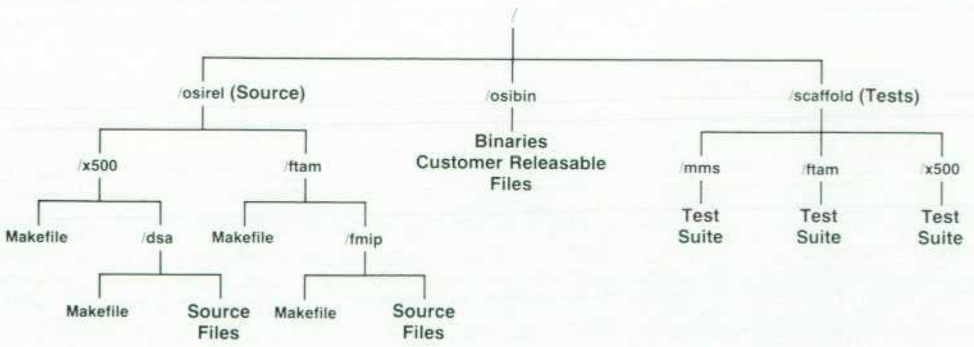


Fig. 2. *Part of an HP MAP 3.0 build tree. Note the separate directories for source, binaries, and executables. The source and test directories were further divided so each project team could completely own their own subdirectory.*

words, at what point will their modules be accepted as an integratable piece into the final product? In a document known as the integration entry criteria, the HP MAP 3.0 integration team explicitly spelled out these criteria. Some sample items included:

- Each module was required to conform to build process rules.
- Specific test suites were required to accompany each submitted module (i.e., a quick and full regression suite).
- All test suites were required to run in a special test execution environment (specifically, the scaffold).[1]

These module criteria helped to raise the likelihood of successful later integrations. Those teams that began using the criteria in this document from the start of their coding phase spent far less time converting their modules and development environments to fit the later needs of integration.

Module Integration Plan. One of the most important outputs of the initial development phase of the lifecycle is the development of a module integration plan. HP MAP 3.0's multidivisional nature made this even more difficult because in many instances modules that resided in the same HP-UX executable program were developed by different teams in geographically separate locations. Significant effort went into producing a plan that reduced the amount of engineering effort and calendar time it took to produce a single product from the many HP MAP 3.0 components.

The plan was constructed using data flow techniques to gain as much parallelism in the integration process as possible (see Fig. 4). Each bubble in the plan represented an integration step that could be accomplished without reliance on modules not previously integrated. For instance, the initial FTAM application program interface integration was accomplished using stubs for all modules except for a portion of the upper layer architecture.

To complete the integration plan it was necessary to identify, for each bubble:

- Project teams responsible for completion (consider geographic difficulties)
- Functionality at entry and exit
- Testing level at entry and exit
- Needed test suites
- Needed test equipment
- Timetable (schedule, manpower, etc.)
- Module dependencies
- Needed stubs or emulators.

The completed plan had the benefit of making the integration steps simple, manageable and measurable.

The HP MAP 3.0 program also made attempts to preserve the software investment in test suites between integrations. For instance, the test suite used to test the initial FTAM and MMS integrations was also used to test all subsequent integrations that involved the FTAM and MMS modules. No new test suites had to be written. Instead, the same suites were run on the newly integrated code with stubs and software emulators replaced with their real counterparts.

Initial Integration Phase

If the initial development phase is characterized by planning, then the initial integration phase is the execution of that plan. The objective of this phase is to expend as little calendar time and engineering effort as possible to assemble all pieces of the software product on a stable operating system (i.e., assemble the product while holding stable all pieces that are external to it). The primary integration responsibilities during this phase are to:

- Regularly perform daily and release builds and provide feedback on status to management, developers, and testers
- Continuously improve the build strategy and process to improve its reliability
- Continuously monitor progress of the integration plan and modify it as needed to account for changes.
- Continuously support distributed toolsets.

A missing element in this list is the execution of each integration bubble on the integration plan. Often in large programs, the product is so complex that no single team can hope to have the knowledge needed to assemble all the pieces of the puzzle in a reasonable amount of time.

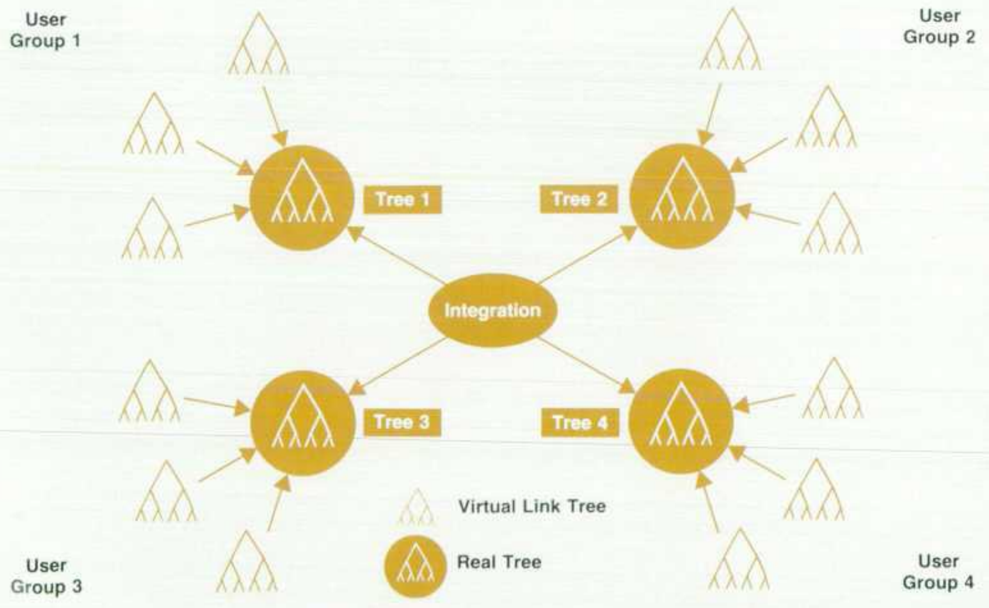


Fig. 3. *HP MAP 3.0 build user groups. The HP MAP 3.0 development environment provided for the easy distribution of builds to teams with different needs. Individual users were virtually linked to a real copy of a build tree.*

Even on programs where the modules have clearly defined interfaces, the first time a module is required to function with another module, defects are found requiring knowledge and expertise from all the module builders to debug and fix the defect. Such was the case in the HP MAP 3.0 program.

As development teams finished their modules, they were given the responsibility for executing one of the integration bubbles in the integration plan. Attempts were made to assign these responsibilities to one geographic location to reduce communication needs between entities. However, many times there was still the need to assemble small, short-lived, cross-project, and cross-divisional teams to work on major integration problems. These teams proved to be very effective because they brought together knowledge from one or more modules, test suites, or previous integrations.

As part of the daily build process, integrators need to provide continuous, appropriate feedback to the various members of the program team. The build process can be improved by providing a daily feedback mechanism that includes such things as the state of the current build, problems encountered, and historical metrics (e.g., modules responsible for breaking a build, number of successful builds, code stability, etc.), and by having a strong commitment from each team to the build's success. The regular feedback also gives management a good understanding of the state of the product's quality level and stability. The key to using the feedback effectively is to ensure that it is nonpunishing and that it is used constructively to determine what steps to take to improve the build process.

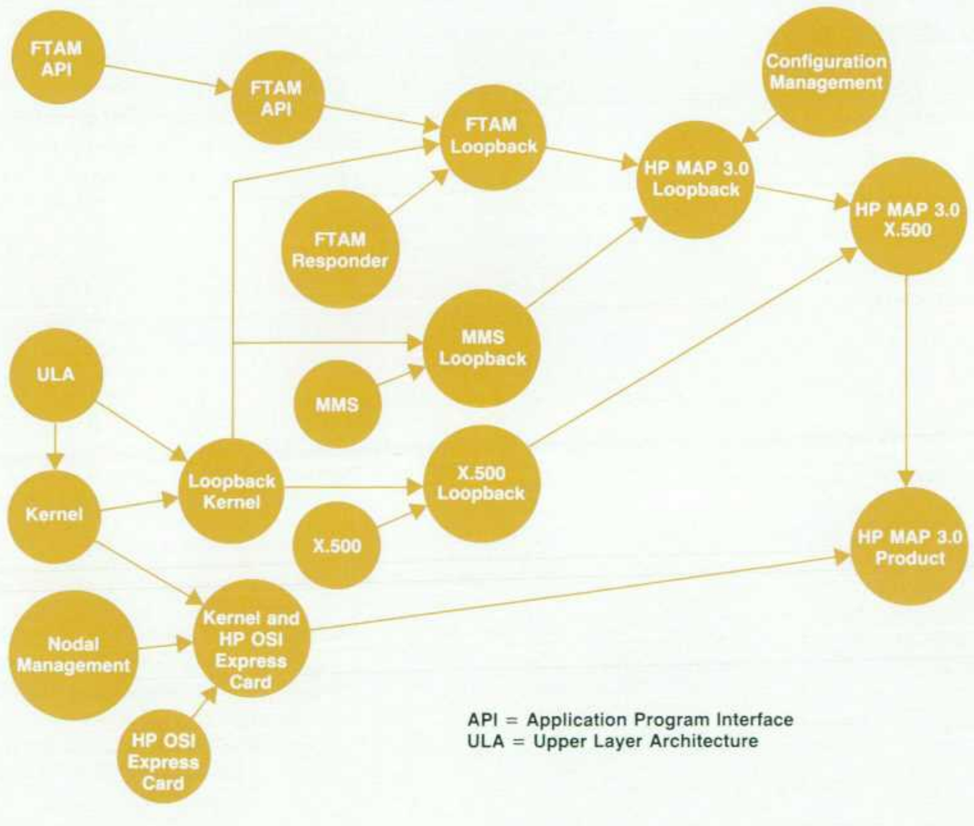
Also critical during this phase is the effective management of the integration plan (see Fig. 4). As with any data flow process, if any early integration bubble completes late or without the required exit criteria (e.g., functionality, test suites, quality level, etc.), it can impact an integration bubble later in the plan. Minimizing these impacts requires good cross-team communications and coordination. The HP MAP 3.0 team used a variety of forums to deal with integration difficulties including the use of emergency action teams to focus on defects, cross-divisional program management teams, and weekly or biweekly integration teleconferences. Some of the solutions included putting more effort on integration bubbles that were falling behind, changing the entry and/or exit requirements of certain bubbles, and revising the order of the integration plan itself.

Final Product Integration Phase

While the product integration phase was primarily concerned with assembling an unstable product on a relatively stable platform, final product integration involves taking a reasonably integrated and stable product and making it work on a new, and often unstable, base operating system. This phase of the integration lifecycle is necessary primarily when the integrated product must be shipped with a new operating system or other dependent system. The primary integration responsibilities during this phase include:

- Representing the technical needs and responsibilities of the program to the operating system release team
- Modifying the build process and tools to build the final product on the new operating system
- Incorporating any operating system integration tools into the developers' tool kit.

Since it spans the entire program and usually has the



API = Application Program Interface
ULA = Upper Layer Architecture

Fig. 4. *Part of the HP MAP 3.0 integration plan. Each bubble represents a separate integration activity complete with entry and exit criteria.*

The Integrated Personal Development Environment

The integrated personal development environment (IPDE) is a set of tools designed to give each software developer the ability to acquire and modify a copy of the most recent product build tree.

Originated for the HP 9000 Series 500 HP-UX kernel project, the ideas for the environment were later incorporated into a toolset for use by the Network File System (NFS) teams before being adopted and enhanced for the HP MAP 3.0 program. The IPDE is designed to work with the following software development paradigm:

- The developer works on a copy of the product build and checks a file out of the source control system and modifies it.
- The developer recompiles all affected pieces of code and thoroughly tests the change.
- The modified file is checked back into the source control system for incorporation into the next product build.

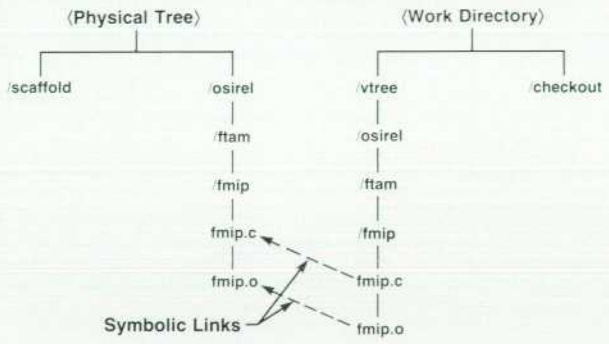


Fig. 1. *An IPDE work directory after a user has linked into a new build tree. Note that all the files in* vtree*, the user's virtual copy of the tree, are symbolic links to files in the physical tree.*

Each IPDE user is given a work directory containing a configuration subdirectory, a virtual tree directory, and a checkout directory. After a build tree is available, a virtual copy of the build tree is created under the virtual tree directory. The virtual tree looks identical to the real tree, except that all files in the virtual tree are actually symbolic links to the real files in the physical tree as shown in Fig. 1.

When a user checks out a file from the source control system (Fig. 2), a new, parallel directory is created in the user's checkout directory, the real checked-out file is placed in the checkout directory, and the the symbolic link file in the user's virtual tree is changed to point to the new checked-out file. The user then modifies the new file as desired.

Because of special modifications to the build process, when the user recompiles the checked-out module, the symbolic link to the old binary is dissolved and the new binary appears in the user's virtual tree (see Fig. 3).

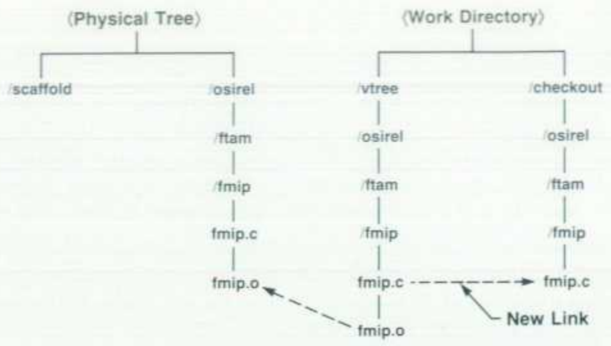


Fig. 2. *The IPDE work directory after a user has checked out the file* fmip.c. *A new copy of the file has been created in the checkout directory and the symbolic link in the user's virtual tree changed to point to it.*

After a new build tree becomes available, the user again is linked into the new tree. Assuming the user has not yet checked in a work file, the new virtual tree would once again look like Fig. 1. While keeping existing work files, the user has picked up the latest copy of the product build, incorporating the latest updates, fixes, and enhancements from other teams. The user need only recompile the virtual tree after each relink to continue working. This recompilation is highly desired from an integration point of view since it helps ensure that the user's work files are still compatible with the latest version of the product build and helps avoid later build problems.

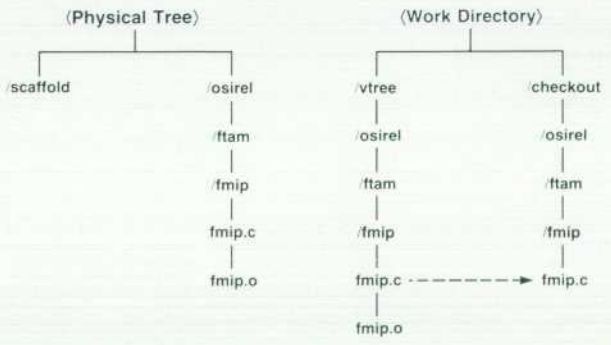


Fig. 3. *The IPDE work directory after the user has rebuilt the modified file* fmip.c. *The build process was written to remove the link to the target object module,* fmip.o *before compiling, leaving a private copy of the new file in the user's work directory.*

best technical overview of the entire product, the integration team is ideally suited to serve as the program's technical representative to the operating system release team. For the HP MAP 3.0 program this entailed monitoring, communicating, and assessing the impact of the HP-UX 7.0 schedule and deliverable changes on the program. This also involved ensuring that all HP MAP 3.0 software components met HP-UX 7.0 system release requirements, and building and releasing versions of the HP MAP 3.0 product to the HP-UX 7.0 integration teams.

The amount of integration effort required during this phase should not be underestimated, especially if the release platform is in an unstable state when the decision is made to begin using it. Because typical development, testing, and integration activities have often occurred on a customer released, stable operating system, the issues and concerns that arise during the early phases of the integration lifecycle are primarily internal. Once a new, potentially unstable operating system is adopted, issues must be resolved that are often external to the new product. Having a representative or liason to the HP-UX system release team proved invaluable in resolving issues and avoiding problems.

Although significant effort was spent getting the HP MAP 3.0 product tested to HP-UX 7.0 quality levels, the initial transition from a customer shipped, stable version of the HP-UX platform to the new HP-UX 7.0 integration environment did not take a large amount of time. To ease this transition, the HP-UX 7.0 program developed a transition toolkit called the *build environment* that functioned like a cross compiler. It allowed developers to build their products on a stable HP-UX platform, but to run the resultant binaries on a potentially unstable target system. The integration team was able to modify the build process and the IPDE toolkit easily to incorporate the build environment, making the changes almost transparent to developers and testers.

Conclusion

Several key factors led to the success of the multidivisional HP MAP 3.0 integration efforts and can be applied to other such efforts. These include:

- The integration team should view itself as a service organization whose internal customers are developers and testers.
- The entire program should view the product integration phase as a critical part of the product's development. They should design their software product and processes for integratability, just as hardware engineers must design their products and processes for manufacturability.
- A common set of interfaces between all teams must be established.
- There should exist thorough, complete, and reviewed integration plans.
- There should be regular builds, at least weekly.
- There should be a program-wide build process and a set of integration and development tools to support it.

The integration lifecycle described in this article is just another component of the entire product lifecycle, paralleling the development phases of design, coding, unit testing, and the more traditional integration periods. Project teams wishing to incorporate this integration component into their lifecycle should modify it to meet the needs of their own program. By performing the early planning and continuous coordination emphasized in this lifecycle, large, multidivisional projects can efficiently use their resources and reduce the development time and cost of their product.

Acknowledgments

The author would like to acknowledge the contribution of all the HP MAP 3.0 integrators: Tom Robins, Toni Atkinson, Diane Bracken, Ishun Chang, Stephen Fung, Mike Robinson, Jan Stevenson, and Brad Taylor. Special thanks to Karl Jensen for the original idea for the integrated personal development environment tool, and to John Dilley and Christina Mahon for their enhancements to the tool.

References

1. C.D. Fuget and B.J. Scott, "Tools for Automating Software Test Package Execution," *Hewlett-Packard Journal*, Vol. 37, no. 3, March 1987, pp. 24-28.

Authors

August 1990

6 — HP MAP 3.0

Bruce J. Talley

Bruce Talley is a product manager at HP's Information Systems Division and a team leader responsible for marketing and releasing HP's MAP 3.0 product. He joined HP in 1987 and has planned several successful OSI early release sites, delivered sales engineering technical training courses, and trade show and customer demonstrations. A member of the Society of Manufacturing Engineers, Bruce earned a BS degree (1984) in business administration from the University of Arizona, and an MBA degree (1990) from Golden Gate University in San Francisco. Before joining HP, he was a communication specialist with Telefile Computer Products Incorporated in Irvine, California, a communication specialist with Electronic Data Systems Corporation in Southfield, Michigan, and a coop student with IBM in Tucson, Arizona. Born in Blythe, California, Bruce is married and resides in Santa Clara, California.

Collin Young Woon Park

Collin Park joined HP's Data Systems Division in 1976. He worked on the development of the kernel modules for the HP MAP 3.0 product. Born in Honolulu, Hawaii, he earned his BS degree (1976) in mathematics, and his MS degree (1980) in electrical engineering from Stanford University. He attended graduate school assisted by HP's Honors Coop Program. Collin was the U.S. representative to the ISO presentation working group in 1987, and one of several HP representatives on the ANSI X3T5.5 (OSI) task group from 1986-to-1987. His past assignments included work on the hardware and software development of datacom for HP 300 and HP 3000 computers, and he was a project manager for development of the HP 3000 IEEE 802.3 card. A member of the IEEE Computer Society, Collin's professional interests center around software engineering productivity. He is married, has one daughter, and lives in Redwood City, California. His interests include bible study, skiing, hiking, and camping.

11 Upper Layer Architecture

Sanjay B. Chikarmane

Sanjay Chikarmane graduated from the Indian Institute of Technology in Bombay, India, with a Bachelor of Technology degree (1981) in electrical engineering, and from Syracuse University in New York with an MS degree (1982) in computer engineering. A software development engineer, he joined HP's Personal Computer Group in 1985. Sanjay developed the upper layer architecture and implemented the OSI presentation layer for the HP MAP 3.0 product. Before joining HP, he designed and developed a real-time operating system and X.25 networking protocols at Rolm Corporation. Born in Bangalore, India, Sanjay is married and lives in San Jose, California. His hobbies include skiing, cricket, and photography.

15 Directory Services

Beth E. Cooke

Beth Cooke tested and supported the directory services product for the HP MAP 3.0 project after joining HP's Information Networks Division in 1988. Databases are one of her major professional interests. Before joining HP, she worked as a summer intern at IBM and Magnavox Electronics. Beth received her BS degree (1988) in computer science form Purdue University in Indiana. Born in Fort Wayne, Indiana, Beth and her husband have recently moved to Milwaukee, Wisconsin. An avid Purdue sports fan, Beth's hobbies include windsurfing, snow skiing, and water skiing.

Colleen S. Fettig

Managing and playing on a competitive soccer team are two of Colleen Fettig's after-hours activities, and she brings the same team spirit to HP's Information Networks Division. As a development engineer, Colleen designed and implemented the database, database access module, and the cache mechanism for the HP MAP 3.0 X.500 directory services. She also designed and implemented database management tools for the X.500. Prior to joining the X.500 development team, she implemented the intrinsic interface and developed the COBOL definition extractor for the HP System Dictionary, which is used with HP 3000 computers. She joined HP's Computer Support Division in 1982, the same year she received her BS degree in computer science from Oregon State University. Born in Seattle, Washington, Colleen is married and lives in San Jose, California. The soccer team she manages and plays with is part of the competitive first division of the Bay Area Women's Soccer League. Her interests also include bible study, backpacking, fishing, mountain biking, and outdoor activities.

Darrell O. Swope

Networking and distributed applications are the professional interests of software engineer Darrell Swope. A graduate of the Georgia Institute of Technology with a BS degree (1987) in computer science, he joined HP's Information Networks Division that same year. Darrell was responsible for the development of the X.500 protocol for the HP MAP 3.0 product. Born in Marietta, Georgia, Darrell is married and lives in Sunnyvale, California. He enjoys skiing, mountain biking, and basketball.

Paul B. Koski

Soon after joining HP's Information Networks Division as a member of the technical staff in 1988, Paul Koski worked on X.500 protocol development for the HP MAP 3.0 product. He earned an AA degree (1979) from Pierce Community College in California, a BS degree (1984) in natural resources management, and an MS degree (1987), in computer science, both degrees from the California Polytechnic University at San Luis Obispo, California. Distributed computing is Paul's major professional interest. Before joining HP, he was a consultant and network systems developer with Impell Pacific Company. Born in Lansing, Michigan, Paul is married, has two children, and lives in Cupertino, California. His interests include coaching youth sports, basketball, backpacking, fishing, running, and reading.

Roy M. Vandoorn

A scuba diver, emergency medical technician, Red Cross volunteer, and CPR (cardio-pulmunary-resuscitation) instructor, Roy Vandoorn's multiple interests also include developing a trouble-shooting program for HP service engineers. As a technical staff member of HP's Information Networks Division, Roy developed the directory user agent library (DUALIB) for the HP MAP 3.0 X.500 directory services. He's now working on new product developments. This is Roy's second article in the Journal. In 1986, he coauthored an article on a rule-based system to diagnose malfunctioning computer peripherals. He received an AA degree (1976) from DeAnza College in Cupertino, California, a BA degree (1978) in mathematics with a concentration in computer math, and an MS degree (1980) in mathematics, both from San Jose State University in California. Before joining HP's Computer Support Division in 1980, Roy worked as a systems analyst in a college coop program for IBM, where he supported and documented an interface between an inventory control system and a purchasing program. Born in Edmonton, Alberta, Canada, Roy currently resides in San Jose, California. Besides his life-saving avocations, Roy also enjoys diving, skiing, fishing, photography, and travel.

24 FTAM/800

Steven W. Manweiler

Born in Phoenix, Arizona, Steven Manweiler received his BS degree (1984) and MS degree (1986) in computer science from the University of Arizona. He joined HP's Colorado Networks Division in 1987, and developed and tested the HP Map 3.0 File Transfer, Access, and Management (FTAM) system. Prior to joining the HP MAP 3.0 program, Steve worked on testing HP's ARPA/Berkeley networking services. He is HP's representative to the National Institute of Standards Technology special interest group on FTAM. Steve's professional interests include distributed systems, programming languages, and compilers. He is a coauthor of a previous technical journal article on distributed systems and programming languages. He lives in Fort Collins, Colorado, and enjoys golf, cycling, running, hunting, and skiing.

31 MMS/800

Peter A. Lagoni

Peter Lagoni works as an R&D engineer with HP's Colorado Networks Division. He was responsible for the design and development of the MMS protocol machine module for HP MAP 3.0, and is the lab support engineer for the HP MAP 3.0 MMS product. He also assisted in performance and interoperability testing for HP MAP 3.0. Pete earned a BS degree (1977) in agronomy from Iowa State University, and a BS degree (1986) in computer science from Colorado State University. He joined HP as a summer intern in 1986 while studying at Colorado State University, and worked on Colorado Networks Division's HP-UX toolbox. He came to HP full-time in 1987 as a software development engineer. He is a member of the MAP 3.0 MMS Technical Committee and the NIST OSI Implementors Workshop MMS special interest group. Born in Davenport, Iowa, Pete is married, has one child, and lives in Fort Collins, Colorado. His interests include home improvement projects, family activities, and environmental concerns.

Christopher Crall

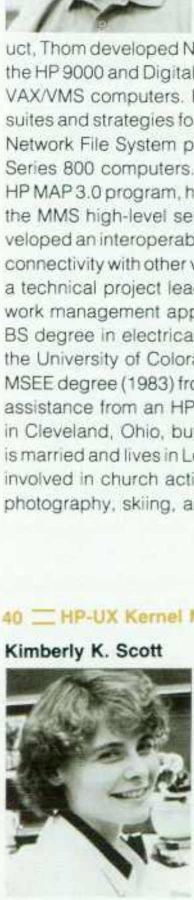
Software development engineer Christopher Crall was responsible for the design and implementation of the MMS interface for HP MAP 3.0. He joined HP's Colorado Networks Division in 1985 after earning a BS degree (1981) and an MS degree (1984) in computer science from Iowa State University. He also worked on ARPA/Berkeley networking services for the HP 9000 Series 800, and tested network services between the HP 9000, HP 3000, and HP 1000 computer systems. Before joining HP, Chris was a programmer with John Deere Company, and an instructor at Iowa State University. He is a member of the MAP 3.0 Application Interface Subcommittee, which developed the MMSI standard. Born in Albia, Iowa, Chris is married, has one child, and lives in Fort Collins, Colorado. He enjoys spending time with his family, basketball, and bicycling.

Thomas G. Bartz

Thomas Bartz joined HP in 1981 as a hardware development engineer assigned to digital hardware design for the HP 9000 Series 500 workstation. Since 1983, when he joined HP's Colorado Networks Division, and prior to working on HP's MAP 3.0 product, Thom developed NS Network File Transfer for the HP 9000 and Digital Equipment Corporation's VAX/VMS computers. He also developed test suites and strategies for HP's ARPA/Berkeley and Network File System products for the HP 9000 Series 800 computers. As a team member in the HP MAP 3.0 program, he designed and developed the MMS high-level service provider, and developed an interoperability test plan to verify MMS connectivity with other vendors. Thom is currently a technical project leader for HP OpenView network management applications. He received his BS degree in electrical engineering in 1981 from the University of Colorado. He also earned an MSEE degree (1983) from Stanford University, with assistance from an HP Resident Fellowship. Born in Cleveland, Ohio, but raised in Colorado, Thom is married and lives in Loveland, Colorado. Actively involved in church activities, his hobbies include photography, skiing, and tennis.

40 — HP-UX Kernel Modules

Kimberly K. Scott



As a software development engineer in HP's Information Networks Division, Kimberly Scott developed the tracing and logging functions for the HP MAP 3.0 kernel modules. She joined HP's Computer Systems Division in 1981. Since then, Kim has developed SNA communications products for the HP 3000 computer. Currently, she is responsible for maintaining the OSI Express card driver. Kim earned a BA degree (1981) in computer science from Indiana University, and an MS degree (1984) in computer science from Stanford University. Born in Webster City, Iowa, Kim resides in Santa Clara, California. Her hobbies include ballet and jogging.

Eric C. Scoredos

Eric (Rio) Scoredos is a software development engineer with HP's Information Networks Division. He helped develop the driver and diagnostics for HP MAP 3.0. Previously, he developed operating system interfaces for the GMT 400 and pathway portions of the MAP 2.1 product. Rio joined HP's Advanced Manufacturing Systems Operation in 1985. Before joining HP, he was a software development engineer at Rolm Corporation working on telecommunications applications, at Sielox Corporation working on user interfaces and graphics, and at Data General Corporation developing a new computer architecture. He received a BA degree in history from the University of California at Berkeley, and studied computer science at Stanford, the University of North Carolina at Chapel Hill, and at the University of California at Santa Cruz. He is a candidate for a PhD degree in literature, pending submission of his dissertation, at the University of California at Santa Cruz. Born in Washington, D.C., Rio lives in Boulder Creek, California. His hobbies include canoeing, skiing, bicycling, and diving.

Richard Henry Van Gaasbeck

A software development engineer in HP's Information Networks Division, Richard Van Gaasbeck developed the CONE interface adapter and OSI Express card driver for HP MAP 3.0. In his previous assignments, he developed the telnet pseudo driver for the HP 1000 computer, and enhanced and maintained the real-time executive drivers for the HP 1000. Rich, whose professional interests include I/O drivers, microcomputers, and graphics, joined HP's Data Systems Division in 1985. He received his BS degree (1985) in computer science from the California Polytechnic University at San Luis Obispo, California. Born in Baltimore, Maryland, Rich is married and lives in Mountain View, California. His interests include softball, home improvement projects, music, astronomy, and reading.

50 — Interoperability Testing

Jeffrey D. Meyer

Jeffrey Meyer joined HP's Information Networks Division in 1986, after he graduated from the University of Wisconsin at Madison, where he earned a BS degree (1984) in computer science and math, and an MS degree (1985) in computer science. Jeff was responsible for the design and development of the diagnostic and verification tools and the interoperability testing strategy for HP MAP 3.0. Prior to that, he worked on a team responsible for implementing MAP 2.1 services for HP 3000 computers. Born in Pasadena, California, Jeff is married and now lives in Sunnyvale, California. His interests include biking, juggling, hacky-sack, and traveling.

54 — Software Integration Lifecycle

Douglas R. Gregory

After joining HP's Colorado Networks Division as a system tester in 1985, Douglas Gregory worked on HP's first ARPA/Berkeley 4.2 sockets for HP 9000 Series 300 computers. As a member of the technical staff, he codesigned host presentation services and developed the software integrate-and-test system for the HP MAP 3.0 program. Doug is now a productivity manager at HP's Graphics Technology Division. His major professional interests include networks, software engineering, and software project management. He received a BA degree (1983) in English, a BS degree (1983) in computer science, and an MS degree (1985) in computer science, all from Washington State University. Born in Fairbanks, Alaska, Doug lives in Ft. Collins, Colorado. interests include hiking, fishing, back-country skiing, road and mountain biking, traveling, and photography.

64 — Programmable Pulse Generators

Gerd F. Koffmane

After graduating from the University of Stuttgart with an engineering diploma in theoretical electronics and microelectronics, Gerd Koffmane joined HP's Böblingen Instruments Division in 1983 as an R&D engineer. For the HP 8131A pulse generator project, he was responsible for developing the timing system architecture and the bipolar timing IC. Previously, he designed a fiber optic receiver. Gerd serves as HP's representative in microelectronics at EuroPACE (the European Program of Advanced Continuing Education). His professional interests include bipolar IC design, microelectronics, and IC technology. Born in Heidenheim, West Germany, he is married and lives in Althengstett at the edge of the Black Forest. His hobbies include sailing, bicycling, guitar playing, and music.

Werner Berkel

Werner Berkel is the project manager of the team that developed the HP 8130A and 8131A pulse generators. After joining HP in 1978 as a project engineer at the Böblingen Instruments Division, he helped develop the HP 8180A data generator and served as project manager for the HP 8151A optical pulse power meter and the HP 8145A optical time-domain reflectometer. He earned his engineering diploma (1977) from the Engineering Academy in Karlsruhe, West Germany. His professional interests include circuit design and technology. He has authored previous articles in the HP Journal on timing systems, fiber optics, and the optical power meter. Born in Speyer, West Germany, Werner is married and lives in Merklingen, West Germany, near the Black Forest. His hobbies include bicycling, soccer, and technical literature.

Frederick L. Eatock

Fred Eatock is a hardware development engineer with the HP Systems Technology Division in Cupertino, California. He helped develop a bipolar integrated circuit for the HP 8130A and 8131A pulse generators. Fred joined HP's Optoelectronics Division in 1977. He has contributed to the development of ICs for optoelectronic, instrument, and computer systems, and is currently a member of a hardware design team developing computer processor boards. He is listed as an inventor on three patents on analog comparators, voltage regulators, and low-noise optical detection systems. Before joining HP, he developed linear integrated circuits at Fairchild Semiconductor Corporation. He is the author of a paper on instrumentation amplifiers presented at the 1973 International Solid State Circuits Conference (ISSCC). Fred graduated from the University of British Columbia, earning a BASc degree (1964) in electrical engineering. He also has an MSEE degree (1973) from the University of Santa Clara in California. Born in Regina, Canada, he lives in Cupertino, California, is married, and has a son. His interests include camping, shortwave radio, photography, astromony, and electronic construction projects.

Heino Höpke

Heino Höpke joined HP's Böblingen Instruments Division as a design engineer in 1985, shortly after receiving an engineering diploma in computer science from the Engineering School in Hamburg. He helped design the HP 8130A and 8131A pulse generators, specifically the software, the microprocessor board, and the keyboard. Born in Spieka, Niedersachsen, Heino is married, has two children, and lives in Ehningen, Baden-Würtemberg. His interests include jogging, photography, and his family.

Patrick Schmid

Patrick Schmid joined HP's Böblingen Instruments Division in 1987 and designed the timing board of the HP 8131A pulse generator. He is currently the R&D training manager for Hewlett-Packard GmbH. Patrick received his engineering diploma in 1987 from the University of Stuttgart. His hobbies include music and nature studies.

Hans-Jürgen Snackers

Hans-Jürgen Snackers joined HP's Böblingen Instruments Division as a mechanical engineer in 1986, shortly after receiving a micromechanics degree from the Engineering School in Furtwangen in the Black Forest in West Germany. He was responsible for the mechanical design, hybrid interconnections, and thick-film layout and process of the HP 8130A and 8131A pulse generators. Born in Ludwigshafen/Rhein, Hans-Jürgen is married and lives in Sulz an Neckar. His hobbies include hiking, jogging, and railway modeling.

79 500-MHz Output Section

Hans-Jürgen Wagner

Hans-Jürgen Wagner was responsible for development of the output amplifier board, the hybrid technology, and the GaAs integrated circuit for the HP 8131A pulse generator. His professional interests include microelectronics, ICs, and hybrid technology. He received his engineering diploma in 1984 in technical electronics from the University of Stuttgart. Born in Ludwigsburg, Hans-Jürgen lives in Sindelfingen. He is married and has two children. He enjoys bicycling, soccer, dramatic acting, and technical literature.

Stefan G. Klein

As a member of the HP 8131A pulse generator development team, Stefan Klein was responsible for designing the transducer board. He joined HP's Böblingen Instruments Division in 1987 as an R&D project engineer. His professional interests center around microwaves and technology. A graduate of the University of Saarbrucken, Stefan earned an engineering diploma in 1985 in high-frequency microwave and measurement technology. Born in Schaffhausen, West Germany, Stefan lives in Gechingen, is married, and has two children. His hobbies include railway modeling, trumpet playing, and jazz.

85 300-MHz Output Section

Volker Eberle

As a project leader for the HP 8130A pulse generator, Volker Eberle was responsible for the output board and for transferring the HP 8130A to production. He is currently investigating system architecture. After joining HP in 1975, Volker helped develop the HP 8092A delay module for the HP 8080A pulse generator system, a bipolar LSI chip for several instruments, and the HP 81511A optical head. He is an engineering graduate of the University of Stuttgart, earning an engineering diploma in 1974 in communication and high-frequency techniques. Born in Stuttgart, Volker resides in Böblingen. He is married, has two daughters, and enjoys playing the accordion, biking, and model railroading.

Peter Schinzel

Peter Schinzel joined HP as an R&D engineer at the Böblingen Instruments Division in 1987. He helped design the HP 8130A slope generators and hybrid technology. A graduate of the University of Stuttgart, Peter earned his diploma in 1987 in communication and theoretical electronics. Born in Stuttgart, he now lives in Böblingen. His hobbies include biking and astronomy.

Günter Steinbach

Günter Steinbach earned an engineering diploma (1977) in electrical engineering from the University of Karlsruhe in West Germany. He was a visiting scholar at the University of California at Berkeley in 1978, and then studied for his PhD degree (1985) in electrical engineering while working at Ruhr University in Bochum. Gunter joined HP as an applications engineer in 1985 at the HP Technology Center in Santa Clara, California, and developed custom bipolar ICs, including the linear output amplifier IC for the HP 8130A pulse generator. His work has resulted in a patent on an analog-digital converter. A member of the IEEE, Günter's professional interests center around bipolar integrated circuit design. Born in Pforzheim in Southern Germany, he is married, has two sons, and lives in San Jose, California.

500-MHz and 300-MHz Programmable Pulse Generators

These instruments are capable of testing the most advanced CMOS, ECL, and GaAs devices. A custom bipolar IC generates the timing parameters.

by Werner Berkel, Gerd Koffmane, Frederick L. Eatock, Patrick Schmid, Heino Höpke, and Hans-Jürgen Snackers

THE PULSE GENERATOR is a versatile and fundamental instrument that is widely used for parametric measurements on digital devices in automatic test and bench R&D applications. Using a fast pulse generator at the input of a device under test and a fast sampling oscilloscope at the output, measurements of setup and hold times, propagation delays, maximum toggle frequencies, input level sensitivities, and many other parameters can be made at normal operating speeds. The pulse generator is also well-suited for characterizing transmission lines. It can be used to measure crosstalk, loss, impedance matching, and other characteristics.

In fast bipolar digital technology and the fast-growing GaAs field we saw a demand for a fast pulse generator. After the initial investigation, we defined two. The HP 8131A (Fig. 1) is a 500-MHz pulse generator with fixed 200-picosecond rise and fall times. The HP 8130A (Fig. 2) is a 300-MHz generator with variable transition times. Most of the building blocks of the HP 8131A are used in both instruments, the main exceptions being the HP 8130A slope generator and output amplifier. Both instruments are fully programmable via the HP-IB (IEEE 488, IEC 625).

500-MHz Pulse Generator

The HP 8131A's 200-ps transition times (10% to 90%) and 500-MHz maximum repetition rate are suitable for testing the most advanced CMOS, BiCMOS, ECL, and GaAs devices. For higher repetition rates, a transducer mode allows the generator to convert an external sine wave at a frequency up to 1 GHz to a square wave with 200-ps transition times and selectable amplitude up to 5V.

The pulse period (50% level) is programmable from 2 ns to 99.9 ms. The pulse width (50% level) is programmable from 500 ps to 99.9 ms, the maximum depending on the period. Timing resolution is three digits—10 ps best case.

The output swing is programmable from 100 mV to 5V in a ±5V offset window into 50 ohms. An optional second channel provides two different but synchronized signals for simulating clock and data signals for setup and hold time measurements.

Fig. 3 shows typical HP 8131A waveforms.

300-MHz Variable-Slope Pulse Generator

The HP 8130A's rise and fall times (10% to 90%) are programmable from 1 ns to 100 μs (independently from 2 ns). The pulse period is programmable from 3.33 ns to 99.9 ms, and the pulse width (50% level) is programmable from 1.5 ns to 99.9 ms, the maximum depending on the period. Timing resolution is 10 ps best case.

The output swing is programmable from 100 mV to 5V in a ±5V offset window into 50 ohms, and an optional second channel is available.

Fig. 4 shows typical HP 8130A output waveforms.

Block Diagram

Fig. 5 is a simplified block diagram of the two pulse generators. The top level shows the base unit of both instruments excluding the power supply and the processor unit.

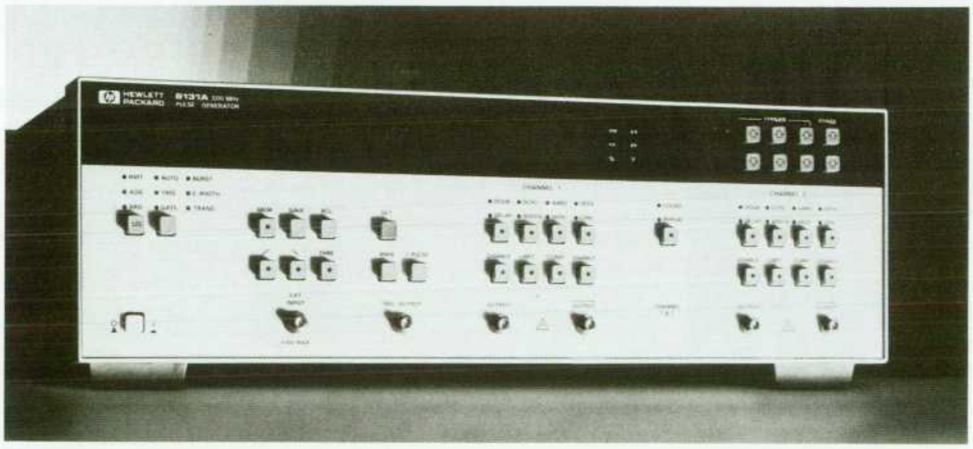


Fig. 1. *HP 8131A 500-MHz pulse generator with fixed 200-ps rise and fall times (optional two-channel version).*

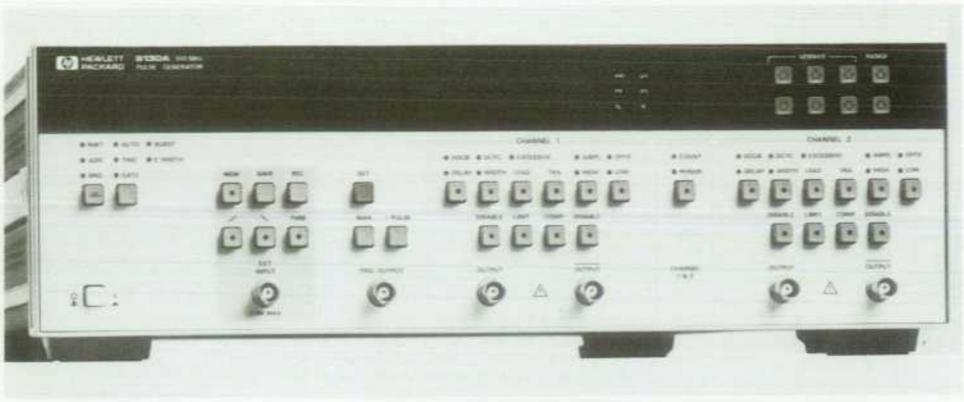


Fig. 2. *HP 8130A 300-MHz pulse generator with variable transition times (optional two-channel version).*

The trigger input amplifies and shapes external signals for the external width, external burst, external trigger, and external clock modes. After the period oscillator, which generates the basic pulse train, the signal is split into the trigger output channel and the main channels. The delay generator determines the delay between the main channels and the trigger output. The delay generator is followed by the width generator, which determines the output pulse width.

The second level of the block diagram shows the HP 8131A output system, which is described in detail in the article on page 79. A pulse shortener circuit reduces the minimum width of 1 ns generated by the width generator to 500 ps. An ECL signal modified by the normal/complement switch drives the GaAs amplifier. The 200-ps transition times are fixed for the full amplitude range of 100 mV to 5V.

The third level of the block diagram shows the HP 8130A output system, which is described in detail in the article on page 85. The variable transition times are generated by two slope generators: a GaAs a fast slope generator and a slow slope generator built using surface mount technology. The HP 8130A output amplifier includes a vernier stage for output swing adjustment and is designed to be extremely linear because of the programmable slopes. A post-attenuator is used to achieve the output voltage range of 100 mV to 5V.

IC Processes

One of the major challenges in the pulse generator project was the difficulty of developing the high-performance, high-speed analog circuits. When the project began, there was little experience to draw upon in the design of broad-

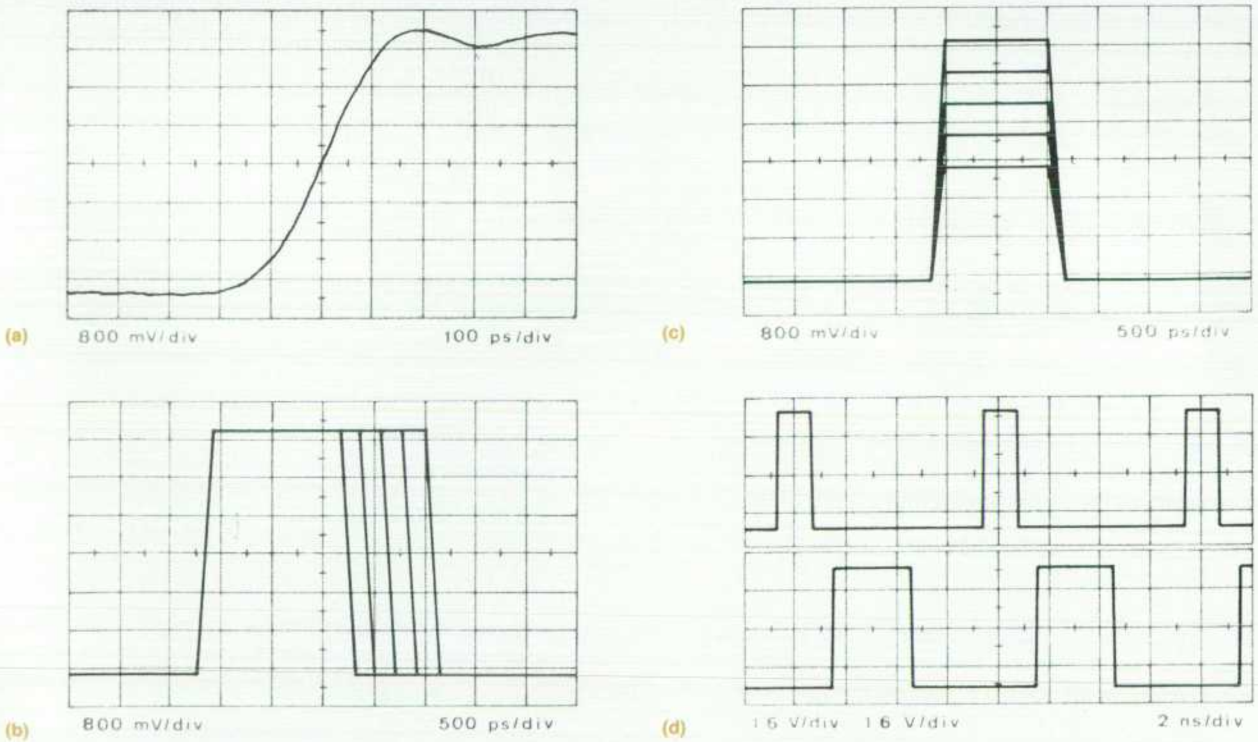


Fig. 3. *Typical HP 8131A output waveforms. (a) 200-ps fixed leading and trailing edges. (b) Variable pulse width. (c) Variable amplitude. (d) Optional second channel.*

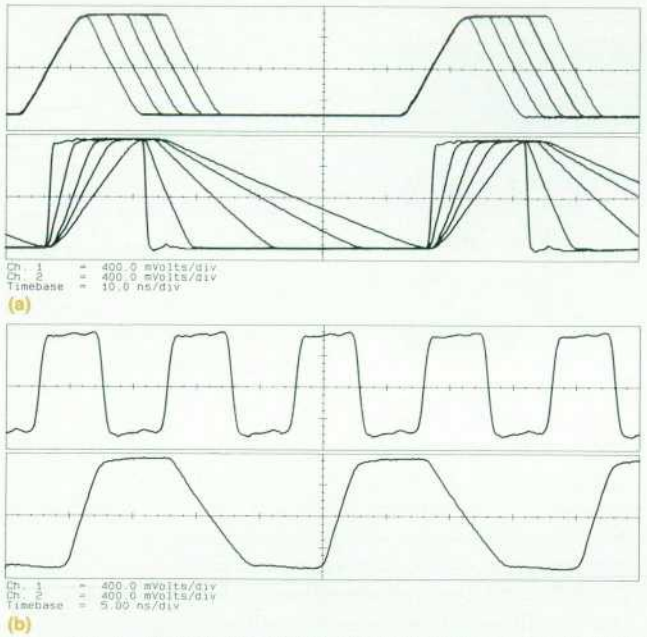


Fig. 4. *Typical HP 8130A output waveforms. (a) Variable widths and slopes. (b) Simulated clock and data signals with optional second channel.*

band linear ICs running from dc to GHz.

We needed a 500-MHz timing circuit, a 600-ps slope generator, and an adjustable output amplifier. A feasibility study to find the best available, most reliable, integrated circuit processes for these requirements led to the choice of two proprietary HP processes.

For the high-speed timing system we chose HP's 5-GHz bipolar process, and for the output amplifier we chose HP's gallium arsenide RFIC process. The design of the timing circuits is described in the next part of this article. The GaAs amplifier design is covered in the article on page 79.

Timing Board

In the HP 8130A and HP 8131A pulse generators, all of the timing for the two 500-MHz channels is generated on a single board. This timing board is used in both instruments. It consists of four major parts: three timing circuits to generate period, delay, and width and the trigger input stage. In the optional two-channel version, the delay and width generators are simply duplicated. Fig. 6 is a block diagram of the timing board.

Timing Circuits

As mentioned above, there are three timing circuits in the standard instrument and five in the optional two-channel pulse generator.

The timing circuits for period, delay, and width generation are very similar. Each consists of one of the 500-MHz bipolar timing ICs described above, a 200-MHz bipolar counter, five current sources, one voltage source, and some control circuitry. The period generator has an additional 10-MHz TTL burst counter. Fig 7 is a diagram of the timing circuit.

Depending on the mode selected by the digital mode control circuitry, the timing circuit can operate as

- A gateable oscillator
- A triggerable monostable multivibrator (one-shot)
- A transparent IC (output signal = input signal).

The timing IC and the counter form the core of the timing circuit. The timing IC is capable of generating a pulse train with a period between 2 ns and 100 ns and a pulse width between 1 ns and 100 ns. This analog timing generation is

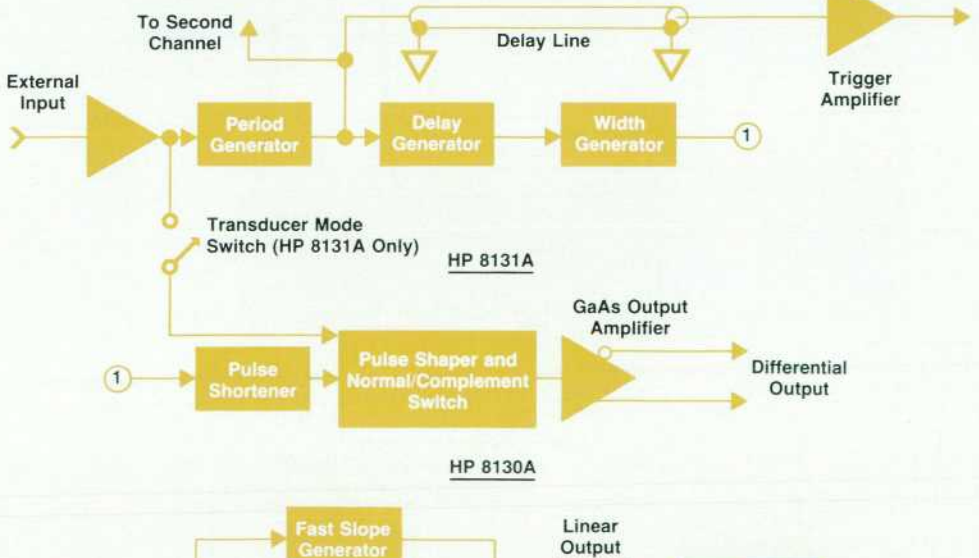


Fig. 5. *Block diagram of the HP 8131A and HP 8130A pulse generators. The top level is common to both instruments. The middle level is the HP 8131A output section. The bottom level is the HP 8130A output section.*

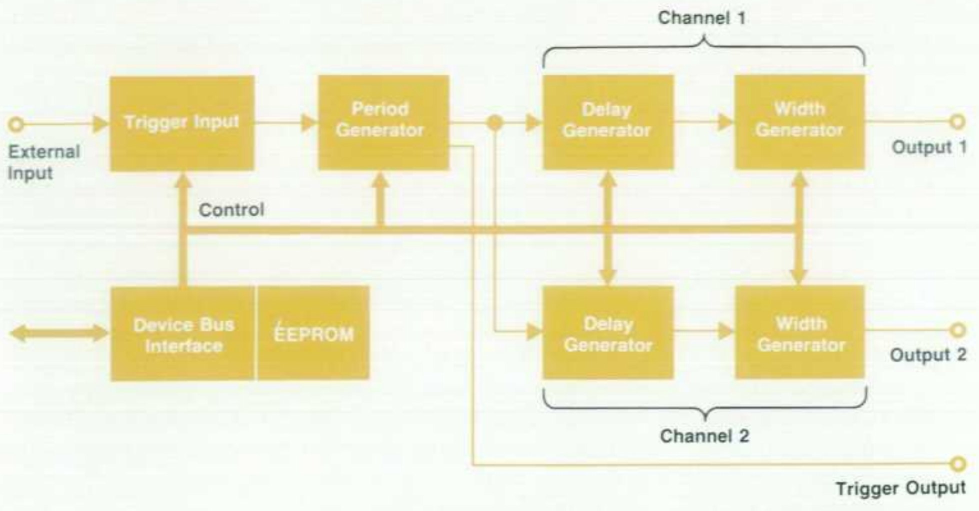


Fig. 6. *Block diagram of the timing board (optional two-channel version).*

split into two ranges. In the first range, timing parameters below 10 ns are generated and the timing resolution is 10 ps. In the second range, above 10 ns, the resolution is 100 ps.

The counter is used to enlarge the period or width by a factor of 10^n, where n is between 1 and 6. This makes it possible to generate timing parameters up to 100 ms.

Like the timing IC, the counter is a custom bipolar IC designed in the HP-5 process.

Timing Parameter Generation

Fig. 8 shows how the period, delay, and width are generated on the timing board in the normal operating mode. The period generator works as a free-running oscillator. Its repetition rate is the period of the pulse train. The output signal is used twice. It supplies the output trigger stage on the output board and it triggers the delay generator. The delay and width generators work as one-shots, both triggered on the rising edges of their input signals. Thus the delay generator triggers the width generator after the programmed delay, which corresponds to the pulse width of the delay generator. The width generator determines the width of the output pulse. The output of the width generator goes to the output board, where it is amplified and becomes the output of the instrument.

In addition to the normally used auto mode, the timing board can be used in an externally triggered mode by switching the period generator to a transparent state. The

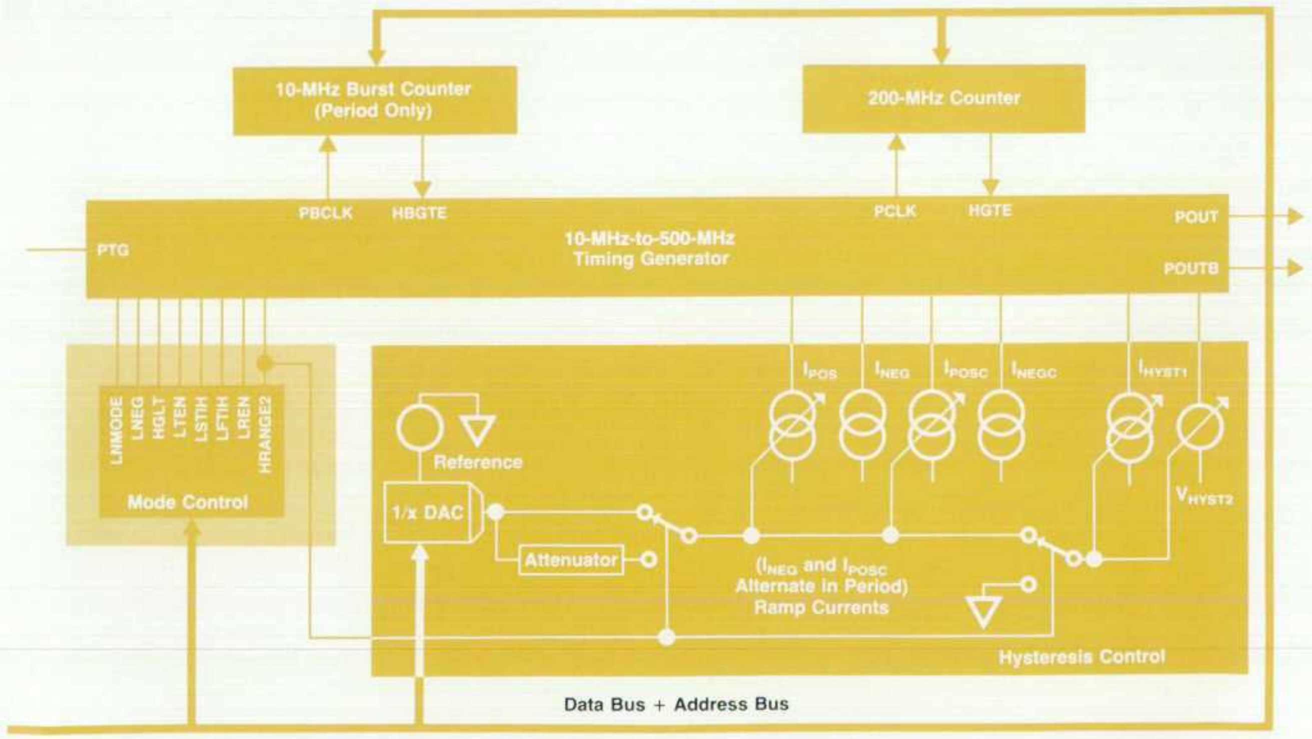


Fig. 7. *Timing circuit (period, delay, or width generator) block diagram.*

external input signal then directly triggers the delay generator and appears at the same time at the trigger output.

In the gated mode, the period generator operates as in the auto mode, but with its gate input enabled. This makes it possible for an external signal to start and stop the period generator. Built-in last-cycle-complete logic ensures that all pulses of a pulse stream will have the same timing characteristics and that no pulse can be shortened by the external input signal.

In the burst mode, the period generator is started by an external trigger event and stopped after a predefined number of pulses, selectable from 1 to 9999. The maximum frequency of 200 MHz in this mode is achieved by using the bipolar counter to count the number of pulses. In the period range above 100 ns, this counter is used to enlarge the timing IC's repetition rate, and an extra TTL counter, the burst counter, takes over the pulse-counting role.

Triggering the width generator not only on the rising edge of its input signal but also on the falling edge results in the double-pulse mode. In this mode the output consists of pairs of pulses separated by a time interval determined by the delay generator. The maximum repetition frequency in this mode is 277 MHz ($t_{per} \geq 7.2$ ns).

For most of the operating modes of the timing board, an external signal must be supplied. To make it easy for the user to trigger the pulse generator on almost any interesting signal, a 500-MHz bipolar trigger input stage is implemented on the timing board. This trigger input can be set to trigger on the positive or negative edge of the input signal at a trigger level programmable in a range between +5V and −5V.

Timing Board Adjustment

The timing board is completely adjusted by an adjust program. Therefore, there are no manual adjustments on the board. All adjustment information is stored in an EEPROM on the board. This method makes it possible to adjust and test the timing board in less than 30 minutes for a two-channel board. Most of the adjustment and test procedures run automatically, so only for about 10 to 15 minutes does a technician have to handle the board. The rest of the procedure is performed entirely by automatic test equipment.

As explained previously, the principle of timing generation is very simple. A capacitor is charged by a certain current. The voltage on the capacitor increases until it reaches the upper threshold of a Schmitt trigger. The capacitor is then switched to a negative current to discharge it. The voltage decreases until it reaches the lower threshold of the Schmitt trigger. The current source is switched again, recharging the capacitor, and the cycle repeats. If the timing IC is used as a period generator, this charging/discharging cycle runs continually, while it runs only once if delay or width is being generated.

To adjust the timing parameters, it is necessary to align the parameters in this circuit, which can vary in production between individual timing circuits. These parameters are:

- The value of the ramp capacitor
- The charging and discharging currents
- The internal propagation delay of the timing IC.

Because of the complex circuitry necessary to stimulate and control the timing IC, at least 10 to 12 potentiometers would have to be implemented to adjust one timing circuit. On a complete timing board, this would result in 50 to 60 potentiometers. To make the problem more difficult, there would be several interdependencies between the potentiometers and there are some limits to observe. Obviously, a better solution had to be found. As already mentioned, the solution was to adjust the timing board automatically, by means of a program. All of the components needed for complete software control were available. The HP 54120A,

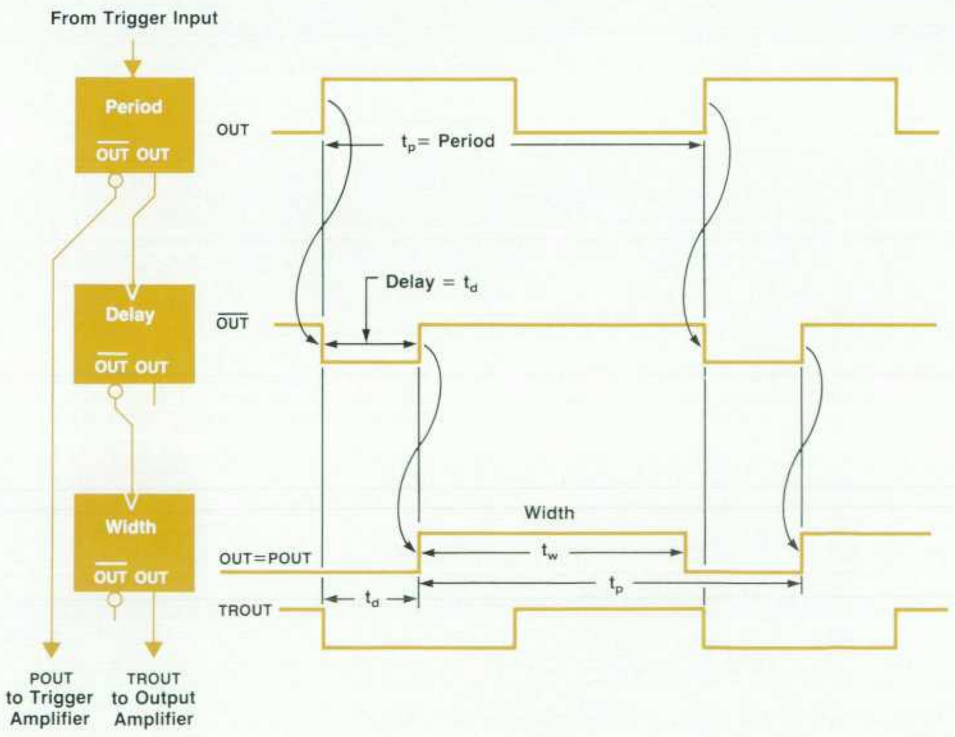


Fig. 8. *Timing parameter and timing generator relationships.*

a fast digital oscilloscope, could be used for accurate high-speed timing measurements, and because the pulse generator was designed to be programmable, digital-to-analog converters (DACs) were already implemented on the board. It was only necessary to make these 12-bit DACs instead of the 10-bit DACs that would normally have been required. The improved DAC resolution makes adjustment possible over a wide range while maintaining the required timing resolution.

Theory

The most direct approach to software adjustment is to find the DAC values for some equidistant timing values within a particular timing range, as shown in the following example:

Period (ns)	: 2.0 2.5 3.0 3.5 4.0 4.5 5.0 ...
DAC Values	: 100 150 190 250 310 350 400 ...

For each timing value, the corresponding DAC value has to be found as accurately as possible. This usually takes a few iterations. Once found, these DAC values can be stored in an EEPROM. The microprocessor in the pulse generator would access these values and calculate intermediate values using linear interpolation.

The disadvantage of this method is that it is necessary to measure a large number of timing values very accurately to ensure small deviations across the entire timing range. For this reason, a faster solution was wanted.

Our solution is to calculate the dependence between each timing parameter and the control mechanism—for example, the period of the timing IC as a function of the DAC value—taking all possible variations into account. A realistic formula for the dependence between the delay or width and the charging current is:

$$T = \frac{C_{ramp}V_{hyst}}{I_C} + kt_{pd} \qquad (1)$$

where T is the timing value (delay or width), V_{hyst} is the voltage window of the Schmitt trigger, C_{ramp} is the ramp capacitance, I_C is the charging current, t_{pd} is the internal propagation delay of the Schmitt trigger, and k is a factor greater than or equal to 2.

The timing value T is proportional to $1/I_C$. To operate the DAC in a reasonable manner, the timing should be proportional to the DAC value. To achieve this, the current has to be proportional to 1/Z, where Z is the DAC value. Therefore, the dependence between the current and the DAC value is given by:

$$I_C = \frac{Z_{max}}{Z}I_{ref} + I_o \qquad (2)$$

where Z_{max} is the maximum DAC value (e.g., 4095), Z is the actual DAC value, I_{ref} is the DAC reference current, and I_o is an additional current offset.

Combining equations 1 and 2, the relation between the DAC value and the timing can be calculated as:

$$T = \frac{C_{ramp}V_{hyst}}{\frac{Z_{max}}{Z}I_{ref} + I_o} + kt_{pd} \qquad (3)$$

or, written in other terms:

$$T = \frac{1}{A/Z + B} + C. \qquad (4)$$

A, B, and C are characteristic values of the function that depend on the individual timing circuit. This formula can be inverted to calculate the DAC value for a given T:

$$Z = \frac{A}{1/(T - C) - B} \qquad (5)$$

This function is implemented in the adjust program for the timing board. The program only has to determine DAC values for a few timing values and can then calculate A, B, and C. These characteristic values are stored in the EEPROM. The system microprocessor in the pulse generator reads them and calculates the DAC values according to equation 5.

The advantage of this procedure is that fewer measurements are necessary than in the normal procedure. In addition, these measurements don't have to be so accurate, because the function doesn't need exact timing to calculate A, B, and C.

Practical Experience

Using this procedure in production showed that it was possible to do an accurate adjustment very quickly. The run time for the adjust program is about 15 minutes for a two-channel timing board.

A few problems were encountered. One of them was that in the highest frequency range, the timing IC's characteristic is nonlinear. Therefore, in this range the algorithm to calculate the characteristic values has to be changed to get good results. Another problem, a disadvantage of doing the adjustment using a relatively complex function, is that if the adjust program fails, it is impossible to correct the values stored in the EEPROM, because they do not represent a single DAC value but a part of a function. For this reason, the program had to be designed very carefully, using error trapping routines to catch any error that might occur.

500-MHz Universal Pulse Timing IC

A custom bipolar integrated circuit realized using HP's 5-GHz IC process provides all of the pulse timing capabilities necessary for the HP 8131A and HP 8130A pulse generators. Fig. 9 shows its functional block diagram. It consists mainly of a spike generator block, a timing generator block, and some logic to control the signal paths.

The spike generator is the first block in the signal path starting with the timing IC's trigger/gate input PTG. Depending on the operating mode (gated, triggered, or transparent) the spike generator either passes the incoming signal unchanged to its outputs or generates short pulses corresponding to either the positive edge or both the positive and negative edges of the incoming signal. The short pulses are generated in an emitter-coupled monostable multivibrator

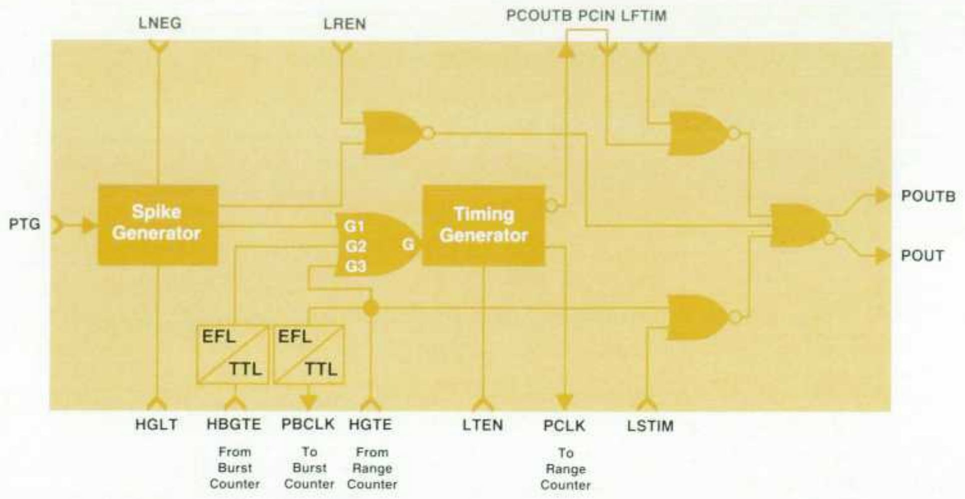


Fig. 9. *Functional block diagram of the timing IC.*

(one-shot) and are typically around 800 ps wide.

In transparent mode the input signal is guided directly to the timing IC's main outputs POUT and POUTB by activating the control signal LREN. In this case the timing generator block is disabled by LTEN.

In all other modes the timing generator operates as a multigated VCO. Free-run mode is obtained by keeping the PTG input active. In triggered mode the short pulses from the spike generator are used to gate one period only, thus operating the timing generator as a one-shot. For frequencies from 10 MHz to 500 MHz or one-shot times from 1 ns to 100 ns, the timing generator's output PCOUTB (connected externally to PCIN) is routed to POUT and POUTB by control signal LFTIM. The external range counter is inhibited. For frequencies lower than 10 MHz or pulse widths larger than 100 ns, the external range counter (clocked by PCLK) is enabled and ORs its terminal count signal via HGTE with the gate signal. The third gate input, HBGTE, is controlled by the terminal count signal of a burst counter (clocked by PBCLK) and is used in the period generator only. PCOUTB and PCLK are externally single-terminated CML outputs. POUT and POUTB are double-terminated EECL outputs with source termination on the chip. Fig. 10 shows an example of multiple gating for the case of a slow four-pulse burst with the range counter dividing by a factor of 4. In reality, the range counter divides by powers of 10.

Spike Generator

To meet the performance specifications of the instruments, a number of performance goals were established for the spike generator. These included the generation of tightly controlled 800-ps-wide pulses at repetition rates exceeding 500 MHz with low pulse jitter, widths independent of the supply voltage, and a low residual temperature coefficient.

Fig. 11 shows a functional block diagram of the spike generator including a simplified schematic of the pulse generation circuitry. With the HGLT mode control input held low, a square wave applied to the PTG input produces two complementary 800-ps-wide pulse trains at outputs D and G1, synchronized to the positive edges of the PTG signal. At the same time, a second complementary train of 800-ps-wide pulses is produced at the terminals X and Y, but synchronized to the negative edges of the PTG signal. Holding HGLT high causes the PTG input to be passed transparently to the D and G1 outputs. In this mode, outputs at X and Y are inhibited. A pulse width control input is provided for fine adjustment of the pulse width and to accommodate the process spread of the on-chip components that define the pulse width.

In the pulse generator mode, a signal applied to the PTG input is level shifted and applied to a differential amplifier composed of transistors Q_A and Q_B. The amplifier outputs are applied to a pair of emitter followers, Q_C and Q_D, which drive the timing capacitor C. Referring to nodes V_1 through V_4, on the negative-going edges of V_1 and V_2, transistors Q_C and Q_D, respectively, will be turned off for a period of time proportional to the slew rate of the voltages at nodes V_3 and V_4. This cutoff period defines the pulse width, and is determined by the value of capacitor C and the currents I drawn by the current sources for transistors Q_C and Q_D. During the time that Q_C is off, the voltage at node V_5 is high. Voltage V_5 is then buffered and directed to outputs D and G1. A second buffer connected to the collector of

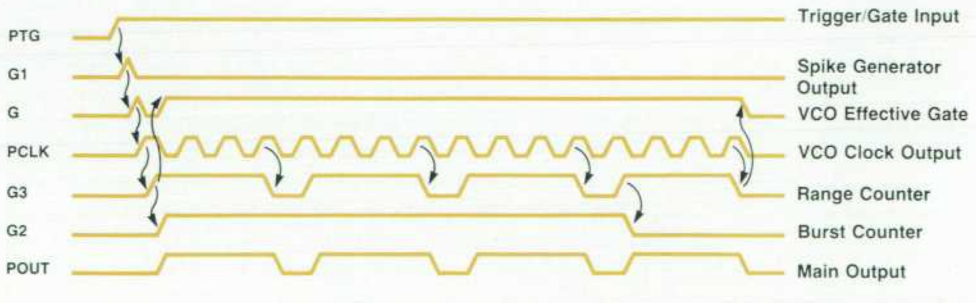


Fig. 10. *Multiple gating by the range and burst counters. In this example, the burst count is 4 and the range counter divides by 4. (In reality, the range counter divides by powers of 10.)*

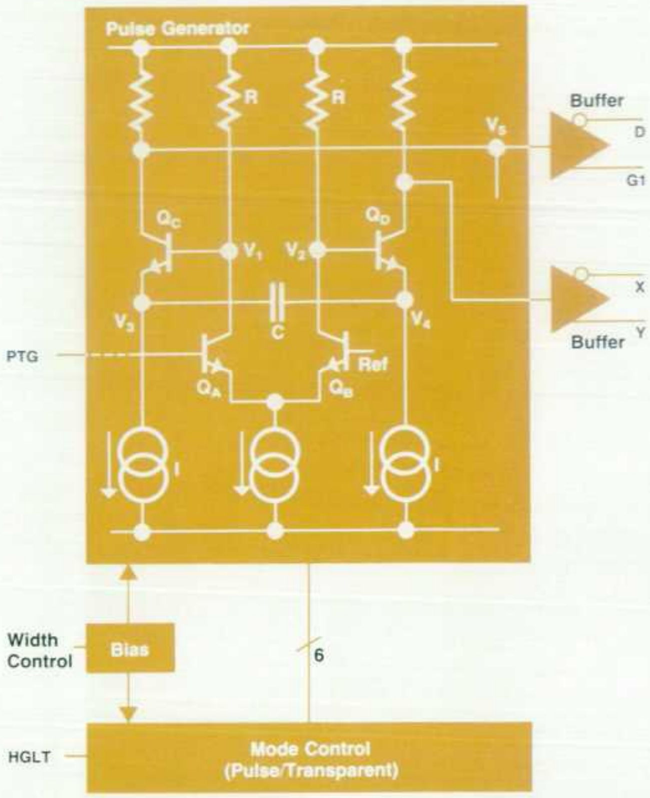


Fig. 11. *Functional block diagram of the spike generator including a simplified schematic of the pulse generation circuitry.*

transistor Q_D provides outputs at X and Y. In the transparent mode of operation, signals applied to the PTG input are amplified, passed with small delay, and wire-ORed to node V_5 where they are also buffered to the D and G1 outputs. Fig. 12 shows simulation waveforms for the circuit in the pulse generation mode of operation. Mathematical analysis of the circuit shows that pulse width can be estimated by the function:

$$t_w = KRC$$

where K is a constant of proportionality determined by on-chip resistor ratios, and R and C are on-chip circuit elements as shown in Fig. 11.

Timing Generator

Fig. 13 shows the functional block diagram of the timing generator together with external control voltage and current sources. In the center is the ramp capacitor C_1, which is charged by current I_{POS} and discharged by current I_{NEG}. These currents are alternatively switched to the ramp node R by the diode bridge, resulting in a triangular voltage at the ramp node. A Schmitt trigger circuit senses the ramp voltage. When the ramp crosses the switching point, the Schmitt trigger switches the diode bridge, thus inverting the current flow through the capacitor. A second capacitor, C_2, can be switched in parallel with C_1 by a saturated transistor to increase the timing range by one decade. With the gate input G low (inactive), the ramp node R is held high by the bridge through the ORed transistors with the Schmitt trigger's inverting output O being low. When the gate signal G becomes high (active), the ramp is released and starts cycling. A comparator compares the gate signal with the switching status of the diode bridge. When the gate signal becomes asynchronously low (inactive), the comparator prevents the downgoing ramp from being interrupted, which would lead to an incomplete pulse. Only if the ramp is upgoing with the gate input low will the ramp be latched at its high level through the diode bridge. This stop state is very stable because of the positive feedback loop formed by the output comparator, the OR gate, and the positive input comparator. Fig. 14 shows the principle of the syn-

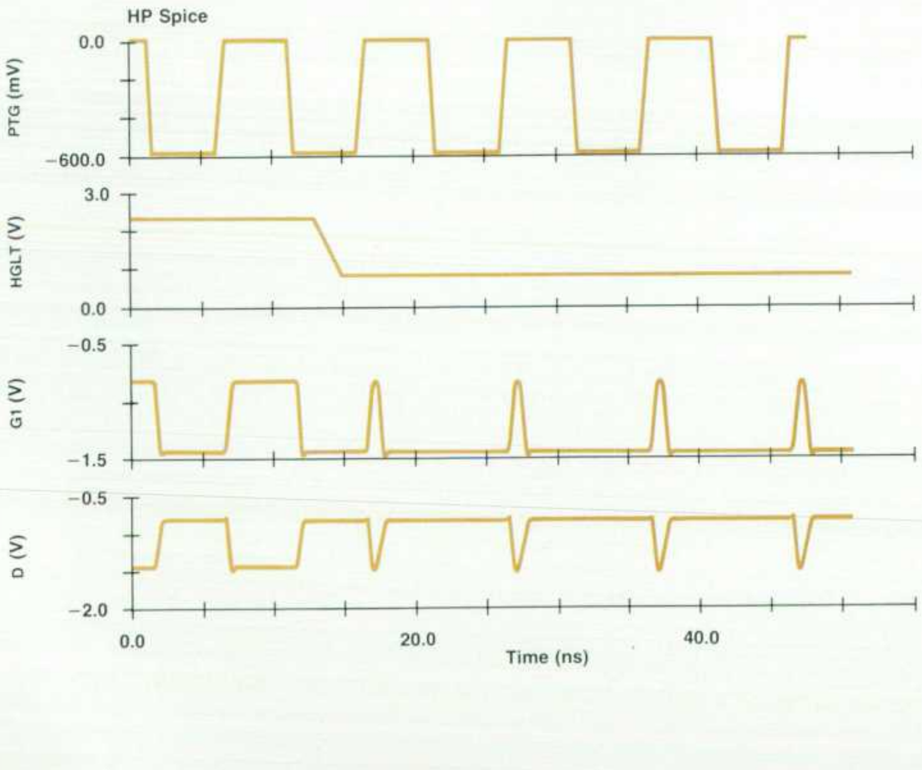


Fig. 12. *Simulated spike generator waveforms in the pulse generation mode.*

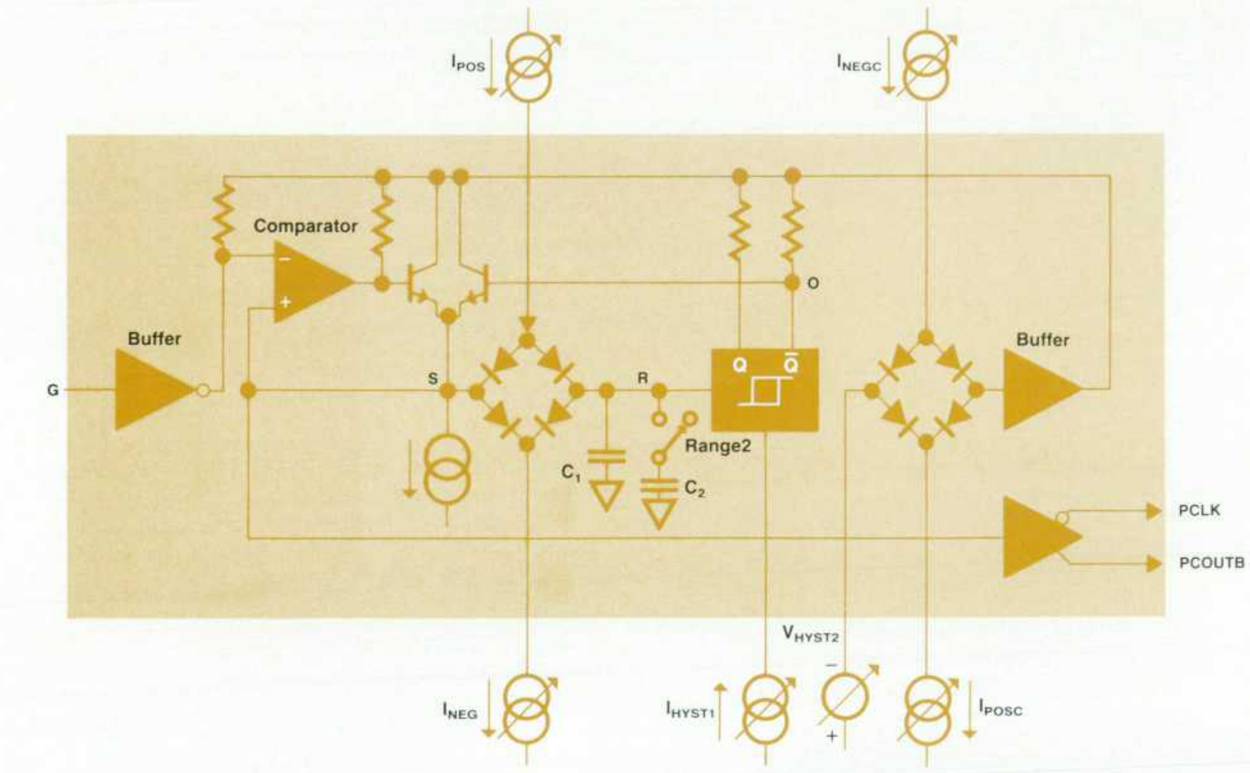


Fig. 13. *Functional block diagram of the timing generator with external control sources.*

chronous gate and trigger operation.

The diode bridge switching signal S is connected to the output amplifier, where it is split into the complementary output signals PCLK and PCOUTB.

The last block is a compensation network with a second diode bridge and inverse charging currents I_{POSC} and I_{NEGC} connected to a buffer. This circuit is used to compensate offset errors of the switching diode bridge and will be explained later.

The hysteresis of the Schmitt trigger can be controlled in amplitude and offset by I_{HYST1} and V_{HYST2}, respectively.

Operating Modes

If the timing IC is used as a period generator, the timing generator works as a VCO and is operated with variable, symmetrical charging currents I_{POS} and I_{NEG}. This leads to an output duty cycle of 50%.

With $I_C = |I_{NEG}| = I_{POS}$, the period is given by:

$$t_{per} \approx \frac{2C_{ramp}V_{hyst}}{I_C} + 4t_{pd}$$

where C_{ramp} is the effective ramp capacitance, V_{hyst} is the peak-to-peak voltage between the Schmitt trigger's switching levels (hysteresis amplitude), I_C is the variable charging current ($I_C = I_{POS} = I_{NEG}$), and t_{pd} is the propagation delay from the time the Schmitt trigger reaches the switching level until the charging current is reversed.

To generate steep ramps for short periods, a small C_{ramp} and a large I_C are used, and the propagation delay t_{pd} becomes the dominant speed-limiting factor. This is also true for the Schmitt trigger's internal positive feedback loop, which becomes a negative feedback loop for short periods because of the phase shift resulting from the internal propagation delay. To compensate for the propagation delay, the hysteresis amplitude V_{hyst} has to be decreased.

However, if V_{hyst} is small, it is necessary to use a very small I_C to get the VCO oscillating with a large period. A small charging current I_C will be more noisy, and this noise

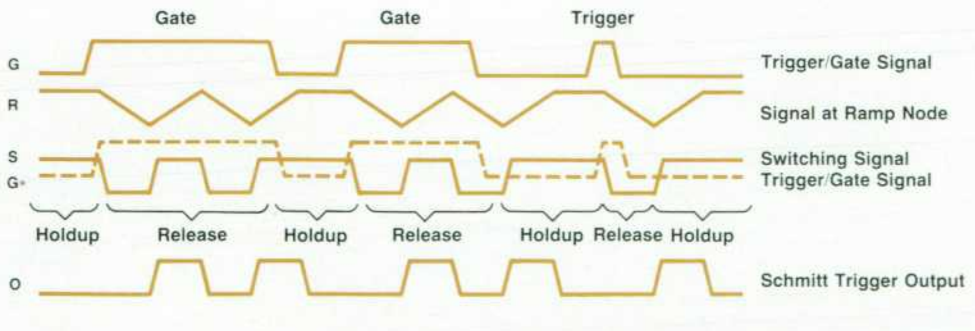


Fig. 14. *Synchronous gate and tigger operation. The last cycle is always completed.*

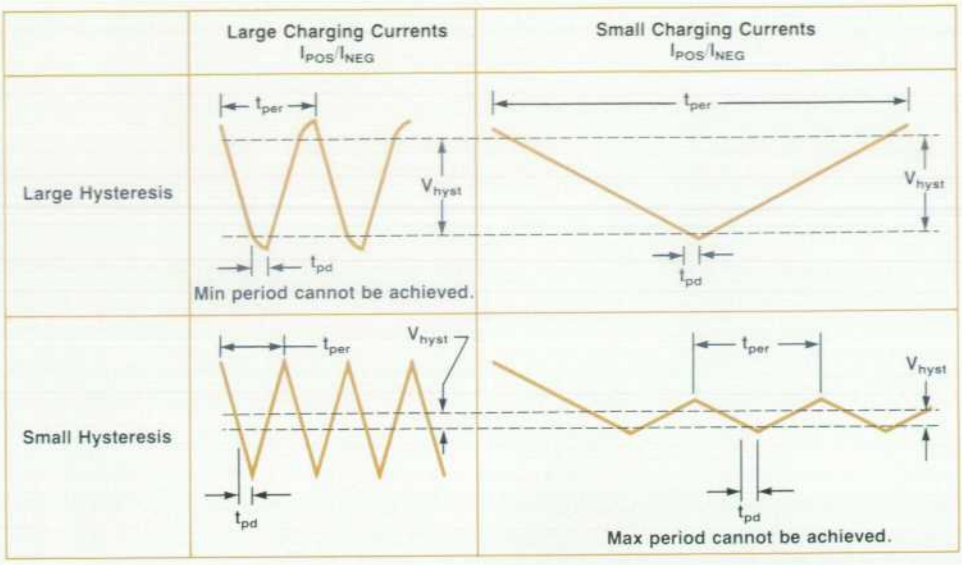


Fig. 15. *Influence of the hysteresis of the Schmitt trigger on the VCO operation with different charging currents.*

will be transformed into high timing jitter by the flat ramp. The small hysteresis together with the flat ramp slope also leads to very slow transitions at the Schmitt trigger's output because of its relatively low closed-loop gain. Fig. 15 shows the influence of the Schmitt trigger's hysteresis on the VCO's behavior for large and small periods. To realize both high speed and low jitter, the Schmitt trigger's hysteresis has to be set according to the period. For periods smaller than 10 ns this is done by controlling I_{HYST1} and V_{HYST2} together with I_{POS} and I_{NEG}. For periods larger than 10 ns, the propagation delay becomes more and more negligible, and the hysteresis can be kept constant.

When used as a delay or width generator, the timing IC has to be operated as a one-shot for widths up to 100 ns and as a VCO for widths larger than 100 ns. The width is determined by the rising ramp only. The falling ramp creates an inevitable dead time, which limits the retrigger period. Because this leads to a limited delay and width range for the instrument, this dead time has to be kept as short as possible. Therefore, a variable I_{POS} is used to control the width and a large, fixed I_{NEG} is used to minimize the dead time.

The dead time is given by:

$$t_{dead} = \frac{C_{ramp}V_{hyst}}{I_{NEG}} + t_{pd}\,(1 + I_{POS}/I_{NEG}).$$

The width is defined as:

$$t_w = \frac{C_{ramp}V_{hyst}}{I_{POS}} + t_{pd}\,(1 + I_{NEG}/I_{POS}).$$

The period in VCO mode or the retrigger time in one-shot mode is:

$$t_{per} = t_{dead} + t_w.$$

Fig. 16 shows the principle of the delay/width operation.

Because of the large fixed I_{NEG} and the variable I_{POS}, the

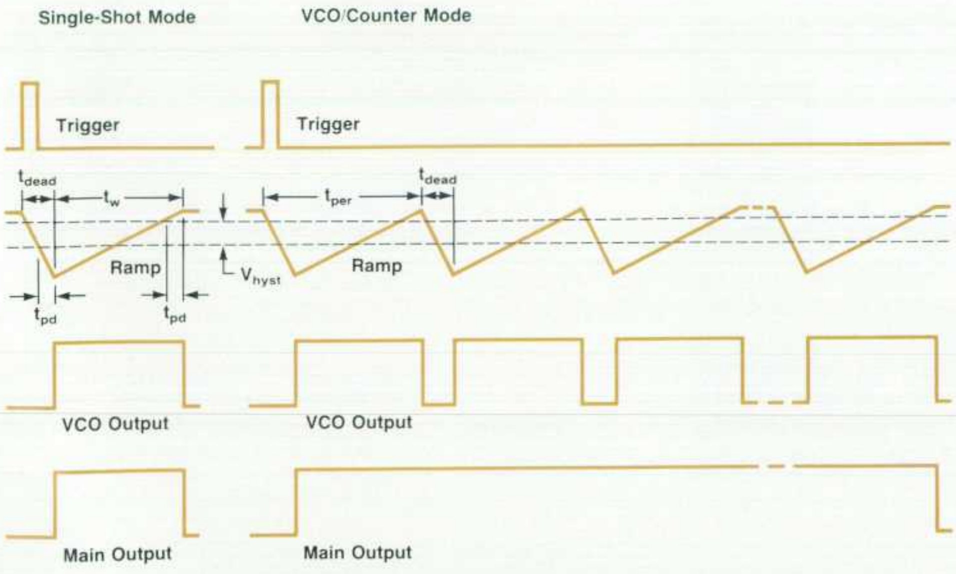


Fig. 16. *Principle of delay and width generator operation.*

diode bridge has to operate with changing asymmetrical currents over the timing range. This leads to a changing offset voltage across the diode bridge. Without compensation of this offset voltage, the rising ramp would be clamped by the diode bridge near or even before the upper switching level of the Schmitt trigger, resulting in increased jitter or even in malfunction. The offset is compensated by offsetting the Schmitt trigger's output signal by exactly the same offset voltage, but in the opposite direction. Because the offset voltage is a fairly complex function of diode voltage drops at different diode forward currents, the easiest way to generate this compensation offset voltage is to use the same diode bridge with inverse charging currents I_{POSC} and I_{NEGC} together with a buffer amplifier. These currents are shown in Fig. 13.

Realization

The timing IC was realized using the HP-5 bipolar process, which has an f_T of 5 GHz. The analog blocks are custom designs down to the component level. For the logical blocks, cells from a cell library were modified and used. The 2.7-mm-by-3.0-mm die, together with four surface mount parts, is assembled into a custom 72-pin multilayer PCPGA (printed circuit pin-grid array) package. Fig. 17 shows the assembled package with the lid removed. The decision to use a custom PCPGA package was driven mainly by the need to place the timing and bypass capacitors very close to the die. Another reason was the possibility of designing 50Ω lines for all signal inputs and outputs and wide, low-impedance lines for ground and power supply purposes. The die is directly mounted on a copper slug connected to a three-fin heat sink. This keeps the junction temperature below 105°C with 2 watts worst-case power dissipation and under worst-case environmental conditions.

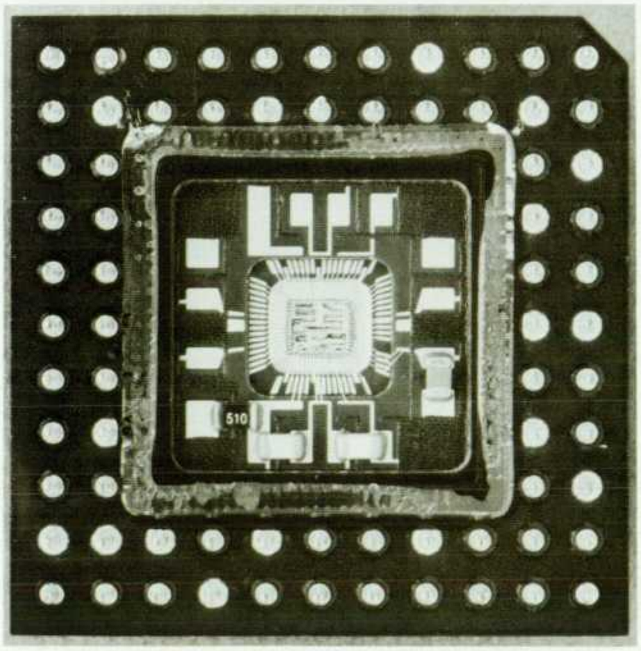


Fig. 17. *The timing IC in its PCPGA package with the lid removed.*

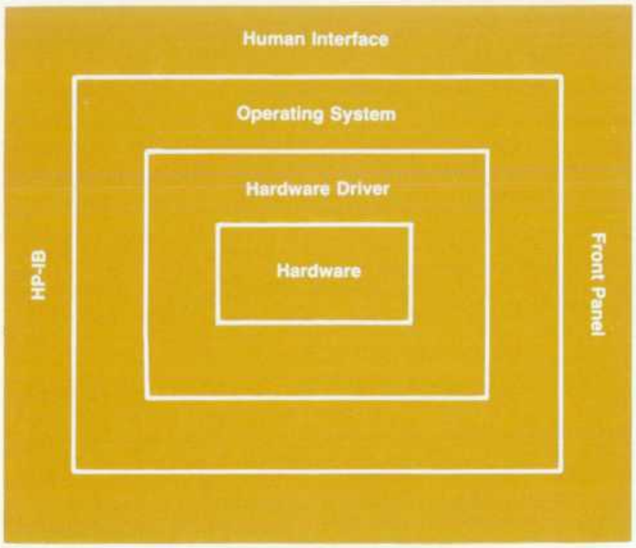


Fig. 18. *Layered firmware architecture.*

Firmware Architecture

The system architecture can be described as a layered model consisting of four layers, the first three software and the last hardware (see Fig. 18). The top layer contains the human interface, which is the bridge between the outside world and the instrument and contains the front-panel con-

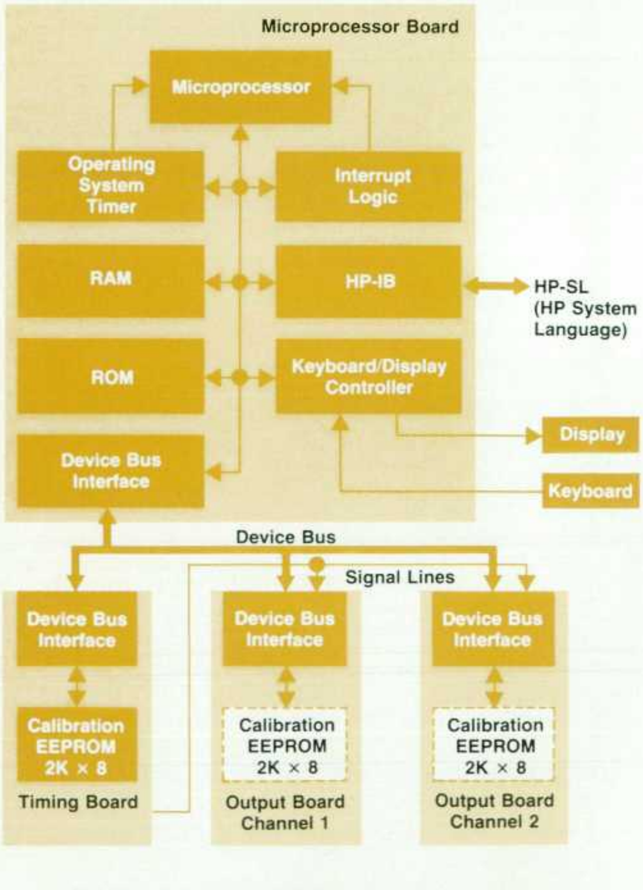


Fig. 19. *Block diagram of a two-channel pulse generator.*

trols and the HP-IB interface. The next layer is a real-time multitasking operating system, which performs the internal system control. The lowest software layer contains the drivers for the hardware.

Each parameter (period, width, etc.) of the device is assigned to a driver. If a parameter has changed, the corresponding driver number will be stored into a buffer called the driver queue. A specific task of the operating system (the hardware task) takes this number and calls the corresponding driver routine to change the hardware state. These driver routines use the adjust values stored in EEPROM on the board to calculate the correct digital values to program the pulse parameters.

The lowest layer of the architecture is the hardware layer. A simplified block diagram of the HP 8131A and HP 8130A two-channel version is shown in Fig. 19. It consists mainly of a timing board, one or two output boards, and the microprocessor board. The boards are connected by an internal device bus. In the HP 8131A pulse generator there is an EEPROM on the timing board. In the HP 8130A there are additional EEPROMs on the output boards for storing calibration constants. All calibration constants are measured and calculated by the adjust program described earlier and are stored into the EEPROMs automatically. The firmware uses these adjust values to calculate the digital values to program the boards.

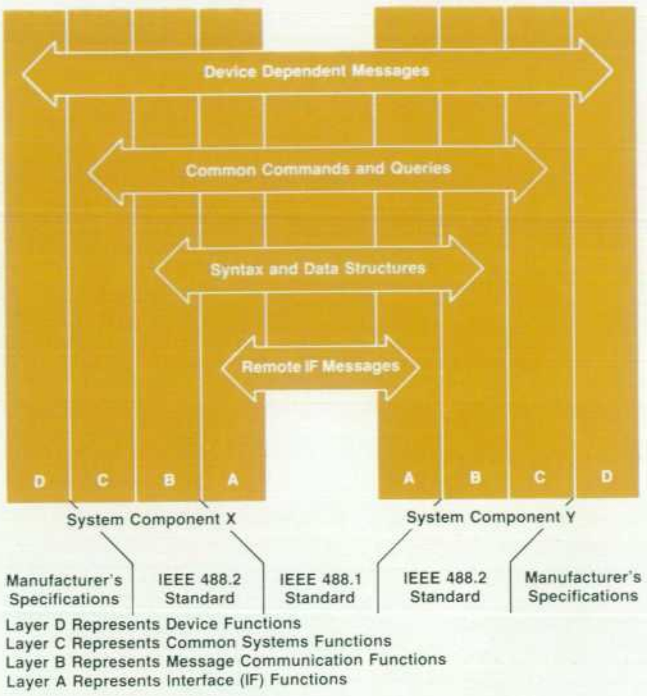
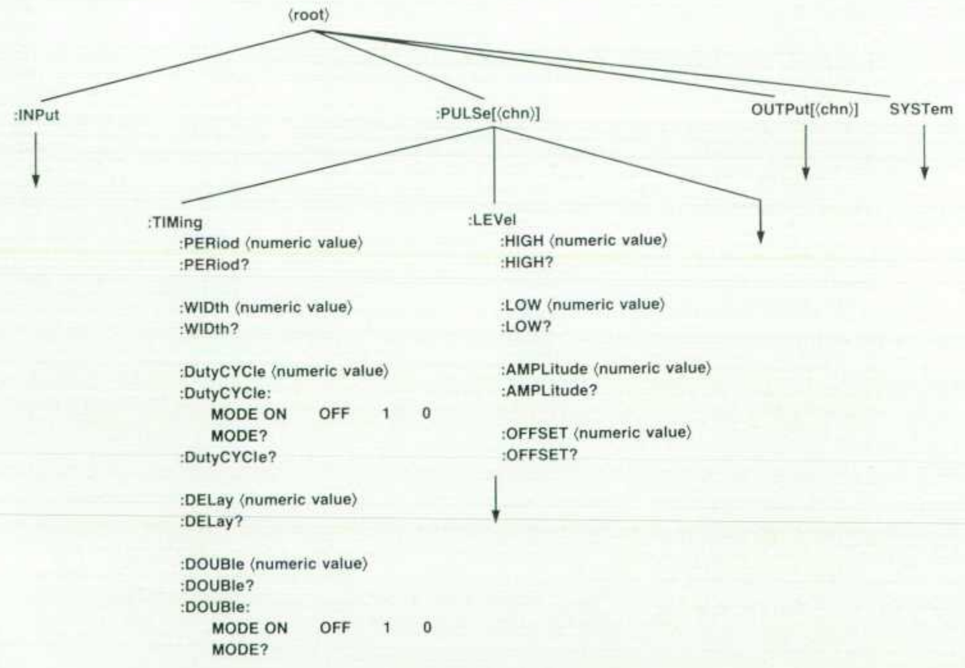

Device Dependent Messages
Common Commands and Queries
Syntax and Data Structures
Remote IF Messages
D
C
B
A
A
B
C
D
System Component X
System Component Y
Manufacturer's Specifications
IEEE 488.2 Standard
IEEE 488.1 Standard
IEEE 488.2 Standard
Manufacturer's Specifications
Layer D Represents Device Functions
Layer C Represents Common Systems Functions
Layer B Represents Message Communication Functions
Layer A Represents Interface (IF) Functions

Fig. 20. *Relationships between IEEE 488 standards.*

HP-IB Programming

All modes and parameters are programmable via the HP-IB. The standard versions of the instruments have one channel. In the optional dual-channel version all parameters are independently adjustable except that the period generator, the trigger logic, the burst counter, and the modes are common for both channels. The HP-IB interface conforms to three standards—two external standards and one internal HP standard. The first is IEEE 488.1, *Standard Digital Interface for Programmable Instrumentation*, the original HP-IB. Because each company using this standard handled message protocol and data formatting in a different manner and instruments were programmed by different command sets for similar functions, the IEEE 488.2 standard was developed. This standard describes a set of codes, data formats, message protocols, and common commands to use with the IEEE 488.1 standard. The instrument interface can be divided into several functional layers, as shown in Fig. 20. The lowest layer is the remote interface messages layer or the IEEE 488.1 bus. This layer is the physical inter-

(continued on page 78)



```
:TIMing
    :PERiod ⟨numeric value⟩
    :PERiod?

    :WIDth ⟨numeric value⟩
    :WIDth?

    :DutyCYCle ⟨numeric value⟩
    :DutyCYCle:
        MODE ON     OFF    1    0
        MODE?
    :DutyCYCle?

    :DELay ⟨numeric value⟩
    :DELay?

    :DOUBle ⟨numeric value⟩
    :DOUBle?
    :DOUBle:
        MODE ON     OFF    1    0
        MODE?

:LEVel
    :HIGH ⟨numeric value⟩
    :HIGH?

    :LOW ⟨numeric value⟩
    :LOW?

    :AMPLitude ⟨numeric value⟩
    :AMPLitude?

    :OFFSET ⟨numeric value⟩
    :OFFSET?
```

Fig. 21. *This subset of the HP System Language is used for HP-IB (IEEE 488) control of the HP 8131A and HP 8130A pulse generators.*

Hybrid Assembly

Custom high-performance integrated circuits often need thick-film hybrids for bonding and providing the IC with its electrical environment. For the HP 8130A and HP 8131A pulse generators, a new kind of hybrid assembly was developed. A simple way was found to connect a high-performance hybrid directly to a printed circuit board. Two of these hybrids are used in the pulse generators. One is a switched GaAs amplifier for the HP 8131A and the second is a slope generator followed by a linear amplifier for the HP 8130A.

Specifications

The basic initial requirements for the hybrid assembly were:

- Transition times as short as 150 ps
- Frequency range 1 to 2 GHz
- Direct contact between the hybrid circuit and printed circuit boards
- Low cost
- Integrated circuit junction temperature less than 125°C at the maximum instrument ambient temperature of 60°C
- Easy serviceability
- High number of electrical connections (about 40).

At the beginning of the project no hybrid assembly was available that matched all these requirements. For low-frequency hybrids with a small number of connections, leadframes are the most common solution. High-frequency hybrids are assembled into flatpacks or plug-ins, which are custom-designed packages in most cases. The hybrid is fixed in these packages and can never be removed for service or upgrading. To change the hybrid, the expensive package has to be removed as well.

Description

The main problem in the design of the hybrid was the signal connections to the printed circuit board. Leadframes seemed to be the best solution, but signal performance degrades because the 50Ω conductor impedance cannot be maintained. It is also difficult to plug the leadframes into the printed circuit board. The solution was found in the elastomers offered by several vendors that have multiple, separate, parallel conductors around their surfaces. The elastomers are covered with a polyimide film and copper foil, which is gold-plated. The copper foil is divided into many conductors separated by small gaps.

The most critical aspect of a hybrid directly assembled to a printed circuit board is its thickness tolerancing. The elastomers are able to absorb all the thickness tolerances, which are up to 0.4 mm for the printed circuit board and 0.3 mm for the remaining assembly. Unlike all former solutions, the elastomers can provide a high number of connections that can also be disconnected easily.

Fig. 1 shows the hybrid assembly. The elastomers are fixed by a molded frame. The frame and a metal plate above it are pressed down to the hybrid and the printed circuit board and are fixed by four screws. The screws can be tightened firmly without cracking the hybrid because the frame is supported by the printed circuit board and not by the hybrid.

The hybrid circuit and the printed circuit board are on the same plane. The elastomer is pressed down by the molded frame onto the surfaces of both components. One part of the elastomer contacts the border area of the printed circuit board and another part contacts the border area of the hybrid. Thus the elastomer connects the two parts like a bridge between two islands.

The metal plate is necessary for supporting the molded frame, which tends to rise a little at higher temperatures. If the frame were allowed to rise, secure connections could no longer be guaranteed. Changing the geometry of the frame would solve this problem, but height limitations do not allow this, so the metal plate was added. It accomplishes its task quite well.

Signals are carried by the elastomers on top of the hybrid. A ground path is provided by a second layer of elastomers from the backside of the printed circuit board to the backplane of the hybrid.

For servicing, only two screws on the backside and the hybrid frame have to be removed. The thick-film substrate is attached to a copper heat sink with an epoxy adhesive. In the substrate is a small hole for a pedestal of the heat sink. The IC is attached directly to the pedestal. This solution was chosen because the power dissipation of the GaAs IC is about 4W. Taking the alumina substrate and the interface between the substrate and the copper heat sink out of the thermal path improves the thermal impedance between the GaAs IC and the heat sink considerably. The thermal performance is a direct result of the excellent heat conductivity

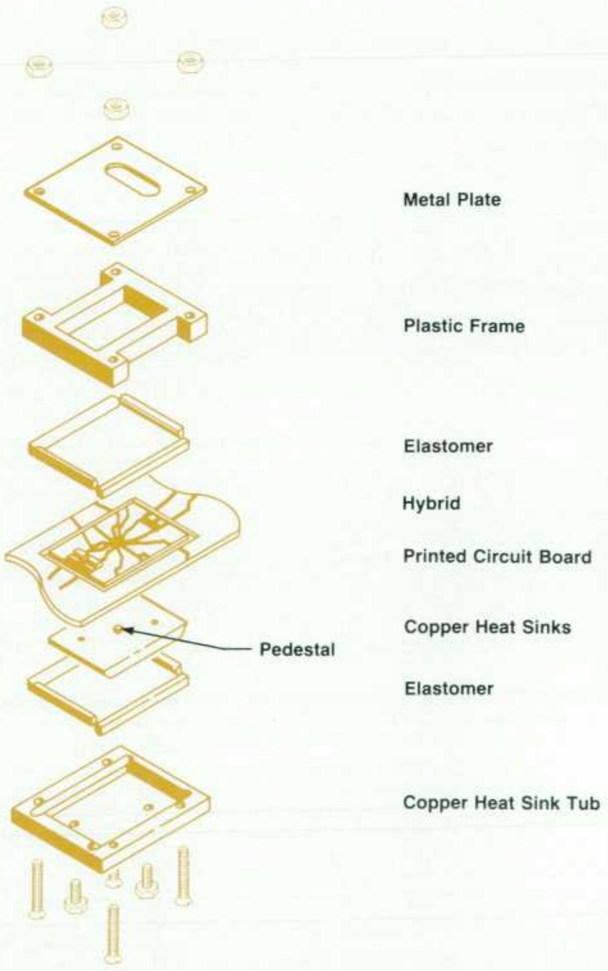


Fig. 1. *Hybrid assembly using elastomers to interconnect the hybrid circuit and the printed circuit board.*

of the copper heat sink.

With the use of copper as a heat sink new problems and discussions came up about the mismatch of the two different thermal expansion coefficients (Table I).

Table I
Thermal Expansion Coefficients

Material	Thermal Expansion Coefficient (10^{-6}/K)
Copper	16.5
Epoxy	40.0
Al_2O_3	6.0

The maximum length difference of the two attached components is about 0.06 mm. Most of the resulting mechanical stress is taken up by the epoxy because of its low die shear strength. In spite of this mismatch, there have been no known failures in the hundreds of hybrids sold in the last two years. These assemblies have passed four different environmental tests and a stringent reliability test.

While a copper heat sink is an acceptable solution for hybrids up to the size of ours, bigger hybrids would require another heat sink material whose thermal expansion coefficient is better matched to the alumina substrate. Two possibilities are Kovar or the new copper-moly-copper composite. Disadvantages of these materials are about a fivefold increase in material cost and about a 50% decrease in thermal conductivity.

At the beginning of the project there were some contact problems with the elastomers. Solder residue, fingerprints, and humidity led to a kind of salt corrosion between the printed circuit board and the elastomer. Cleaning all the parts that may have any contact with the elastomers with propanol appears to have solved this problem.

A hole in the hybrid frame and the hybrid plate permits removal of some peaking bonds in the output path. The bonds can be removed during test without disassembly of the hybrid. The open bonds, which are not protected by a ceramic lid on the hybrid, are sufficiently protected by the frame.

Fig. 2 shows the top and bottom parts of the hybrid assembly with the two hybrid circuits it is designed to contain.

Summary

A simple, economical hybrid interconnection solution has been developed. Environmental tests were successfully passed, and electrical performance up to 2 GHz is excellent. Many connections between a hybrid and a printed circuit board are possible. Hardware design is not limited by the assembly during layout development.

Hans Jürgen Snackers
Mechanical Engineer
Böblingen Instruments Division

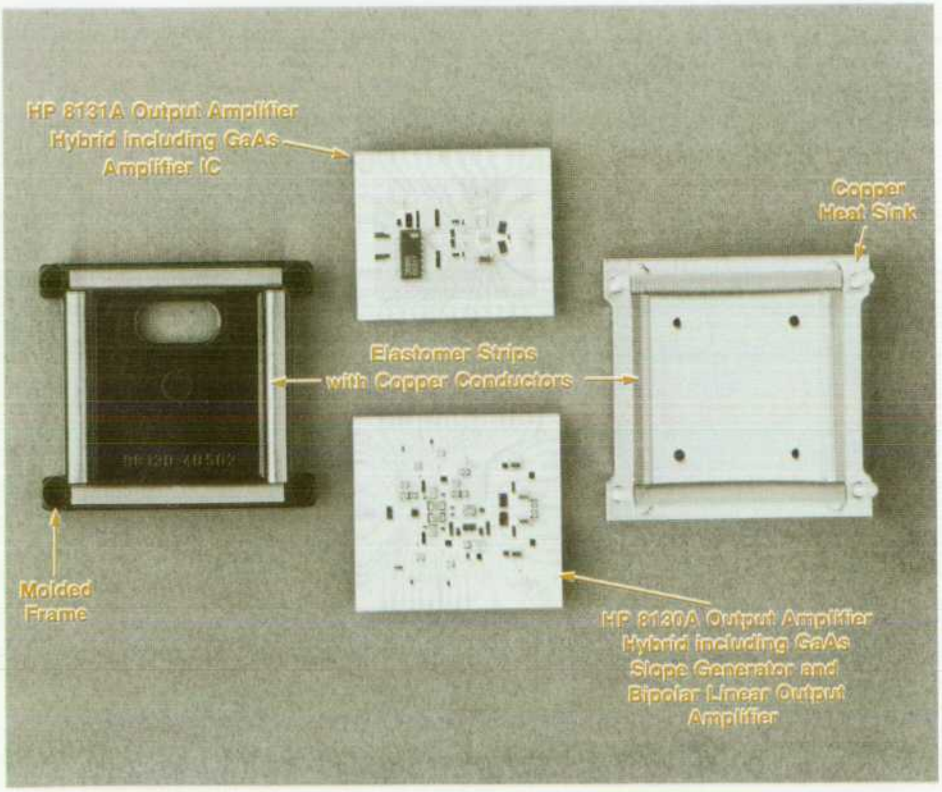

HP 8131A Output Amplifier Hybrid including GaAs Amplifier IC
Copper Heat Sink
Elastomer Strips with Copper Conductors
Molded Frame
HP 8130A Output Amplifier Hybrid including GaAs Slope Generator and Bipolar Linear Output Amplifier

Fig. 2. *Hybrid assembly and hybrid circuits.*

face. It includes the mechanical connector, wiring, and electrical signals. The IEEE 488.2 standard defines the middle two layers. These consist of the syntax and data structures layer and the common commands and queries layer. The syntax and data structures layer defines how data is communicated between devices. For example, it defines the use of the ASCII character set for data representation. It also defines data formats for binary numbers. The final layer is the device dependent message layer, which each manufacturer defines. These messages are the device commands for programming an instrument over the HP-IB.

Based on these standards (IEEE 488.1 and IEEE 488.2) Hewlett-Packard has created an internal standard called the Hewlett-Packard System Language, or HP-SL. This internal standard is written as a designer's guide for messages to be included in HP programmable instrumentaion. Fig. 21 shows the subset of the HP-SL that is used to program the HP 8131A and HP 8130A pulse generators.

Acknowledgments

During this project we received a great deal of good advice and assistance from Tom Hornack's group in HP Laboratories and from Val Peterson's group at the Microwave Technology Division.

The authors want to thank all the people involved in the design, layout, production, and test of the timing IC. Special thanks go to Bill Brown and his team at the HP Santa Clara Technology Center for their support during the design phase. Particular thanks go to Jim Grace for his contributions to the design and to Jean DeGrenier, Kazuko Kikuta, and Jennifer de Neve for taking care of the IC layout. Special thanks also to Mike Oshima, the responsible production engineer, and to Don Riccomini, SCTC marketing, for their efforts to coordinate all the people involved in the design. Thanks also to Joe Garibaldi, test engineer, and Kevin Martin, reliability engineer. Last but not least, particular thanks to Craig Tanner at the HP Colorado Integrated Circuit Division for handling the packaging and assembly issues.

It is impossible to mention all the people who have contributed to make this project feasible, but we would like to take this opportunity to extend many thanks to all the people who were involved.

A 500-MHz Pulse Generator Output Section

Surface mount, thick-film hybrid, and gallium arsenide technologies contribute to the advanced output capabilities of the HP 8131A pulse generator.

by Stefan G. Klein and Hans-Jürgen Wagner

THE OUTPUT SECTION of the HP 8131A 500-MHz pulse generator consists of the transducer board and the output amplifier. This section provides short pulse widths, pulse trains up to 1 GHz derived from external sine waves, output level control, and overvoltage protection. A thick-film hybrid circuit containing a custom gallium arsenide integrated circuit is the main component of the output amplifier.

Transducer Board

The transducer board is the interface between the timing board (see article, page 64) and the output amplifier. All of the components on the board are surface mount devices. The transducer board has the following functions:

- Shaping and refreshing of the input signals from the timing board
- Normal/complement switch
- Generation of pulse widths less than 1 ns
- Restoration of an external sine wave signal in transducer mode.

Fig. 1 shows the functional block diagram of the transducer board. In normal mode, the signal from the timing board goes to one input of the first NOR gate. The function of this NOR gate is to select between the signal from the timing board (all modes except transducer mode) and the signal from the transducer (transducer mode). After the NOR gate, the signal path branches. Both of the resulting paths contain differential amplifiers which shape the signals.

After the amplifiers, one signal path goes directly to one input of the second NOR gate, while the other signal path goes through a 500-ps delay line to the other input of the second NOR gate. Normally, the direct signal path is switched off by control signal HSPU, and the NOR gate works as a simple inverter. When the direct signal path is switched on, the output pulse width is 500 ps shorter than the input pulse width. Fig. 2 shows the principle of short pulse generation. With the short pulse generation circuit the HP 8131A is able to generate pulse widths from 1 ns down to less than 500 ps.

After the second NOR gate, the signal path branches again. One path uses the noninverted output of the NOR gate and the other uses the inverted output. Both paths contain differential amplifiers, which refresh the pulses from the NOR gate. A normal/complement relay after the amplifiers selects between the noninverted path and the inverted path.

A level shifter following the relay shifts the signal to the required output levels, and the line driver, a differential amplifier, produces a differential signal that goes to the output amplifier hybrid. The line driver delivers a signal with more than 600-mV peak-to-peak amplitude and transition times less than 300 ps.

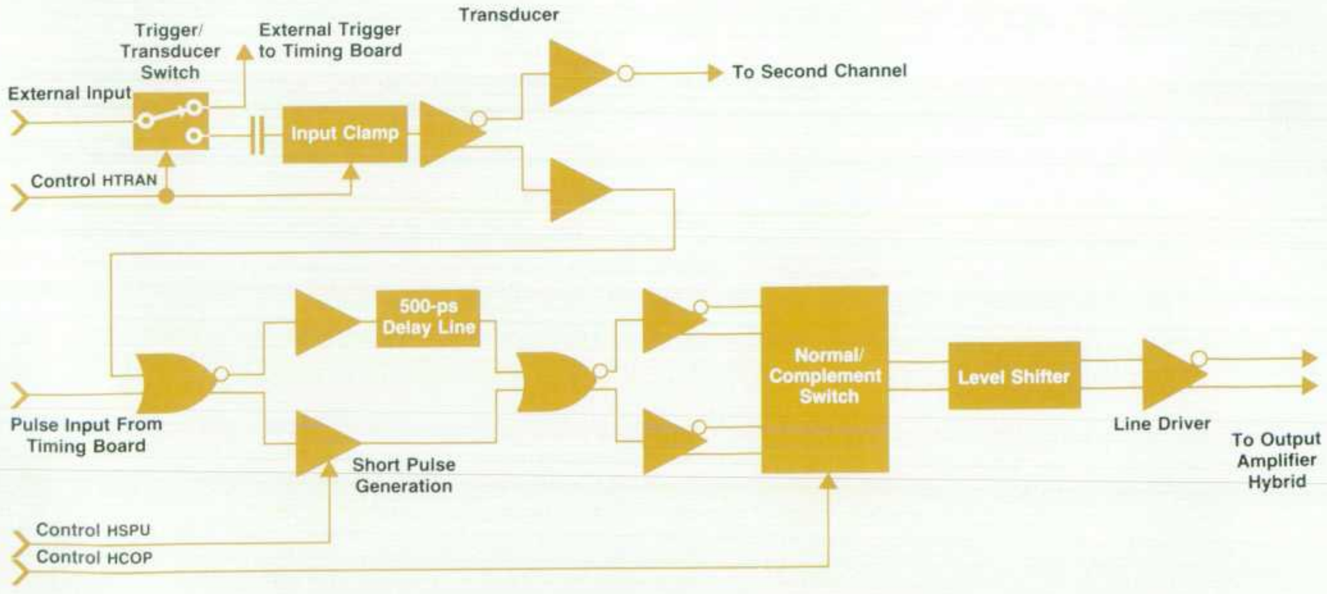


Fig. 1. *Functional block diagram of the HP 8131A transducer board.*

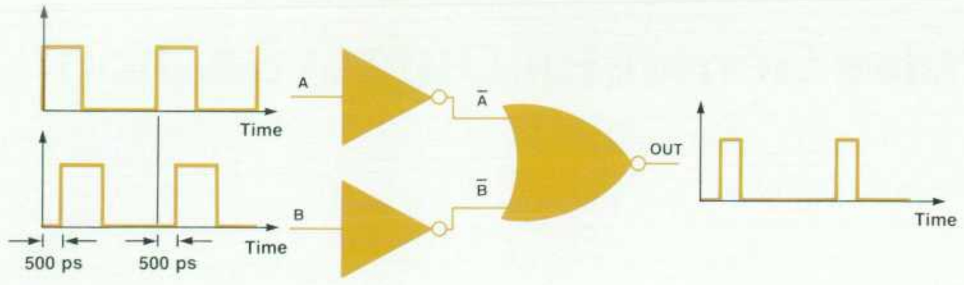


Fig. 2. *Operation of the short pulse generation circuit. The output pulse is 500 ps shorter than the input pulse.*

In transducer mode, the timing board is switched off. A sine wave signal from the external input goes to the transducer, which consists of two cascaded differential amplifiers that are used as a preshaper. To protect the transducer input from high voltages, a diode clamping circuit is used. The shaped signal after the transducer goes to the second input of the first NOR gate. Thereafter, it is processed normally by the following stages and the output amplifier.

Realization

The transducer board is built using a single-sided surface mount process and has a size of 124 mm by 113 mm. The board is mounted on the output board like an IC. The supply voltages, control inputs, and outputs are connected to the output board by means of leadframes. The RF inputs are connected to the timing board by coaxial cables. The key parts of the circuit are the NOR gate IC and the differential amplifier IC, which are packaged in SOIC (small-outline integrated circuit) housings. The 500-ps delay line can be adjusted to compensate for printed circuit board process variations.

Output Amplifier

The main component of the output amplifier is a thick-film hybrid circuit that includes the high-frequency signal path. The hybrid consists of the cascade of a commercial bipolar differential amplifier and a custom GaAs IC (see Fig. 3). The bipolar stage amplifies the input signal to get 1.5V p-p amplitude and less than 350-ps transition times. The GaAs IC consists of three switched, direct-coupled, differential amplifier stages. Also on the hybrid are the input termination, the power supply decoupling, the output termination with peaking inductors, and a level shifting network between the active devices.

The key specifications established for the output amplifier are:

- Amplitude range 0.1V p-p to 5V p-p into 25 ohms
- Rise and fall times less than 200 ps
- Overshoot and ringing less than 15% (10% in most settings)
- Input swing less than 0.6V p-p, differential
- Input transition times less than 1 ns.

Fig. 4 shows the output waveform at the maximum frequency.

GaAs Process

The only process that held promise of meeting the above output specifications at the beginning of the project was the RFIC GaAs process of HP's Microwave Technology Division. It is a 1-μm-gate-length, self-aligned, depletion mode MESFET process with an f_T of 15 GHz. The parasitic reactances of the active devices are small enough to switch large currents with large internal swings in short times. The semi-insulating substrate reduces the parasitic capacitance between the traces and the backside electrode compared to the conducting silicon substrate used in other processes. Transition times below 150 ps within the chip are possible. The large gate-to-drain breakdown voltage allows a large output amplitude. Typical process parameters for a 1000-μm FET are: pinch-off voltage $V_p = -2.1V$, $I_{dss} = 200\,mA$, $g_m = 135\,mS$, $C_{gss} = 2\,pF$, $V_{dg\,max} = 11V$.

The most important question for the design was how to live with the GaAs anomalies in our large-signal, wide-bandwidth, time-domain application. GaAs anomalies are second-order electrical phenomena that occur only with GaAs and not with other semiconductor materials. The most important anomaly for digital time-domain applications is the slow tail problem (see Fig. 5).

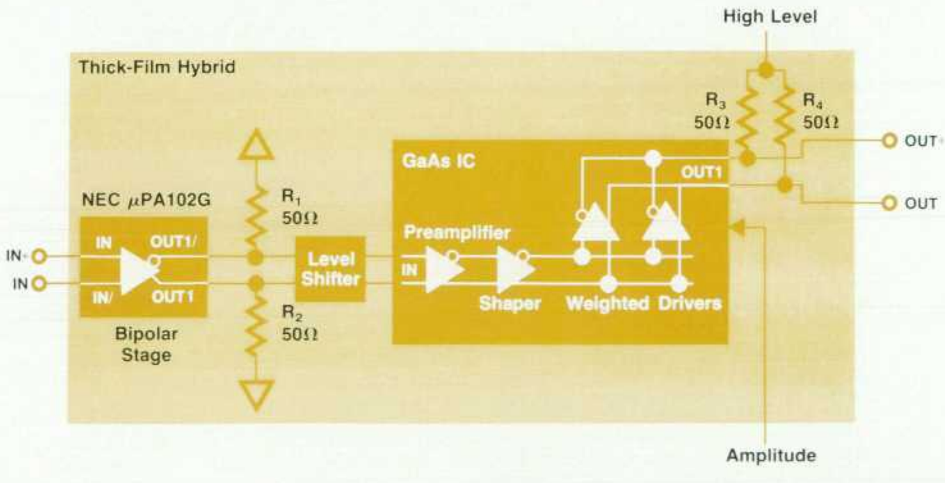
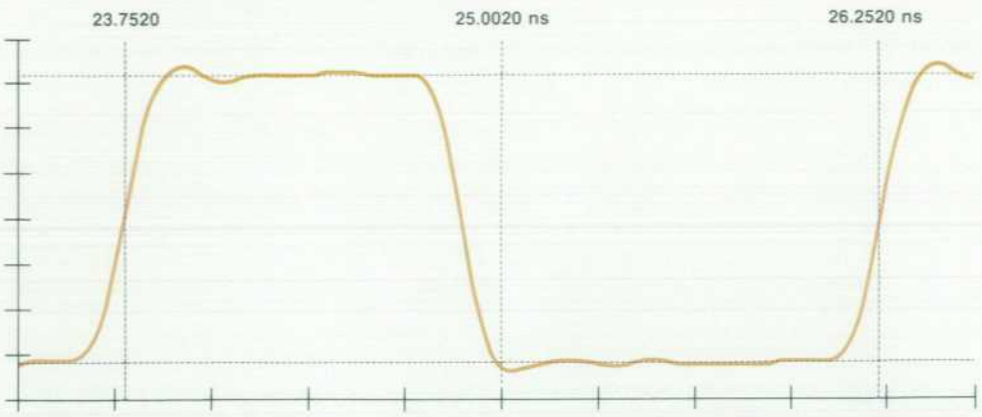


Fig. 3. *Output amplifier thick-film hybrid circuit. A commercial bipolar stage and a level shifter feed a custom GaAs amplifier. Output level control is a function of the weighted output drivers.*



```
Ch. 4             =  800.0 mVolts/div                                       Offset       =  0.000 Volts
Time Base         =  250 ps/div                                             Delay        =  23.7520 ns
Ch. 4 Parameters                          Freq.        =  508.854 MHz      Period       =  1.9652 ns
Rise Time         =  134.4 ps             + Width      =  874.4 ps         - Width      =  1.0908 ns
Fall Time         =  113.2 ps             Preshoot     =  4.000%           Overshoot    =  -3.500%
P-P Volts         =  5.3750 Volts         RMS Volts    =  2.3895 Volts     Duty Cycle   =  44.49%
```

Fig. 4. *HP 8131A output waveform at maximum frequency.*

The slow tail problem is caused by either threshold drift with frequency at the gate node (called gate lag) or variation of the output impedance with frequency at the drain node. The gate lag is measured by stimulating the gate of a FET with a square wave. The drain is biased at 5V and the source is connected to ground through a 1Ω resistance. The swing at the gate is 3V p-p to switch the FET on and off, and the high level is 0V. The voltage swing at the source is small compared to V_{ds} but shows a tail of about 7% of the amplitude with a time constant of about 50 μs. This tail occurs mainly after the FET is switched off. The output resistance r_{ds} of a GaAs FET decreases with frequency to about 30% of the dc value and remains constant for frequencies greater than 10 MHz.

Switched differential amplifiers with resistive loads, gate stages, and cascoded source followers with cascoded current sources are the basic circuits that help cope with the slow tail anomaly (see Fig. 6). A gate stage presents a very high output impedance at the drain:

$$r_d = g_m r_{ds} R_s,$$

where r_d is the small-signal resistance at the drain, g_m is the transconductance of the FET, r_{ds} is the frequency dependent drain-to-source resistance, and R_s is the resistance of the current source. This large output resistance varies with frequency, but compared to the small drain load resistance this variation is neglible, so no tail occurs.

A switched differential amplifier only behaves linearly during the short transition time. After the transition the switched-on FET behaves like a gate stage, so the drain-related slow tail behavior is the same as for the gate stage. For low frequencies and for signals with duty cycles different from 50%, the gate-node-related part of the slow tail problem makes a larger input swing necessary to ensure that the amplifier is fully switched.

The source follower is a linear amplifier. A cascoded device keeps the swing across r_{ds} constant (first-order approximation) so the frequency dependence of r_{ds} doesn't matter. The cascoded current source has a frequency-dependent output resistance that is large compared to the source follower source resistance $1/g_m$. The cascode connection increases the current source output resistance.

GaAs Amplifier

The GaAs amplifier consists of three switched differential amplifiers with resistive loads. The stages are direct-coupled through source followers with diode-based level shifting networks. The preamplifier stage amplifies the input signal (1.5V p-p, 350 ps) to get 3.5V p-p with 350-ps edges. The shaper stage speeds up the edges to get 3.5V p-p with <150-ps edges. The final driver stage switches up to 200 mA in less than 200 ps to get 5V p-p into 25 ohms. It includes an active attenuator that provides an

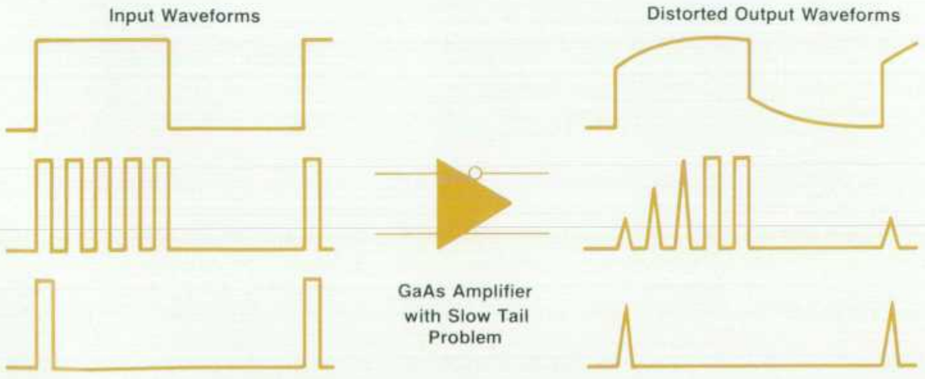

Input Waveforms
Distorted Output Waveforms
GaAs Amplifier with Slow Tail Problem

Fig. 5. *The slow tail problem in GaAs circuits.*

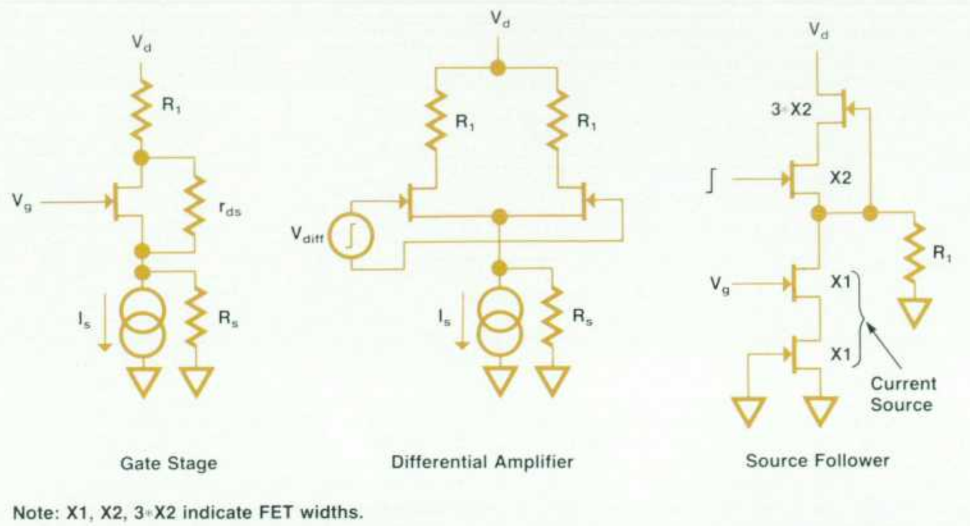


Fig. 6. *Circuits for coping with the slow tail problem include gate stages, switched differential amplifiers with resistive loads, and cascoded source followers with cascoded current sources.*

amplitude range of 0.1V p-p to 5V p-p.

Preamplifier

Fig. 7 is a schematic diagram of the preamplifier stage. The input FETs are biased at a small fraction of I_{dss} so that an input swing smaller than V_p switches the current completely. Because of the large size of the FETs and the large resistive loads, a smaller cascoded gate stage serves as an impedance converter to (1) increase the output resistance compared to the resistive loads and (2) decrease the drain capacitance. The cascode connection of the source follower, which is done to alleviate the slow tail problem, reduces the influence of the gate-to-drain capacitance of the source follower on the load node so that large resistive loads are possible for a given bandwidth specification (>1.2 GHz). To get a small impedance at the output of the preamplifier stage, the series resistance of the level shifting diodes is bypassed by a capacitor.

To force the signal swing at the load resistors to be exactly 3.5V while the resistors can vary by ±15%, a monitor resistor is placed close to the loads. This resistor is measured in a closed-loop system to determine the necessary switching current. To adjust the internal differential offset, the current sources of the level shifting diodes are externally adjustable.

Shaper

The shaper stage schematic diagram (Fig. 8) is very similar to the preamplifier's. A cascoded gate stage is not needed because the resistive loads are smaller and the currents are larger. The source follower stage and the level shift current are larger because the input capacitance of the driver stage is a large 2 pF.

Line Driver

The line driver stage is the direct interface to the customer's load, which is nominally 50Ω. The driver stage determines the output level, rise time, fall time, overshoot, and preshoot. For a nominal customer load the amplitude is programmable between 0.1V and 5V within a ±5V voltage window. For loads different from the nominal, the out-

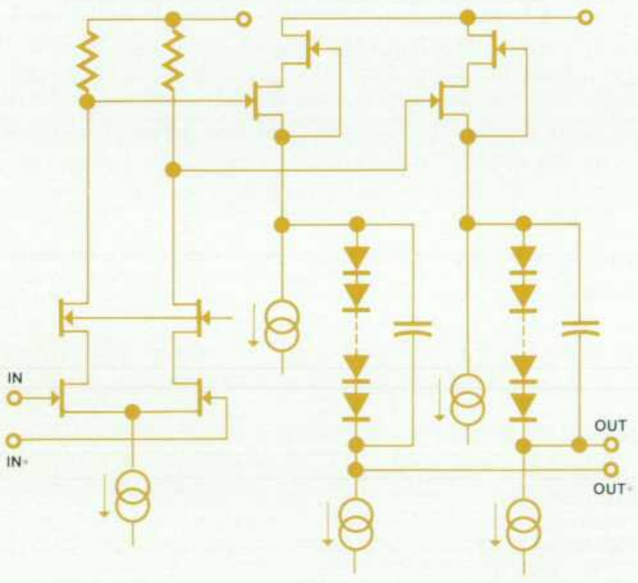


Fig. 7. *Diagram of the preamplifier section of the GaAs IC.*

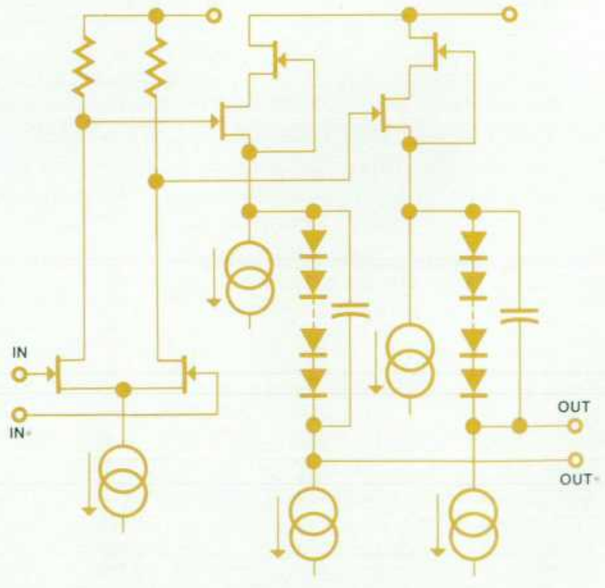


Fig. 8. *Diagram of the shaper section of the GaAs IC.*

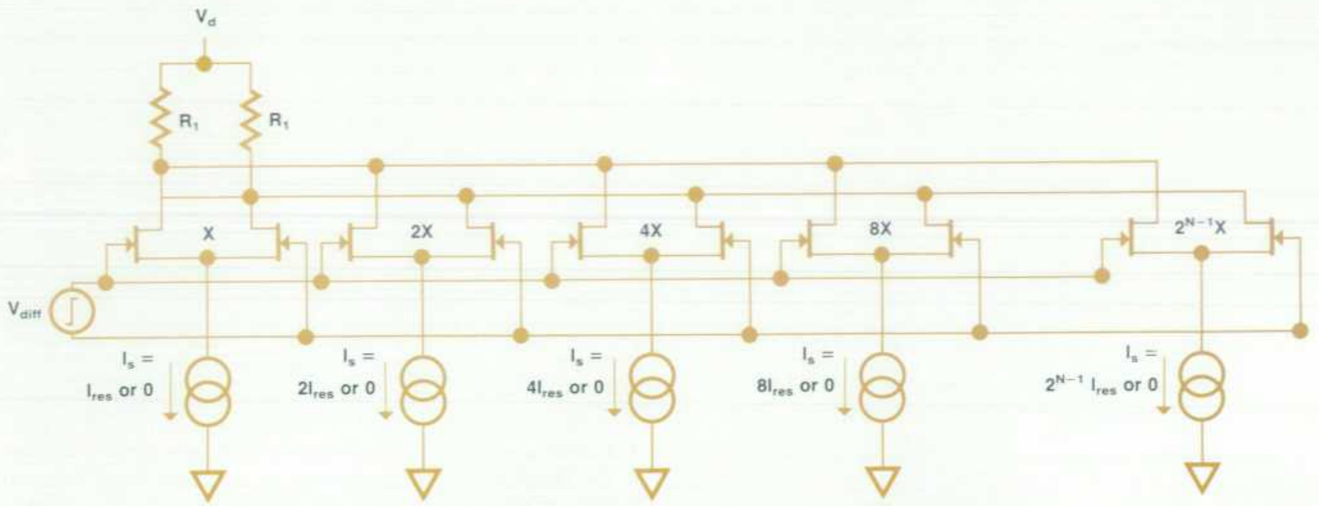


Fig. 9. *An example of the DAC approach to attenuator design.*

put must be protected because reflections can cause the output voltage to exceed the nominal value. Protection is also necessary for customer loads that are connected to an external voltage.

Attenuator. To build an attenuator that has a resolution of 10 mV with fully switched differential amplifiers, two alternative topologies are possible: the DAC approach and the one-stage approach.

Fig. 9 shows the DAC approach. Binary-weighted amplifiers switch currents I_{res}, $2I_{res}$, and so on. Each current source can be switched on and off independently, so any linear combination of currents is possible. For zero amplitude all currents equal zero. If the maximum output amplitude is 5V, then for a minimum output amplitude of 100 mV (linear combination of currents equal to I_{min}), 2% of the total FET size carries current and 98% carries no current. Because the differential pairs are in parallel at the gate and the drain nodes, a large preshoot occurs for small amplitudes because the switching differential voltage has to charge the gate-to-drain capacitance of 100% of the FET size, causing extremly large preshoot currents compared to the signal current I_{min}.

In the one-stage approach, the differential input voltage of a differential pair switches the entire current. The output amplitude is programmed by varying the magnitude of the current source. With a constant input voltage swing, the overdrive of the differential pair increases with decreasing current. The overdrive, coupled with the gate-to-source capacitance and the common source node capacitance, causes overshoot. This overshoot gets dramatically worse at small amplitudes. To maintain constant overdrive, the input drive should be reduced for reduced currents. However, the switching FETs are scaled to drive the largest current. For small amplitudes, the FETs are biased very close to pinch-off and have a small, very flat transconductance. This makes larger overdrive necessary to ensure that the rise time is short enough. The large overdrive and the large parasitic capacitance resulting from the the large FET size cause unacceptable pulse performance.

The solution for this amplifier was a mixed structure (see Fig. 10). Two weighted differential pairs (weight ratio = 7:1) give two amplitude ranges. For the large amplitude range, both pairs switch current, and for the small range the current of the large pair is zero. Within each range the output amplitude is determined by controlling the switched current. Additionally, the shaper and preamplifier swings are adjusted with the amplitude current.

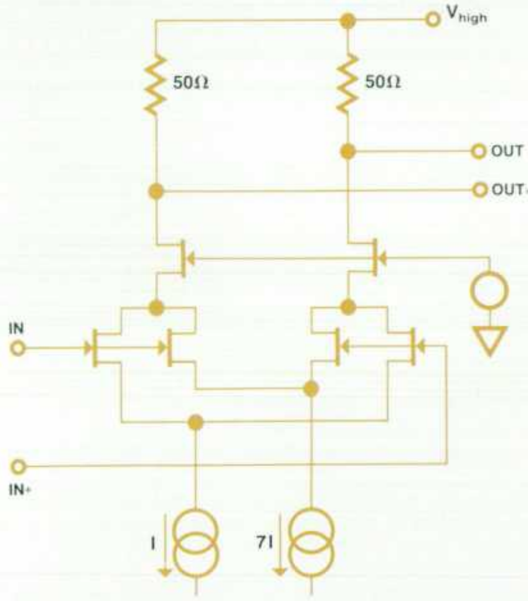


Fig. 10. *The output line driver design is a combination of the DAC and one-stage approaches. It consists of two weighted (7:1) differential pairs that provide two amplitude ranges. Within each range the amplitude is controlled by programming the switched currents. A cascoded gate stage provides overvoltage protection.*

High-Level Control. The high level of the output signal is controlled by varying the common-node voltage of the internal 50Ω resistors. The supply voltages of the GaAs IC float with the high level for all positive high levels.

Overvoltage Protection. The gate stage cascoded with the switching FETs of the driver stage gives two benefits.

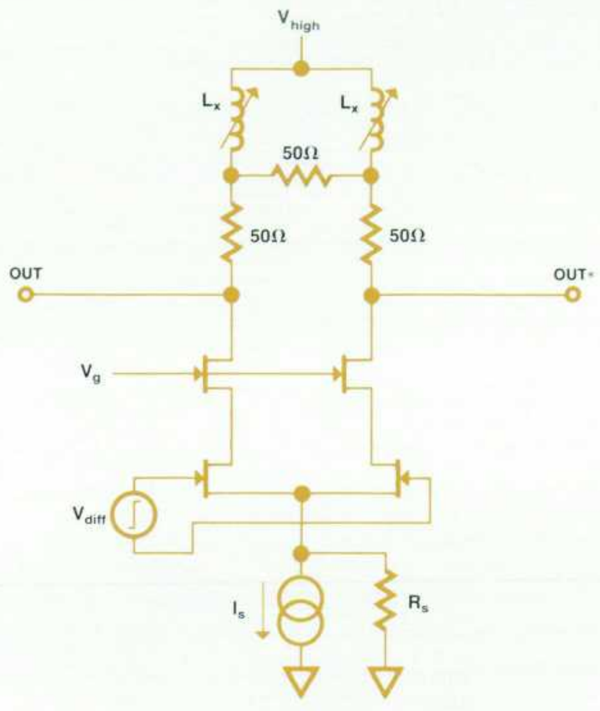


Fig. 11. *Output load configuration on the hybrid circuit.*

The reduced swing at the source node decreases bandwidth limitations at the gate node of the switching pair imposed by the gate-to-drain capacitance. The gate stage also reduces the V_{ds} swing at the switching FETs.

Both external mismatch and external voltages can cause excessive voltage at the drain. For transient overvoltages, ±10V is allowed. For periodic and static overvoltages, a peak detector circuit disables the output and internally terminates the signal. The detector works up to 100 MHz for square wave signals.

Overshoot Adjustment

Within each amplitude range, overshoot is adjusted by controlling the voltage swings at the preamplifier and the shaper. For signals with duty cycles different from 50%, the slow tail problem makes a larger internal swing necessary to ensure that the differential amplifier is fully switched. This causes more overshoot after the transition into the "preferred level" (for duty cycles less than 50% the preferred level is the low level and vice versa). Instead of increasing the voltage swing to overcome the slow tail problem, the differential offset at the shaper stage output is adjusted with frequency. A low-pass filter measures the dc component of the signal, and this information is used to adjust the level shifter current sources accordingly.

A passive overshoot adjustment is implemented on the hybrid as shown in Fig. 11. Because it dissipates 2W, the area of the 50Ω load resistor is larger than the area of the GaAs chip, so this resistor has a large parasitic capacitance. The adjustable inductance L_x in series with the 50Ω load resistor causes the output impedance to increase at higher frequencies. This compensates for the effect of the parasitic capacitance of the load resistor. L_x is printed and can be adjusted by removing bond wires that short the inductance. A 50Ω damping resistor in parallel with L_x eliminates ringing caused by the resonance of L_x and the parasitic capacitance.

Acknowledgments

Thanks to Allan Armstrong and Val Peterson of HP's Microwave Technology Division and Ken Poulton of HP Laboratories for their generosity with their time and expertise in GaAs design.

A 300-MHz, Variable-Transition-Time Pulse Generator Output Section

The design includes separate fast and slow slope generators and custom GaAs and bipolar ICs.

by Peter Schinzel, Volker Eberle, and Günter Steinbach

THE OUTPUT SYSTEM of the HP 8130A 300-MHz, variable-transition-time pulse generator consists of the slope generator and the output amplifier. The programmable rise and fall times are produced by the slope generator, which uses a custom GaAs IC for the fast slopes. The output amplifier is a custom linear bipolar IC.

Slope Generator

A simplified block diagram of the slope generator is shown in Fig. 1. The four basic blocks are the input buffer, the fast slope generator and slow slope interface, the slow slope generator, and the level shifter and impedance converter. The purpose of the slope generator is to generate a signal that combines stable amplitude with variable rise and fall times.

The incoming signal from the timing board is single-ended and has EECL levels. The output signal has programmable rise and fall times from 600 ps to 100 μs. The output amplitude is adjustable from 1.5V to 1.8V.

In the input buffer the signal is split into a slow slope signal path (transition times from 5 ns to 100 μs) and a fast slope signal path. The split was necessary to achieve very fast slopes (below 1 ns) and high-precision slopes for transition times less than 50 ns at the same time. The major goals for the slow slope generator were high transition-time accuracy and high linearity. The major goals for the fast slope generator were fast transition times and a high maximum transfer frequency.

Input Buffer

The input buffer is built with standard components on a printed circuit board using surface mount technology. This technology was chosen to minimize the parasitics (stray capacitances and series inductances) on the printed circuit board. The input signal has transition times less than 600 ps and a voltage swing of 600 mV. The output to the fast slope generator has a voltage swing of 1.6V and transition times less than 650 ps. The edges of the slow slope output are about 1 ns and the output swing is 600 mV.

Slow Slope Generation

The design goals for the slow slope generator were:

- Total range 5 ns to 100 μs
- Leading and trailing edges independently programmable
- Edge ratio 20:1 within one range
- Nonlinearity of slopes less than 2% for slopes greater than 50 ns and less than 5% for slopes between 5 ns and 50 ns
- Slope accuracy better than 5%.

Usually, slopes are generated by charging a capacitor with a constant current (see Fig. 2). The charging current I_{LEE} corresponds to the leading edge of a pulse and the discharging current I_{TRE} corresponds to the trailing edge.

To limit the voltage on the ramp capacitor C_r, a diode bridge is used in front of the capacitor. This bridge serves as a switch controlled by the input step, and is in balance when the voltage V_r at the capacitor has reached the input level. The signal source must have low output impedance to avoid exponential transitions like an RC network's, so an emitter-follower output is used.

Elimination of Offset

In the HP 8130A, to avoid accuracy errors caused by leakage and/or base currents, the ramp is decoupled by an impedance converter with very high input impedance and low output impedance and then fed to a differential amplifier with current feedback (see Fig. 3). The feedback in this stage is large enough to allow linear operation without degradation of the slopes. The other input of the differential amplifier is held at dc. Thus, the single-ended input voltage is converted to differential output currents.

The currents I_{LEE} and I_{TRE} are varied to achieve continuous variation of the leading and trailing edges, respectively. The levels at the ramp node also vary because of the series resistance of the diodes. This can be seen directly at the

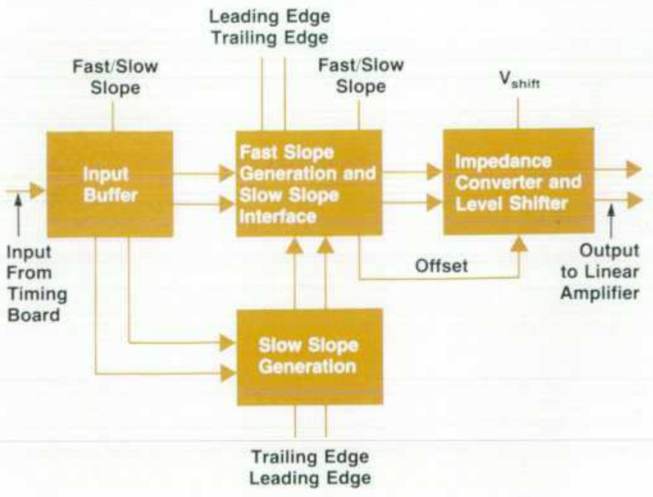


Fig. 1. *Simplified block diagram of the slope generator of the HP 8130A pulse generator.*

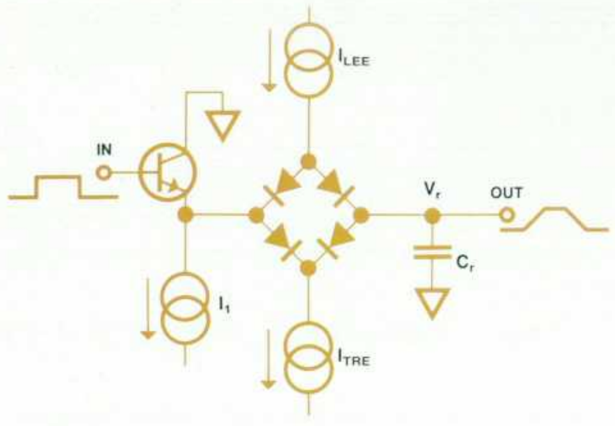


Fig. 2. *Basic slow slope generator.*

output nodes OUT1 and OUT2 as an error in the output signal. To avoid this effect, the other input of the differential amplifier is also connected to a diode bridge. This second bridge is controlled by currents equal to I_{LEE} and I_{TRE}. Slight mismatches of the diode bridges can be adjusted very accurately. The remaining common-mode offset at the amplifier's inputs cannot be seen at its outputs.

Minimum Transition Time

The time during which the ramp capacitor C_r is charged by a current I to some voltage V is given by:

$$t = C_rV/I$$

All three parameters C_r, V, and I contribute to the quality of the ramp.

The ramp can never be faster than the original input signal because the bridge will immediately balance and the voltage at the ramp node will follow the input signal. The speed of the controlling edge is given by the slew rate of the previous circuitry, so the absolute minimum input transition time is achieved at the lowest value of V. On the other hand, a low voltage swing at the ramp node calls for large gain in the following amplifiers, which decreases the bandwidth. Also, the diodes of the bridge have series resistance, and whenever the bridge balances, some rounding of the edges occurs, degrading performance. The amount of this rounding is constant for a given R_s and I, and therefore is relatively smaller for larger V. In view of these effects, the selection of an optimum value of V involves a trade-off between speed and linearity.

As can be seen from the equation above, the minimum transition time occurs for the maximum charging current. The limitation here is also given by the voltage drop across the diodes' series resistance (and, of course, by the current capability of the associated devices). The diode bridge is chosen for an adequate compromise between minimum series resistance and minimum capacitance C_s (see "Capacitive Step" below) at reasonable cost. A GaAs bridge would have met the technical requirements but was found too expensive. Therefore, an HP Schottky quad packaged as a surface mount device was selected.

The values selected for the ramp parameters are C_r = 30 pF, V = 2.4V, and I = 12.5 mA, giving a rise time of:

$$t_r = 0.8(30\ \text{pF})(2.4\text{V})(/12.5\ \text{mA}) = 4.6\ \text{ns}$$

The factor 0.8 accounts for the measurement of transition times between the 10% and 90% amplitude levels.

For a given V and I, the minimum transition time is achieved if the capacitance is as small as possible. The total capacitance is the sum of the lumped capacitance and several parasitics. Since the parasitics are process dependent and temperature sensitive, they need to be kept as low as possible. To keep all parasitic reactances (series inductances as well as parallel stray capacitances) low, the slow slope generator is implemented using surface mount technology.

The complete slope range of 5 ns to 100 μs is achieved by adding additional capacitors. Each range is a factor of 10 higher than the previous one, so the additional capacitor is nine times larger. An edge ratio of 20:1 within one range

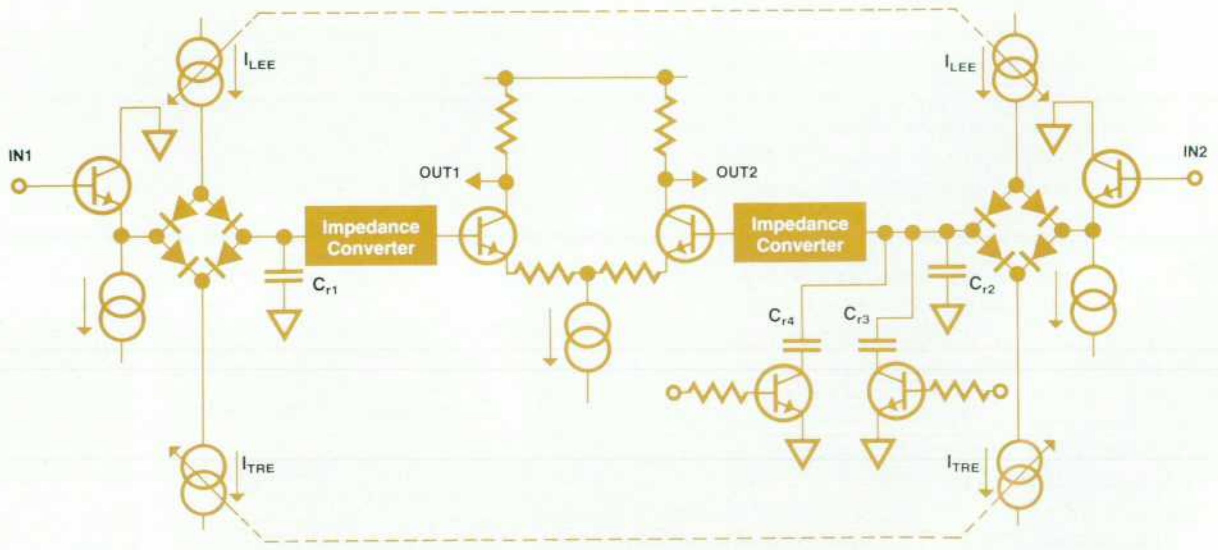


Fig. 3. *Slow slope generator of the HP 8130A pulse generator.*

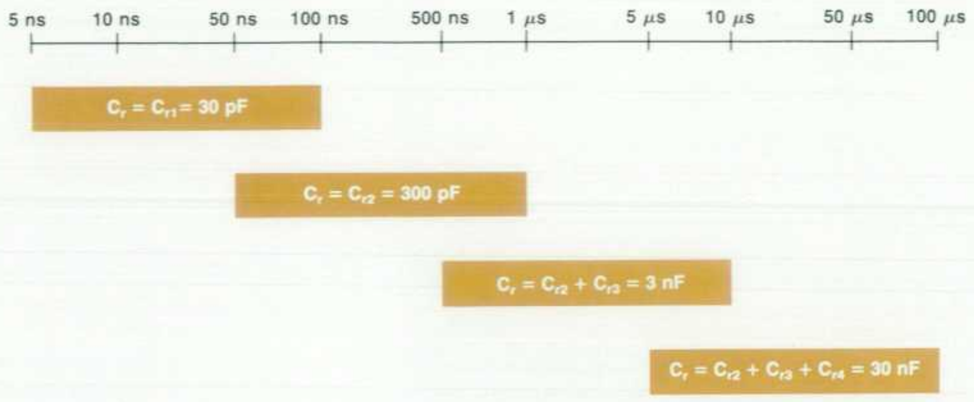


Fig. 4. *Slow slope ranges and the corresponding ramp capacitances.*

can be achieved by setting the charging currents to a ratio of 20:1. This leads to an overlap of the ranges with a factor of two (Fig. 4).

The additional capacitors are switched into the circuit by transistors. However, when the transistor switches are off, they add several picofarads of parasitic capacitance to C_r. To keep the parasitic part of the total C_r as low as possible, the switches are connected at a less sensitive node, that is, at the diode bridge on the opposite side of the differential amplifier (Fig. 3). This works because the second slope range—50 ns to 1 μs—calls for a 300-pF ramp capacitor, large enough that the parasitic capacitance is a small percentage of the total capacitance and does no harm to the accuracy and drift performance. This is also true for the higher ranges. Thus, for slopes from 5 ns to 100 ns, the pulse input signal is applied to the slope generator at its "fast input" IN1, keeping the "slow input" IN2, at dc. For slopes between 50 ns and 100 μs, the pulse input signal is applied to the slow input, keeping the fast input at dc. The range switching is done at the slow input.

Capacitive Step

In practice, between the input and the output of the bridge there is some parasitic capacitance because of the capacitance of the bridge diodes and the stray capacitance of the board. Thus, the input pulse step is also visible at the ramp node with magnitude:

$$V_c = \frac{C_s}{C_s + C_r} V_{in}.$$

Typical values are $C_s = 3$ pF, $C_r = 30$ pF, and $V_c = 0.1V_{in}$.

This is a huge error and cannot be tolerated. C_s has already been minimized, and C_r cannot be increased without degrading the fast transition times. The only solution is to compensate for this effect. This is done by coupling the complement of the input signal through a capacitor of the same value as C_s directly to the ramp node (see Fig. 5).

Fast Slope Generation

The fast slopes are generated by a custom GaAs IC. On this chip are two switched, direct-coupled differential amplifiers, a diode bridge, two sense diodes, two diodes that feed in the slow slope signal, and output source followers (see Fig. 6). The IC is mounted on a thick-film hybrid which also contains the level shifter and the output amplifier. The gallium arsenide process is the same as the one used for the output amplifier of the HP 8131A pulse generator (see article, page 79).

The design goals for the fast slopes were:

- Programmable transition times from 600 ps to 10 ns for pulse widths greater than 1.5 ns. Leading and trailing edges independently programmable for transition times longer than 2 ns.
- Nonlinearity less than 5%.
- Differential output swing adjustable between 1.5V and 1.8V.

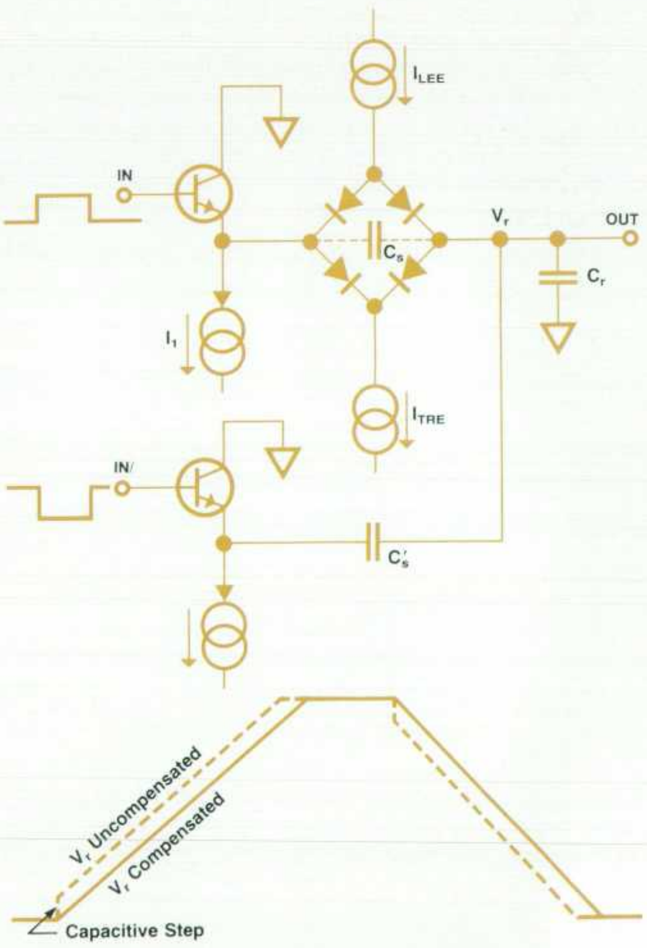


Fig. 5. *Compensation of the capacitive step.*

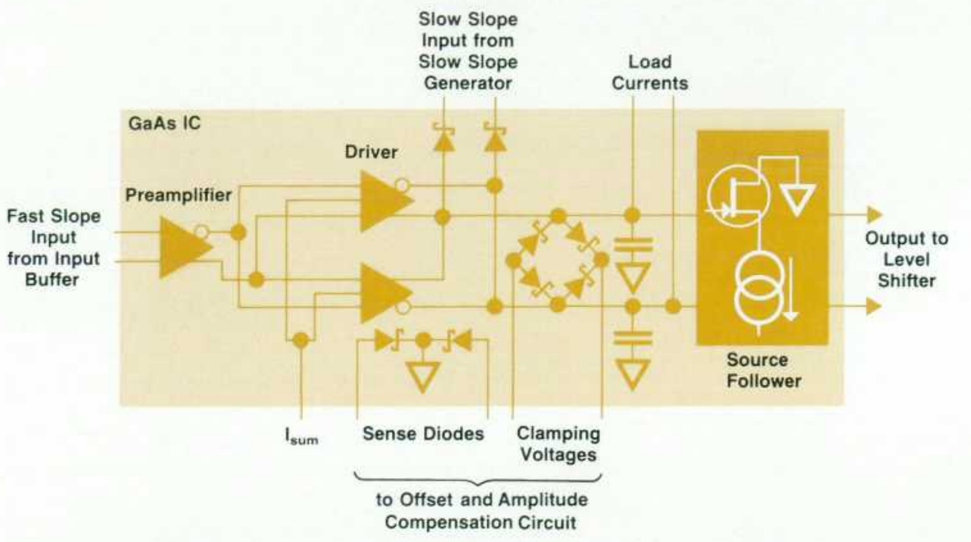

Slow Slope Input from Slow Slope Generator
Load Currents
GaAs IC
Driver
Preamplifier
Fast Slope Input from Input Buffer
Output to Level Shifter
Source Follower
I_{sum}
Sense Diodes
Clamping Voltages
to Offset and Amplitude Compensation Circuit

Fig. 6. *Block diagram of the fast slope generator, a custom gallium arsenide IC.*

- Transition time accuracy 100 ps ±5%.

Input Amplifier. The first stage of the GaAs IC is a shaping preamplifier. The input signal has a voltage swing of 1.6V and transition times less than 650 ps. The outgoing signal has a 3.5V voltage swing and edges less than 400 ps. The schematic diagram is the same as for the preamplifier of the HP 8131A GaAs IC (see article, page 79). Because of the smaller maximum frequency of the HP 8130A, the resistor loads are larger and the FETs are biased with a smaller fraction of I_{dss}. This makes it possible to decrease the transition times and increase the amplitude in one stage.

Driver. The driver consists of two switched differential amplifiers in parallel (Fig. 7). One switches three times as much current as the other. The driver switches the sum of the load currents of the fast slope generator in less than 300 ps. The current, which determines the transition times, varies over a 20:1 range. For slopes below 2 ns, both differential amplifiers operate. For transition times from 2 ns to 10 ns, only the small amplifier operates. Slower transition times are generated by the slow slope generator on the printed circuit board and both amplifiers are turned off. The differential amplifiers are scaled 3:1 to reduce the parasitics of the current switch for low currents to achieve better performance for slopes from 2 ns to 10 ns.

Theory of Slope Generation. Fig. 8 shows how the slopes are controlled. The currents I_1 and I_2 determine the leading and trailing edges independently. I_{SUM} is the sum of I_1 and I_2. At time t_1, transistor Q_1 turns on and Q_2 turns off. The constant current that discharges C_1 is determined by:

$$I(C_1)[t_1,t_2] = I_2 - I_{SUM}$$
$$= -I_1.$$

This provides the slopes of OUT/ from t_1 to t_2, the time when the voltage at OUT/ has dropped to the clamping level V(OUT/)[t_2]. At t_2, diode CR_1 starts to conduct and keeps OUT/ from falling any farther. The voltage relationship is:

$$V(\text{OUT/})[t_2,t_3] = V_{CL-} - V_{CR1}.$$

V_{CR1} is the forward voltage across CR_1. At time t_3, transistor Q_1 turns off and Q_2 turns on. I_{SUM} then flows through Q_2 and the current that recharges C_1 from t_3 to t_4 is:

$$I(C_1)[t_3,t_4] = I_2.$$

At t_4, diode CR_2 starts to conduct and prevents OUT/ from rising above

$$V(\text{OUT/})[>t_4] = V_{CL+} + V_{CR2}.$$

The voltage levels at OUT are determined in a similar manner:

$$I(C_2)[t_1,t_2] = I_1$$
$$V(\text{OUT})[t_2,t_3] = V_{CL+} + V_{CR4}$$
$$I(C_2)[t_3,t_4] = -I_2$$
$$V(\text{OUT})[>t_4] = V_{CL-} - V_{CR3}.$$

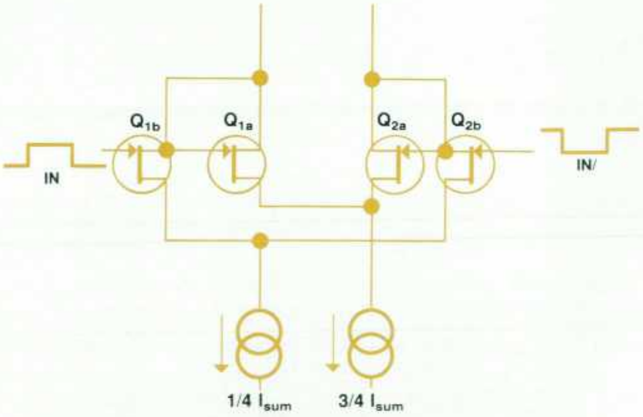


Fig. 7. *Schematic diagram of the slope generator driver on the GaAs IC.*

As mentioned above, the rise and fall times are determined independently by I_1 and I_2. The disadvantage of this circuit is that the amplitude and the differential offset at the outputs are functions of the voltages across the clamping diodes. This means that these parameters are functions of I_1, I_2, and temperature. To compensate for this, two sense diodes CR_{I1} and CR_{I2} are integrated on the IC. CR_{I1} carries the current I_1 and CR_{I2} carries I_2. The voltages V_{CL-} and V_{CL+} are then:

$$V_{CL-} = V_1 + V(CR_{I1})$$

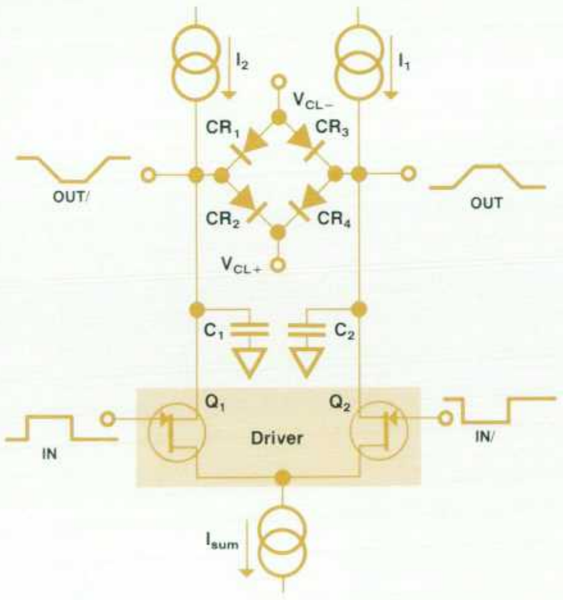
$$V_{CL+} = V_2 - V(CR_{I2})$$

$$V_{ampl} = V_2 - V_1,$$

where V_2 determines the high level of OUT or OUT/ and V_1 determines the low level. V_1 is a fixed voltage. V_2 is adjustable to vary the output amplitude V_{ampl} between 1.5V and 1.8V. This makes the output amplitude independent of the forward voltages of the clamping diodes. The only disadvantage of this circuit is that the differential offset of the output voltages is a function of the ratio of I_1 and I_2. The offset is zero only if I_1 equals I_2 (rise time equals fall time). The offset V_{offs} is equal to:

$$V_{offs} = V(CR_{I1}) - V(CR_{I2}).$$

This offset must be canceled because the linear output amplifier, which is driven by the slope generator, is extremely sensitive to overdrive, especially for slopes between 1.5 ns and 5 ns. To cancel the offset, V_{offs} is fed into the level shifter as an additional control signal. This causes the differential offset between the shifted voltages at OUT and OUT/ to be zero. Assuming ideal matching of all diodes, the output signal is now independent of the currents I_1, I_2, and temperature.



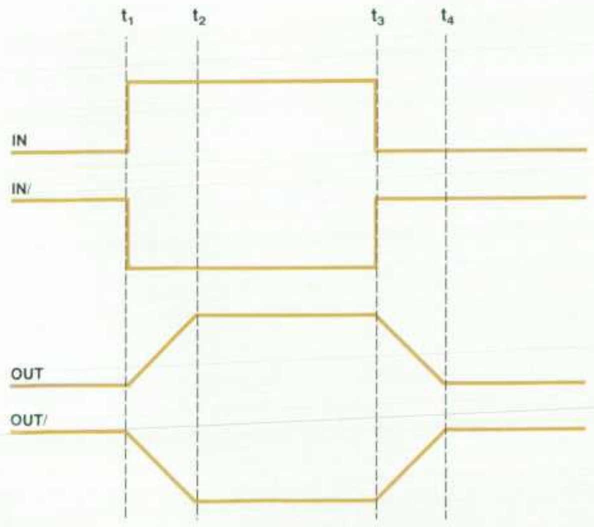


Fig. 8. *Circuit and waveforms for fast slope generation.*

Slow Slope Interface

The slow slope interface of the fast slope generator connects the slow slope signal path with the fast slope signal path. The main problem in the design of the interface was to avoid degrading the fast slopes when the fast slope signal path is turned on. The problem was solved by using small Schottky diodes, integrated on the slope IC itself, to minimize parasitics.

If the fast slope path is operating, the diodes are reverse-biased. When the slow slope signal is being fed into the IC, the driver of the fast slope generator is turned off. The load currents I_1 and I_2 of the fast slope generator are equal (typically 3 mA) and forward-bias the interface diodes. The slow slopes are fed in by the forward-biased Schottky diodes.

Level Shifter and Impedance Converter

Because the output amplifier floats with the programmable output offset, a level shifter with a variable shift voltage is necessary. The input current of the level shifter has to be as small as possible, because every input current causes an error in the transition times. The output impedance has to be less than 50Ω.

The impedance converter is realized by a FET on the IC itself to reduce parasitics and minimize cost. The FET is used as a source follower.

This solution has two disadvantages. First, V_{gs} is a function of temperature, which means that the output level is a function of temperature. Second, the slope nonlinearity tends to increase dramatically because of the droop of the source follower. It was absolutely necessary to avoid these disadvantages, especially the second.

The design goals for the level shifter and impedance converter were:

- Shift voltage 5V to 10V
- Distortion less than 0.5% for rise times greater than 50 ns and less than 1.5% for rise times between 500 ps and 50 ns
- Minimum transition times less than 300 ps.

Fig. 9 is a schematic diagram of the level shifter and impedance converter. As already mentioned, at the input of the level shifter is a FET used as a source follower. To achieve stable operation (independent of the shift voltage) most of the source current (about 40 mA) flows through the on-chip current source. To get a variable shift voltage the current through R_2 is adjustable between 6 mA and 11 mA. The shift voltage is:

$$V_{shift} = V(R_2) + V_{gs}(Q_1)$$
$$= I_{shift}R_4.$$

This means that V_{shift} depends only on the accuracy of I_{shift}, R_4, and U1. The voltage drop across R_1 and R_3 is negligible. U1 is a high-precision, high-speed operational amplifier. It forces a differential input voltage of zero volts at its inputs by controlling the current source I_{LV}, that is, by controlling $V(R_2)$. For high frequencies, the voltage gain G_{HF} of the level shifter is determined by the source follower ($G_{HF} < 1$). To avoid distortion, the level shifter gain must not vary with frequency. Therefore, the input signal is attenuated by the resistive divider consisting of R_1, R_3, R_{V1}, $R_{3/}$, and $R_{1/}$ before it goes to the operational amplifier. This keeps U1 from forcing (by varying I_{LV}) the output swing to equal the input swing for low frequencies. This also means that although the low-frequency source follower gain varies with frequency (this effect is known as droop), U1 forces (by varying I_{LV}) the level shifter gain to remain G_{HF}. With R_{V1}, the gain forced by U1 at low frequencies is adjusted to be equal to the high-frequency gain G_{HF} of the FET.

To reduce distortions at the frequency where U1 starts to work, C_{V1} is adjusted according to:

$$(R_1 + R_3)C_{1par} = R_4(C_{2par} + C_{V1})$$
$$R_{V1} >> R_1 + R_3.$$

This means that both input time constants of the operational amplifier are equal. C_{1par} and C_{2par} are parasitic capacitances of the printed circuit, U1, the hybrid, and the current source. Because the sum of R_1 and R_3 is larger than R_4, the additional capacitance C_{V1} is necessary at the nega-

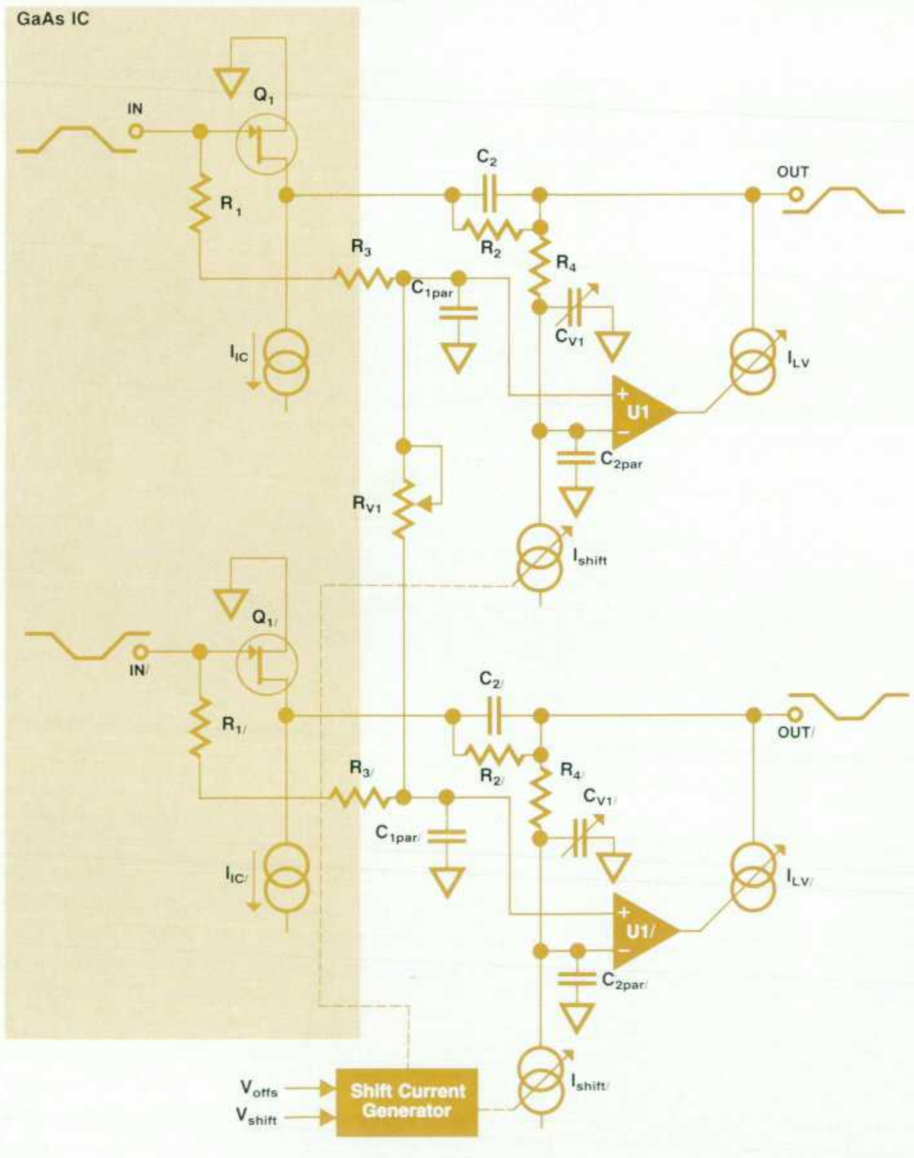


Fig. 9. *Level shifter and impedance converter circuit.*

tive input of U1 (or U1/) to equalize the time constants.

C_2 reduces the dynamic output resistance at high frequencies. This is necessary to feed the following linear amplifier with high-frequency signals. R_1 and $R_{1/}$ are needed for decoupling the signal paths at high frequencies and to avoid parasitic oscillations. Because R_1 ($R_{1/}$) is on the IC and varies with temperature, an additional resistor R_3 ($R_{3/}$), ten times larger, is added to make G_{HF} more stable.

In the fast slope mode, the currents I_{shift} and $I_{shift/}$ are slightly different to compensate for the input offset of the fast slope generator. The difference is adjusted so that

$$V_{offs} = V_{shift} - V_{shift/}.$$

Variable-Gain Linear Output Amplifier

The output amplifier of a pulse generator needs good dc linearity and high speed over a wide range of gain settings, and must handle high output currents and voltages. The linear amplifier in the HP 8130A directly generates the signals that appear at the instrument's output terminals, so it determines many of the instrument's specifications. Design goals that were of particular concern during the development of the amplifier were:

- Output amplitude 0.1V p-p to 5V p-p in a ±5V window
- Minimum output rise time 900 ps
- Output high/low level drift with temperature less than 2% of amplitude
- Output transition nonlinearity less than 3%.

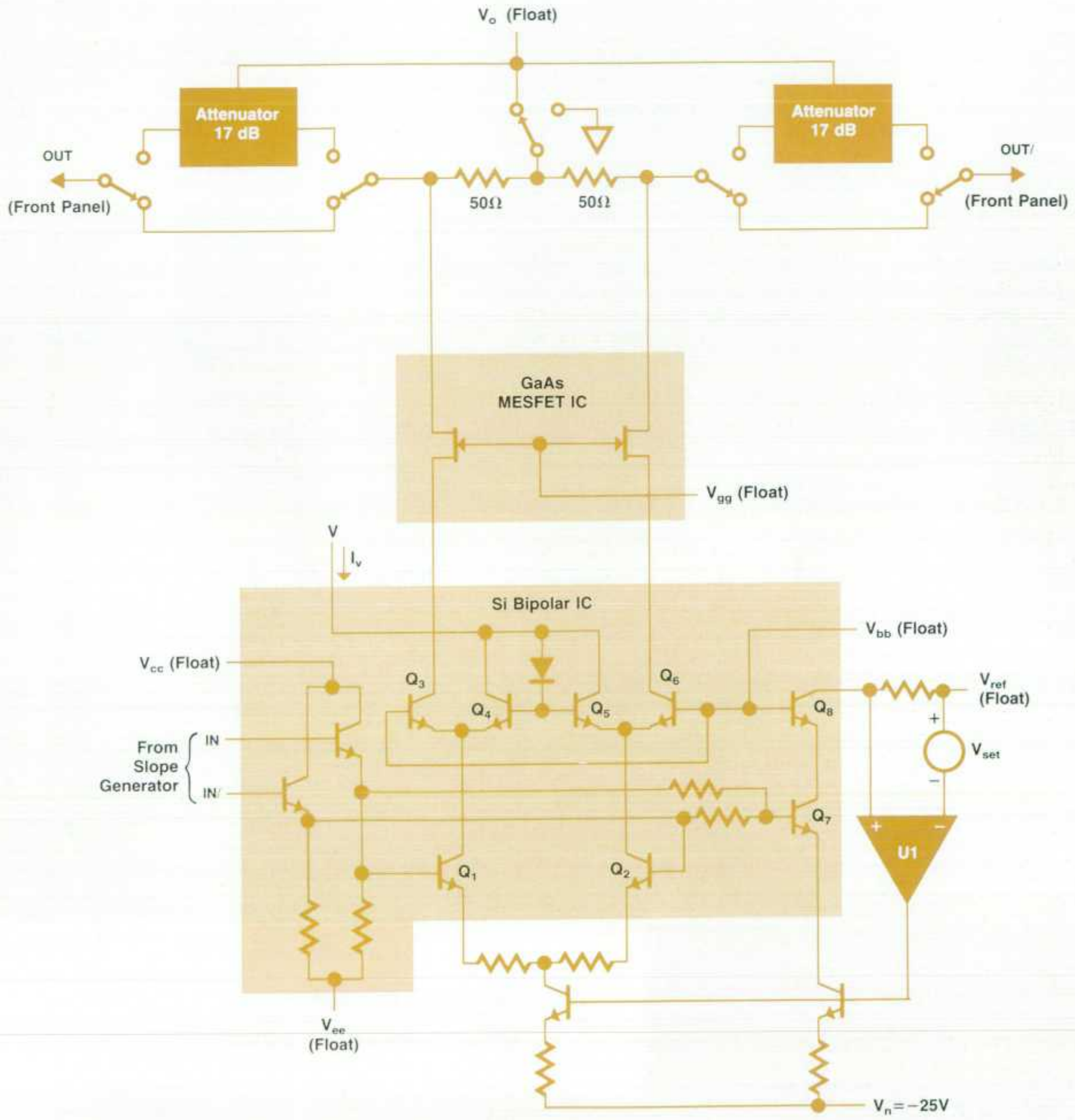


Fig. 10. *Diagram of the output stage with bipolar linear amplifier IC, common-gate GaAs MESFET IC, and postattenuator.*

Current Handling at Speed

The output pulses must be delivered into a 50Ω external load in parallel with a 50Ω termination in the instrument. This means that the amplifier must deliver output current swings up to 200 mA. Offsets up to 5V can be generated by adding a dc current at the amplifier output, so the IC need "only" handle the pulse part of the instrument output.

As in earlier HP pulse generators, the HP 8130A's output amplifier is configured as a transconductance amplifier, turning an input voltage pulse into an output current. This linear amplifier was designed in HP's mature 5-GHz bipolar IC technology to minimize the risk of unpleasant surprises with current handling and speed requirements.

Device Breakdown Voltage

Modern fast bipolar technologies have fairly low breakdown voltages, and in the case of the linear amplifier the collector-to-emitter breakdown voltage specification of 6V was the most critical. Obviously, this clashes with a 10V output voltage range. Floating the whole chip, along with all its supplies and bias voltages, with the output offset reduces the V_{ce} requirement to about 6V, equal to the maximum output pulse amplitude plus 1V for V_{be} and a minimal safety margin. However, nothing prevents the user from letting the instrument output run open so that the amplifier sees a 50Ω load instead of the specified 25Ω, thus doubling the output swing.

Two more measures had to be combined with the floating supplies to relieve this problem. A GaAs common-gate stage at the output with a drain-to-source breakdown voltage of 10V has ample margin to handle normal operating conditions, and a peak detector opens a disable relay when the output amplitude exceeds about 7V p-p, taking care of the open-output case.

Amplitude Range, Linearity, and Stability

The linear amplifier is configured as an emitter-degenerated differential pair with a two-quadrant analog multiplier cell on top (see Fig. 10). The differential pair Q_1, Q_2 acts as a precise, linear voltage-to-current converter. The multiplier cell Q_3, Q_4, Q_5, Q_6 takes care of the output amplitude adjustment since any current fed into its control port V is subtracted from the differential pair's collector currents on their way to the output pads OUT and OUT/.

Unfortunately, when driven with tail currents switched all the way from side to side, these multiplier cells are only linear and dynamically well-behaved over a gain range of about 1 to 1/3 (a gain of 1 means that all the differential-pair current goes to the output pads). This is a far cry from the gain range specification of 50:1.

The solution to the gain range problem is to use three binary-weighted amplifier/multiplier combinations in parallel (only one is shown in Fig. 10), each running at a gain of either 0 (off) or 1/3 to 1 (on). Their currents are 4/7, 2/7, and 1/7 of the 200-mA total. All three together cover amplitudes of 5V to 5V/3 = 1.667V. Switching the largest one off results in amplitudes from 5V × 3/7 = 2.86V to 5V × 1/7 = 0.71V, and the smallest one alone covers 5V/7 = 0.71V to 5V/21 = 0.24V. Finally, an external attenuator takes care of the reduction down to the minimum amplitude of 0.1V.

An amplifier/multiplier stage is switched off by sourcing enough current into its multiplier control input to force its gain to 0. Obviously, at that gain, linearity is not a concern anymore. All stages that are on, that is, have a gain greater than zero, are run at the same gain to avoid settling anomalies.

Correction is required to meet the drift specification of <2% over a 50°C temperature range. In the linear amplifier, the temperature coefficient of the base current error is reduced by a feedback loop that uses the external op amp U1 as well as two on-chip dummy transistors Q_7 and Q_8 running at the same current densities as the amplifier and multiplier transistors, respectively.

Acknowledgments

The authors would like to take this opportunity to thank all of the people involved in the project. Particular thanks go to HP's Microwave Technology Division, especially to Allan Armstrong, who designed the GaAs FET cascode and provided significant support in the design of the slope IC, and to Don Montgomery and Matt Borg, who tested the GaAs ICs. The hybrid circuit is manufactured and tested at HP's Colorado Springs Technology Center. Special thanks to Dale Pittock for his contributions to the testing and characterization of the hybrid.

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
August 1990 Volume 41 • Number 4
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

```
Fr: DICK DOLAN/BLDG16PUB        00072856
                                     446
To: LEWIS, KAREN
    CORPORATE HEADQUARTERS
    DDIV  0000   20BR
```

CHANGE OF ADDRESS: To subscribe, change your address, or delete your name from our mailing list, send your request to Hewlett-Packard Journal, 3200 Hillview Avenue, Palo Alto, CA 94304 U.S.A. Include your old address label, if any. Allow 60 days.

5953-8582