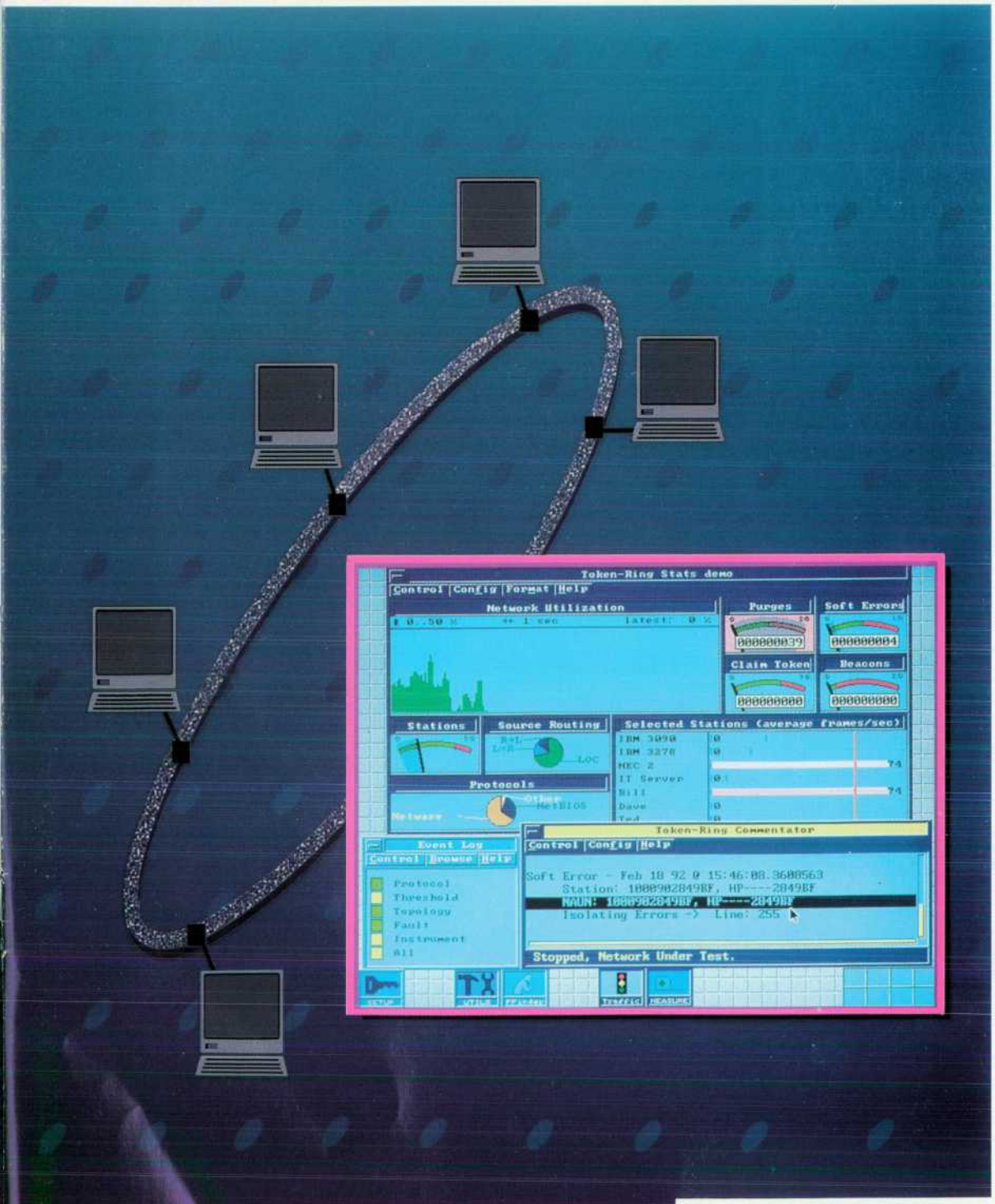
HEWLETT-PACKARD
JOURNAL

October 1992



HEWLETT PACKARD

HEWLETT-PACKARD
JOURNAL

October 1992 Volume 43 • Number 5

Articles

Departments

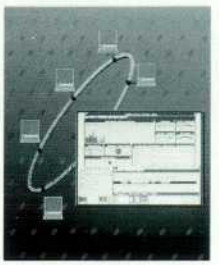
The Hewlett-Packard Journal is published bimonthly by the Hewlett-Packard Company to recognize technical contributions made by Hewlett-Packard (HP) personnel. While the information found in this publication is believed to be accurate, the Hewlett-Packard Company disclaims all warranties of merchantability 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 headquarters office in your country or to the HP address on the back cover. When submitting a change of address, please include your zip or postal code and a copy of your old label. Free subscriptions may not be available in all countries.

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



A protocol analyzer is an instrument for monitoring and interpreting the data at a point in a data communication network, along with the synchronization, error correction, and control information that accompanies the data. The definition of the correct form for all of this information is called a protocol, and many different standard protocols exist. Trouble on a network is often caused by deviations from the correct protocol, which may or may not be caused by hardware failures. A protocol analyzer is supposed to help the network troubleshooter find such problems and restore service quickly. In its approach to this task, the HP 4980 Network Advisor family of personal-computer-based protocol analyzers automates much of the work of interpretation and fault analysis that traditional analyzers have left to the troubleshooter, offering both protocol commentary and expert system support for the first time. Along with the values of various fields and frames, the Network Analyzer tells the user when unexpected values or frame sequences occur. A built-in artificial intelligence application called the Fault Finder automates the troubleshooting process, using the same rules as expert troubleshooters to investigate likely causes and zero in on the problem. The article on page 6 introduces the Network Advisor, and the expert Fault Finder system is described in the article on page 11. The software architecture of the Network Advisor divides tasks between two environments: the general-purpose environment (page 21), which implements the user interface, and the analysis and real-time environment (page 29), which acquires data and processes it in real time. The Network Advisor recognizes most major protocols; it can identify the protocol being monitored and does not need to know it in advance. Recognition and interpretation of the syntax and semantics of the various network protocols are the tasks of the Network Advisor's decodes facility (page 34). Here the Network Advisor differs from traditional protocol analyzers both in the number of protocols it can handle and in its ability to provide not just data but answers to protocol problems.

The protocol analyzer isn't the only approach to maintaining the health of a digital network. Depending on the type, size, and importance of a network, distributed monitoring may be appropriate. The HP Model E3560 digital performance monitoring and remote test system is designed for continuous surveillance of digital telecommunications networks according to Recommendation G.821 of the International Telephone and Telegraph Consultative Committee (CCITT). The system provides network managers with statistics that reflect the quality of network service and collects alarms that signal failures in network elements. It scans data streams at the four main data rates in the European CEPT hierarchy (2, 8, 34, and 140 megabits per second), looking for alarms and binary errors. The system's demultiplexing capability can pick out and monitor lower-rate tributary streams within a higher-rate data stream. The design of the HP E3560 system is described in the article on page 89.

In the article on page 48, the designers of the new HP Model 71500A microwave transition analyzer describe it as "a cross between a high-frequency sampling oscilloscope, a dynamic signal analyzer, and a network analyzer." Indeed, its block diagram doesn't contain any elements that aren't found in one or more of those well-known instruments. The contribution of this new microwave instrument is in its architecture, that is, how its components relate to each other, and in its programming. Using periodic sampling, analog-to-digital conversion, and digital signal processing in new ways, it brings time-domain analysis to microwave component engineers who in the past have had to rely primarily on frequency-domain measurements. Time-domain measurements are particularly important in pulsed-RF and nonlinear device testing, and the microwave transition analyzer is optimized for these applications. Its architecture allows it to make magnitude and phase measurements on RF pulses with rise times as fast as 25 picoseconds. The article on page 48 introduces this new analyzer, demonstrates many of its new measurements and applications, and explains the importance of its high sensitivity, synthesized sampling rate, and stationary sampling mode. The design trade-offs and challenges are discussed in the article on page 63.

Electronic spreadsheet applications let people express business problems in the familiar rows and columns of a ledger. With a spreadsheet program, you don't have to know how to program a computer to interact with one to solve business problems. For engineers, the analog of the ledger is the block diagram, and now there's an analog of the spreadsheet program to free engineers from having to translate their block diagrams into unfamiliar computer languages. The HP Visual Engineering Environment, or HP VEE, is a software tool that allows users to create solutions by linking visual objects or icons into block diagrams. The user selects objects from a menu, links them in the way that represents how data flows from one to the other, and then executes the resulting block diagram. The article on page 72 explains what HP VEE does and how it works. As you might expect, a lot of thought went into making its user interface as user-friendly as possible, and that effort is discussed in the article on page 78. Its dataflow architecture, described on page 84, is an object-oriented implementation that strictly separates views of an object from the underlying model.

As faster computers are developed, faster data links are needed to interconnect them. There is already a demand for serial data links capable of gigabit-per-second data rates, 100 times as fast as Ethernet local area networks and ten times as fast as the FDDI fiber optic standard. While gigabit-rate links have been used on long-haul telephone networks for many years, their implementation is too costly and complex for computer use. The HP HDMP-1000 gigabit link chipset (page 103) is the first commercially available, two-chip, 1.4-gigabit-per-second, low-cost, serial data link interface. The G-link chipset consists of a transmitter chip and a receiver chip and requires no external parts or adjustments. The transmitter accepts parallel data and outputs serial data to the link, while the receiver chip reassembles the parallel data on the other end. Using a special encoding algorithm called CIMT (conditional inversion with master transition) and an on-off or "bang-bang" phase-locked loop, the chipset automatically maintains dc balance in the transmitted data and maintains data synchronization. Among its many other possible uses, the G-link chipset has been adopted as the basis for two serial data interface standards.

R.P. Dolan
Editor

Cover

The HP 4980 Network Advisor can be connected to a network like any other node to monitor the health of the network. This rendition depicts a token ring network with several workstations and the Network Advisor connected to it. The Network Advisor is represented by a statistics summary screen, which summarizes activity on the network for a certain period of time.

What's Ahead

Papers on product designs and other topics planned for the December issue are:

- The HP DesignJet plotter, a large-format inkjet drafting plotter
- The HP DraftMaster Plus plotter, a large-format pen plotter that features the SurePlot system for improved drawing reliability
- A multiprocessor HP-UX operating system for HP 9000 computers
- The EISA input/output extension system for HP 9000 Series 700 workstations
- A methodology for migrating application software to a client/server, open systems environment
- Demountable TAB, a new IC packaging technology for modern digital systems
- 1992 Index.

The HP Network Advisor: A Portable Test Tool for Protocol Analysis

This network protocol analysis tool combines expert system technology with a comprehensive set of network statistics and protocol decodes to speed problem resolution for token ring and Ethernet networks.

by Edmund G. Moore

Protocol analysis consists of monitoring and interpreting the data communications protocols on a network. The HP 4980 Series Network Advisor products are protocol analyzers offering support for both LAN (local area networks) and WAN (wide area networks) technologies.

The primary users of a protocol analyzer are people responsible for maintaining communication networks. These users fall into two categories: those responsible for maintaining service within their own company (private network operators) and those who provide service to other companies (network service organizations). Protocol analyzers are used to solve the most difficult network problems. Since these problems account for 20% of all network failures and usually mean degraded network service, the protocol analyzer must:

- Give the user the tools needed to find the problem and restore service quickly
- Be easy to use so that the user does not have to spend time figuring out how to operate the product
- Provide the user with information that is pertinent to solving the problem.

The HP Network Advisor provides features to satisfy these requirements.

Main Features

The Network Advisor is a portable integrated test tool that supports testing of IEEE 802.3 (Ethernet) and IEEE 802.5 (token ring) network configurations (see Fig. 1). The three product configurations for the Network Advisor are given in Table I.

Fig. 1. The HP Network Advisor showing the mainframe and the folding keyboard and display. The display shows an example summary statistics screen in which graphical objects such as bar charts, pie charts, and gauges are used to present information.

Table I
Network Configurations Supported by HP Network Advisor Products

Products	Supported Network Configurations	
	IEEE 802.3	IEEE 802.5
HP 4980A	X	X
HP 4981A	X	
HP 4982A		X

The Network Advisor is made up of two physical components: a PC (personal computer) module and an acquisition module. The PC contains all the user interface functionality (keyboard, display, disks, I/O ports). The acquisition module is a processing system custom designed to meet the data processing requirements of the network under test.

The measurement focus of the Network Advisor is rapid problem isolation—that is, reducing the time needed to restore the network to operation. Traditional protocol analyzers focus on providing users with accurate information. On a LAN, this can mean tens of thousands of frames of data. The challenge for the troubleshooter is to find the frame that is the root cause of a problem. Users of traditional analyzers often spend hours examining pages of data looking for a clue to solve the problem.

The Network Advisor provides the user with not only all of the network frame data, but also abstracted views of that data. These views include: statistics, protocol following,* data filtering, protocol commentary, and expert system support. Some of these tools exist in other products. However, tools like protocol commentary and expert system support are new to the industry.

Traditionally protocol analysis tools have focused on two features: protocol decodes and statistics. Protocol decodes are routines that simply take the protocol header information in the frames and display this information for the user. Statistics give the user information on traffic levels used by the entire network or by individual devices. In the Network Advisor decodes are improved by interpreting the header information. Users are told not only the value of a field, but also the expected value. Additionally, the Network Advisor keeps track of protocol state information, allowing the Network Advisor to tell users when unexpected sequences of frames occur. Both interpretation and state information are new features for a protocol analyzer.

Statistics in the Network Advisor are designed to provide the user with an easy-to-understand summary view of the network. Using the analysis power of the front end and a graphical interface, a great deal of information is displayed in a concise, summary format (see the display in Fig. 1). Data on network use, errors, traffic, protocol distribution, and traffic level on selected frames is combined into one summary display.

Protocol commentary and expert system support are powerful additions to the protocol analysis toolset. A software application called a protocol commentator observes the protocols in network frame data and distills the data into concise events of interest. Expert system support in the Network Advisor is provided by an application called the Fault Finder, which automates the troubleshooting process. Both of these tools are discussed in the article on page 11.

* Protocol following is the process of following the state of a connection. Examples of states include established, resequencing data, and reassembling data. Protocol following is essential for accurate decoding of higher-layer protocols.

Other Features

Besides the new additions to the protocol analysis toolset described above, the Network Advisor provides some traditional protocol analysis features in new ways.

Mechanical and Ergonomic Features. Users of protocol analysis tools such as the Network Advisor frequently need to take the tool to the problem. For this reason portability and ruggedness are key features of the Network Advisor (see Fig. 1). The folding keyboard and display are well-protected. The instrument has a modular acquisition subsystem and the assemblies are connected with a single connector. Four quarter-turn fasteners provide the mechanical connection, making it easy to change from one network technology to another in a few seconds.

The Network Advisor is the first HP product to incorporate active matrix color LCD technology. Color LCD provides the user with reduced size and weight while providing a bit-mapped color graphics interface. The color LCD was added very late in the project at considerable risk to product introduction. The team did it without slipping the product introduction schedule. Response to the package concept and especially to the color LCD has been very positive.

The article on page 41 describes the mechanical design of the Network Advisor.

Use of MS-DOS.® The LAN testing market uses products based on the MS-DOS operating system. The DOS requirement imposed a performance problem for the design team because many of the existing DOS-based products could not match the data integrity goals we had set. The Network Advisor design team solved this problem by creating a machine that has two independent environments: data acquisition and DOS. The data acquisition module is custom-built to ensure data integrity under any user network condition. The DOS module is fully DOS-compatible and provides an excellent user interface. The two modules interface using dual-port memory, which is mapped into the DOS memory space just like commercial PC I/O cards.

We gained substantial benefits from this dual-module approach. Since our DOS hardware was heavily leveraged, we needed only one full-time engineer for both hardware and BIOS development. The architectural split allows us to mix and match different data acquisition and DOS engines to create multiple price/performance products easily. Finally, being fully DOS-compatible allows us to leverage the vast amount of commercial software available, particularly communications software.

Data Capture and Run-Time Analysis. Ensuring that all data from the network can be captured, even under worst-case loading, is a difficult design task. In addition to data capture, the front end (data acquisition module) has substantial data processing requirements. In a general sense, the Network

Advisor is looking at the sum of all network traffic. Analyzing all that traffic would require a processing bandwidth that matches the total processing resources of all networked elements. Therefore, the worst-case processing load imposed on the analyzer will always outstrip the ability of the instrument to process it. Instead of trying to process all network traffic, the Network Advisor focuses on processing a subset of all the network traffic it sees.

As a test tool, the Network Advisor needs to see everything, but not to process everything. Commercial networking chips are designed to be used by network nodes and they automatically reject input that is outside the specification for the network. A test tool must see this out-of-specification data to give the user a complete picture. An HP-developed receive engine ensures that the Network Advisor provides the user with all the data associated with an error frame.

User Interface. MS-DOS-based network test products traditionally provide a single task model user interface. This means that the product allows only one task at a time to execute. The Network Advisor provides a multitasking, bitmapped user interface that is mouse- and keyboard-driven. Users can have multiple tasks executing simultaneously using a windowing environment. Users can start the traffic generator (to simulate a problem on the network) while running network statistics (to observe what effect the traffic generator has) and the network commentator (to be informed of any problems that result). All these tasks can run simultaneously to help solve a network problem.

Product Architecture

As mentioned earlier, the Network Advisor is divided into two major environments: a general-purpose environment, which is essentially a personal computer, and a data acquisition, analysis, and real-time (ART) environment. This division is applied to the hardware and software architectures of the Network Advisor.

Hardware Architecture. The general-purpose portion of the environment contains a 20-MHz Intel386SX microprocessor with 8M bytes of RAM. The BIOS and support chip set are from Chips and Technology Company and the disks are the same ones qualified for use in the HP Vectra PC. The internal display is VGA LCD, either grayscale or color. Our objective was to create a PC-compatible machine. Since the general-purpose environment is a PC, our engineering investment was relatively small. Leverage of design and components occurred whenever possible.

The analysis and real-time environment is based on the AMD29000 RISC chip. RISC technology was selected because of price/performance concerns. We needed a CPU that would have enough bandwidth to perform our run-time analysis and we needed to provide enough bus bandwidth to do DMA transfers of all the frames into main memory during run time. The analysis and real-time environment uses 4M to 16M bytes of memory for program execution and data storage for captured frame data. Program and data space uses 2M to 3M bytes of memory. Data acquisition is based on a programmable front end that uses Xilinx programmable gate arrays. The front end provides framing, pattern recognition, run-time pattern matching, run-time frame filtering, statistical counters (e.g., frames per second, errors per second, etc.), DMA control, and basic node card functionality (to transmit frames and participate in the network protocol as needed). The front end is designed to ensure data integrity in the capture buffer under any network condition, valid or invalid. The ability to ensure data integrity is an important feature of the Network Advisor.

Software Architecture. The general-purpose environment was developed using object-oriented programming. Smalltalk is the language used. Smalltalk provides multitasking, memory management, object-oriented design support, and support for all DOS functions (primarily I/O control). The Network Advisor's user interface was written in Smalltalk to imitate the OSF/Motif user interface.[1]

The software was developed on HP Vectra PCs, making the port to the target hardware quite simple. Since the software team was large, a toolset that allowed multiple users to share the code "image" over a network was employed. This multiuser tool provided us with a networked development environment.

The analysis and real-time software was also developed using object-oriented programming technology, except that C++ was the language used. Software was developed on the HP-UX* operating system and cross-compiled onto the AMD29000. Software development for these modules was also a team effort with the code image residing on a networked HP-UX server. The core of the analysis and real-time code was leveraged from another HP project called CONE (common OSI network environment).[2] CONE is the protocol kernel used in HP workstations for managing networks. The capabilities and design of CONE matched the basic tools we needed for protocol analysis.

The analysis and real-time software is described on page 29. The software developed to run on the PC and the general-purpose environment are described on page 22.

Software Management

Except for the DOS BIOS and the analysis and real-time boot and self-test, which are ROM-based, all the software in the Network Advisor is disk-based. Having a DOS-based software system has proven to be a major benefit to product enhancement and to our customers. During the first year of the Network Advisor's life, we created three major upgrades, several bug fixes, and a new leveraged product. Support for all of these changes was based on shipping new disks.

A DOS-based system does pose some problems. First, there is a tendency for users to modify the system configuration to add applications, drivers, and TSRs (terminate and stay resident programs). Different DOS applications do not always peacefully coexist. In addition to software, the ability to have different disk drives, different amounts of memory, and different CPUs creates a matrix of configurations that can be overwhelming.

Even if the configuration issue is managed, being DOS-compatible means evolving over time. DOS 5.0 has become the standard since our release, and in March 1992 we changed our shipped configuration to be DOS 5.0 with a single disk partition. We still must support our customers who still have Network Advisors with DOS 3.3 and multiple disk partitions. This creates its own logistical problem.

(continued on page 10)

Network Advisor Product Enhancement Philosophy

Our intent was to release an initial Network Advisor product with a credible set of features as quickly as possible. The initial feature set did not provide all of the capabilities we wanted. It did provide enough capability to solve customer problems. We wanted to provide releases of additional software on a frequent basis (two or three times a year). The initial release occurred in July 1991 and since then we have had new releases in November 1991, March 1992, July 1992, and August 1992. Each release added additional capabilities in all areas of the product including:

Fault Finder. The Network Advisor can do automatic troubleshooting via the Fault Finder on the following network media:

IEEE 802.3 Media Access Control (MAC) Hardware
IEEE 802.5 MAC Hardware.

Commentators. Commentators provide a high-level abstraction of protocol activity. Unlike a protocol decode, which displays all of the fields associated with a particular protocol, the commentator reports on the meaning of a frame in the context of the service being provided by the protocol. The Network Advisor provides the following commentators:

Ethernet Novell Commentator
Ethernet ICMP Commentator
Token Ring Commentator
Token Ring Novell Commentator
Token Ring IBM LAN Manager Commentator

Decodes. The Network Advisor decodes most industry-standard protocols. A decode enables protocol experts to examine the contents of a protocol data frame in detail. The Network Advisor provides the following decodes:

Ethernet/LLC 802.3
IEEE 802.2 SNAP ELAP Token Ring
IEEE 802.2 Token Ring MAC SNAP
Appletalk Address Resolution Protocol
Datagram Delivery Protocol
EtherTalk Link Access Protocol
AppleTalk Transaction Protocol
Routing Table Maintenance Protocol
AppleTalk Echo Protocol
AppleTalk Name Binding Protocol
Banyan Vines Internet Protocol
Vines Address Resolution Protocol
Vines Routing Update Protocol
Vines Sequenced Packet Protocol
Vines Internet Control Protocol
Vines Interprocess Communication Protocol
DECnet Data Access Protocol
Session Control Protocol
Data Access Protocol
Network Services Protocol
DECnet Routing Protocol
Local Area Transport Protocol
Novell Netware Code Protocol
Sequenced Packet eXchange Protocol
Internet Packet eXchange Protocol
IBM PC LAN NetBIOS Protocol
Server Message Block Protocol
TCP/IP Telnet Transport Control Protocol
User Datagram Protocol Internet Protocol
File Transfer Protocol
Internet Control Protocol
Address Resolution Protocol
OSI CLNP
OSI ES-IS
OSI TP4
SNA
NetBIOS
XNS Internetwork Datagram Protocol
Sequence Packet Protocol
3COM NetBios Protocol
Server Message Block Protocol

Statistics. The Network Advisor's statistical measurements give the user a graphical view of critical network performance parameters and network users:

Ethernet Summary Statistics
Ethernet Node Statistics
Ethernet Top Talkers
Ethernet Top Error Sources
Ethernet Vital Signs
Token Ring Summary Statistics
Token Ring Station Statistics
Token Ring Top Talkers
Token Ring Top Error Sources

Canned Tests. Canned tests provide a set of powerful troubleshooting tools for performing tasks such as testing for connectivity and finding active stations. Many of the canned tests stimulate the network to simulate network devices. The Network Advisor provides the following canned tests:

Ethernet Transceiver Test
Token Ring List Configuration
Report Servers Token Ring List
LAN Managers Token Ring List
NETBIOS Stations Token Ring List
Novell Stations Token Ring List
Ring Error Monitors Token Ring List
Ring Parameter Servers Token Ring List
All Bridges Token Ring List
All Stations Token Ring List
Calculate Ring Length
Token Ring Lobe Test
Token Ring Request Station ID
Token Ring Station Adapter Status
Token Ring Active Station List
Novell Find Nearest Server
Novell Get List of Servers
Novell View Nodes
Novell Server Ping
Novell Node Ping
Novell Determine Connected Networks
TCP/IP ARP Request
TCP/IP Ping

Node Discovery. The discovery measurement identifies active nodes on the network by observing network traffic. The measurement can find and display MAC (media access control) and network addresses. The binding of MAC and network addresses clearly shows the activity of routers in the network. Discovered nodes can be merged into the system nodelist.

WAN Capability. A data acquisition module that contains the HP 4957 WAN analyzer functionality has been implemented as a PC I/O card. Using the Network Advisor's DOS capability, this card gives customers WAN support, without any additional software.

FDDI Capability. A data acquisition module is available that interfaces to FDDI (fiber distributed data interface) networks. This module was implemented in the same spirit as the IEEE 802.3 and 802.5 modules—ensure data integrity under any network condition.

Disk Drives. During development, we had planned to use 40-Mbyte and 80-Mbyte hard disks. However, by the time we introduced the product, typical disk densities for PC hard disks had changed (50 Mbytes to 105 Mbytes were typical sizes). We switched to 80-Mbyte and 160-Mbyte disk drives during the first year.

Intel486 CPU. We created an Intel486 version of the CPU to keep current with CPU technology. The Network Advisor has been designed to allow adaptation to new CPU and I/O technologies.

Conclusion

The Network Advisor has created a new standard in the LAN test marketplace. It is a tool that not only collects and supplies users with network traffic data, but also extracts pertinent answers from the volumes of data. As a DOS-based system, the Network Advisor offers compatibility, flexibility, and a clear path for evolution. As an instrument, the Network Advisor offers unprecedented performance in data capture and analysis and brings HP quality and data integrity to the LAN marketplace.

References

1. A. Deininger and C. Fernandez, "Making Computer Behavior Consistent: The HP OSF/Motif Graphical User Interface," *Hewlett-Packard Journal*, Vol. 41, no. 3, June 1990, pp. 6-12.

2. S. Dean, D. Kumpf, and H. Wenzel, "CONE: A Software Environment for Network Protocols," *Hewlett-Packard Journal*, Vol. 41, no. 1, February 1990, pp. 18-28.

Embedding Artificial Intelligence in a LAN Test Instrument

The knowledge and processes used by a skilled LAN troubleshooter are built into an interactive expert system application that runs on HP 4980 Series Network Advisor protocol analyzers.

by Scott Godlew, Rod Unverrich, and Stephen Witt

The capabilities of artificial intelligence techniques are provided in the HP 4980 Series Network Advisor protocol analyzers by a software application called the *Fault Finder*. It is a rule-based expert system that is built around a blackboard architecture.[1,2] The rules, written in PROLOG,[3] invoke Network Advisor measurements (statistics, decodes, and applications) that are available to the user. The Fault Finder allows the user to control and view the troubleshooting process at a detailed single-step level or at a fully automated level. It also includes an explanation facility that describes the logic used to solve a specific problem, a definition of the problem, and a description of the actions required to remedy the problem.

This article will discuss LAN troubleshooting, automated troubleshooting using expert systems, the Fault Finder, the architecture of the Fault Finder, and a typical problem solved using the Fault Finder on a token ring network.

Expert troubleshooters use a paradigm of making observations, forming hypotheses, and proving or disproving those hypotheses until a problem is found (see Fig. 1). The Fault Finder uses this same paradigm. A description of how this model is employed by the Fault Finder is mentioned throughout this paper.

LAN Troubleshooting

For the last ten years LAN (local area network) troubleshooters have relied on LAN protocol analyzers that provide a variety of decode measurements, statistical measurements, and active measurements. This is a manual process that depends on the user's knowledge of the instrument, the network, and typical problems that occur on that network. These analyzers require a user to interpret results and select subsequent measurements.

LANs provide high-speed packet switching within buildings or campus facilities. They include CSMA/CD baseband networks, token ring networks, and broadband networks. This paper addresses Ethernet (IEEE 802.3) and token ring (IEEE 802.5) networks. LANs are challenging in their troubleshooting requirements because they operate at high speed, problems emerge and escalate in real-time, and the environment is very complex. Problems can result from poorly architected networks, improper device configurations, faulty cabling or connections, broken devices or printed circuit cards, or incompatible software. A typical LAN can have several network operating systems and protocol stacks. Troubleshooting a network problem requires integrating pieces of data or clues from a variety of sources and using the acquired data to hypothesize and prove problems. Problems that do not cause hard failures but instead only cause performance problems may often go undetected.

Network diagnostic tools have evolved in the same way as networks. Early diagnostic tools included workstation utilities, cable measurement instruments, and simple protocol analyzers that provided decoding of protocol packets. Network troubleshooting requires sequencing through different measurements, using the results of one measurement to select and program the next measurement. As networks became more complex, the problems became more complex. This increasing complexity created a need for more sophisticated tools to enable network managers to solve problems rather than relying on hit or miss solutions, or in some cases simply living with the problem.

Network managers and technicians solve many problems by relying on knowledge of their specific network and its components and by relying on their troubleshooting experiences. For example, a network technician can quickly identify a misconfigured node card by observing the card's receiver congested soft-error frames on the token ring network.

Expert troubleshooters use a mental model or paradigm for troubleshooting. Some perform this at a decidedly conscious level by diagramming the troubleshooting process. Others perform it subconsciously, following "what feels right." In either case the same basic process is used. They start by making observations of the situation. It doesn't matter what problem is being solved, be it a ruptured appendix, a faulty carburetor, or a duplicate IP address—all troubleshooters (doctors, auto mechanics, and network managers) use the same process. They use these observations to formulate

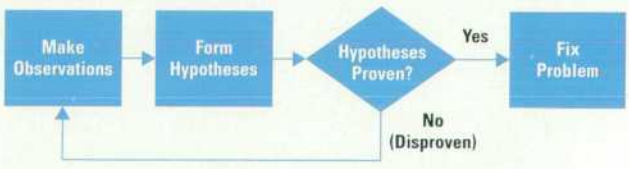


Fig. 1. The observe, hypothesize, and prove (or disprove) troubleshooting process used in the Fault Finder.

hypotheses of what problems might exist. Then they perform tests to prove or disprove the hypotheses. Finally, once a problem is proved they remedy the situation. This model is shown in Fig. 1.

Protocol analyzers offer a variety of measurements for solving problems. Decodes provide descriptions of individual packets on the networks. Statistical measurements provide overviews of network trends such as utilization, errors, and protocol use. Individual applications provide utilities for a variety of functions such as creating an active station list, reading the status of network adapter cards, and testing the media. These are powerful tools in the hands of an expert. However, these analyzers have two major shortfalls. First, to solve difficult problems an expert user is usually required. Second, they require human intervention and cannot complete the troubleshooting task in an automated fashion. Artificial intelligence offers a desirable solution to both of these problems. It allows an analyzer to monitor the network continually for problems and log the results for later perusal by a network manager. It also provides the means to build the knowledge of many troubleshooting experts into a tool that is widely available to network managers.

Automated Troubleshooting Using Expert Systems

Artificial intelligence (AI) solutions that are declarative in format and conventional solutions that are procedural in format can be used together to solve networking problems.* Expert systems (one branch of AI), in a broad sense, are programs that are designed to behave as a human expert in a particular field. Expert systems are particularly useful for problems like networking in which complete information about a problem is not known when the program begins execution. Expert systems gather additional, pertinent information as they execute. Conventional, procedural programs usually execute in a sequential fashion through a set of troubleshooting trees and can take more time and execute incorrect branches. Expert systems gather information after an event and use it to explore multiple problems in parallel.

The requirements for an expert system troubleshooting tool are somewhat diverse. First and foremost the expert system must discover network faults and make observations about the network. A primary goal is to diagnose common network problems quickly, allowing the human user to concentrate on more difficult and obscure problems. To do this, the expert system must be cognizant of the network structure, the protocol environments, the diagnostic tools available, and the troubleshooting methods that will solve the problem quickly. Thus, an expert system tool for network troubleshooting must be able to do the following:

- Measurement Interface:
  - Perform diagnostic functions such as generating station lists, testing connectivity, and performing loopback tests
  - Confirm the existence of a hypothesized fault by executing active measurements such as token ring station adapter status
- Automated Operation:
  - Monitor the network for real-time problems as opposed to gathering information and postprocessing it in a batch fashion
  - Execute in an automatic, unattended fashion to solve intermittent faults, monitoring suspect rings or segments continuously
- Ease of Use:
  - Provide the user with an interpretation of data by suggesting actions, drawing conclusions, and explaining advice
  - Provide an audit trail of suspected problems, measurements executed, and problems found to educate the user and to suggest possible problems that the expert system could not solve
  - Generate alarms and log data to notify the user of proven faults and provide the necessary information to prove that the problem exists
  - Incorporate user inputs by including information that the user already suspects about the network (such as performance problems)
- Topology:
  - Gather and incorporate network topology information
  - Gather error information that is being reported on the network and learn about the configuration of the network
  - Gather and incorporate network baselines to determine what is normal behavior on the network and compare the current operation against normal behavior.

It is critical that an expert system that augments a user's troubleshooting methods behave in a manner that the user can understand. Making observations, forming hypotheses, proving faults, and determining actions to take are critical troubleshooting steps that a network manager can understand and relate to.

Note that expert system tools could also be used to optimize performance, analyze network accounting information, perform network management functions, audit for security violations, and provide information for network planning. However, in this article we are only concerned with an expert system tool whose purpose is to diagnose operational faults on a local ring or segment of the network.

The Fault Finder

The Fault Finder is an expert system that executes as a software application on the Network Advisor family of products. It uses troubleshooting methods that are modeled after expert users in the field, applying knowledge of known network problems. It programs and executes measurements in the same way that an expert user would, taking advantage of the powerful measurement set of the Network Advisor.

The Fault Finder was designed to provide the user with a high degree of interaction with the instrument and a detailed view of the activities of the Fault Finder as it attempts to solve a problem. This was considered critical because many network problems cannot be diagnosed to completion. In these cases it is important to give the user an audit trail and provide as much information as possible that might be pertinent to solving the problem. For example, suppose the Fault Finder suspects a broken transmit wire on a network interface card with address 10005A74624A, but this suspicion proves to be false. The fact that the Fault Finder was investigating a potential problem on a certain station on the network might be of interest to the network technician. The technician might be able to correlate the Fault Finder data with previous troubleshooting data and use this synthesis of

* In the context of computer programs, procedures tell a system specifically how to do a task and declarations tell a system generally what to do.

information to solve the problem. In addition to suspected problems, the Fault Finder records measurements it has executed. Such information may help an expert hone in on the real problem. This audit trail can also be used as an educational tool by the novice troubleshooter.

The Fault Finder's main screen has three tiles that map directly to the observations, hypotheses, and proven faults, which are the mainstays of the expert's troubleshooting process (see Fig. 2). The Fault Finder starts by running measurements that provide observations for the troubleshooting process. These observations are posted to the first window. The rules in the knowledge base are executed and hypotheses are formed. For example, if a statistical measurement is run that observes that the rate of broadcast frames on an Ethernet network exceeds a baseline, the Fault Finder will hypothesize several possible problems including a duplicate IP address and a misconfigured IP broadcast address. The Fault Finder will then perform further measurements such as a ping (internet control message protocol (ICMP) echo request) or an IP (internet protocol) decode to determine whether a problem exists. If the problem exists, it is posted to the Faults Found window, the window is turned red, and the user is notified with an audible alarm.

The user can run the Fault Finder in several different modes. The single-loop mode runs once through the possible fault indicators looking for problems. It follows all results to a conclusion and then stops. The Fault Finder can run in a continuous-loop mode in which it repeatedly cycles looking for faults. The Fault Finder will also accept user symptoms to allow the user to direct the search by including what is already known or suspected about a network fault. Possible symptoms include poor performance, cannot connect, and suspected Novell problems. These symptoms cause the Fault Finder to focus its search initially on suspected problem areas. When appropriate, the Fault Finder will request the user to input the results of cable scanning measurements to aid in diagnosing physical media problems.

The Fault Finder accesses Network Advisor measurements in the same way that the user would. For manual use the user is presented with a window containing a list of all the Network Advisor measurements sorted by categories. The Network Advisor allows the user to select multiple measurements and execute them simultaneously. Each measurement includes a parameterization window for setting up the configuration. For example, a ping measurement requires the user to specify the internet addresses for the Network Advisor and the target node for the ping, and the timeout value (see Fig. 3). The Fault Finder automatically selects measurements, provides parameterization, executes the measurements, and obtains the results. A user performing this task manually can often make a mistake, which can lead to a false diagnosis and many hours of invalid and frustrating troubleshooting.

Network troubleshooting depends on the concept of understanding normal behavior on a network and comparing the observed results with the normal expected behavior. The Fault Finder uses this same approach by keeping a baseline file that documents the expected values for the measurements to compare with the actual results. Each measurement compares its actual results against the expected results in the baseline file.

The Fault Finder uses prioritization and certainties to guide itself through the troubleshooting process. Prioritizations are implemented by assigning a severity and a frequency to each problem. This means that the Fault Finder will pursue more serious problems first. For example, a broken file server is a more serious problem than a broken node and will be investigated first. Certainties refer to the confidence levels assigned to the Fault Finder's results. Each Fault Finder result

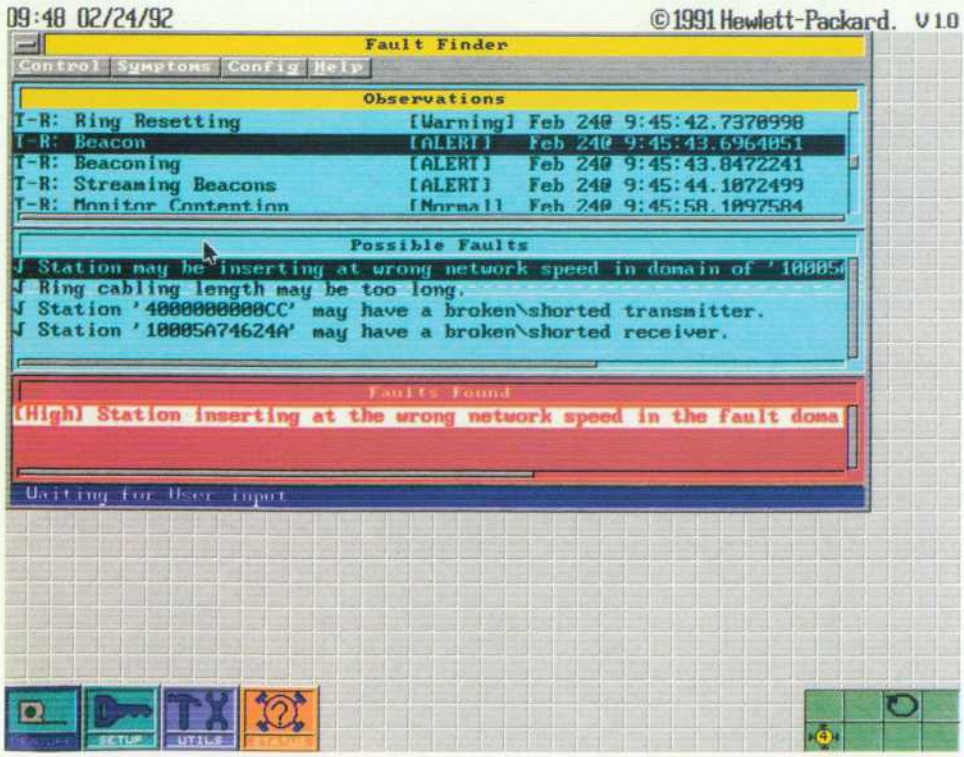


Fig. 2. Fault Finder windows. Each window represents a stage in the troubleshooting process.

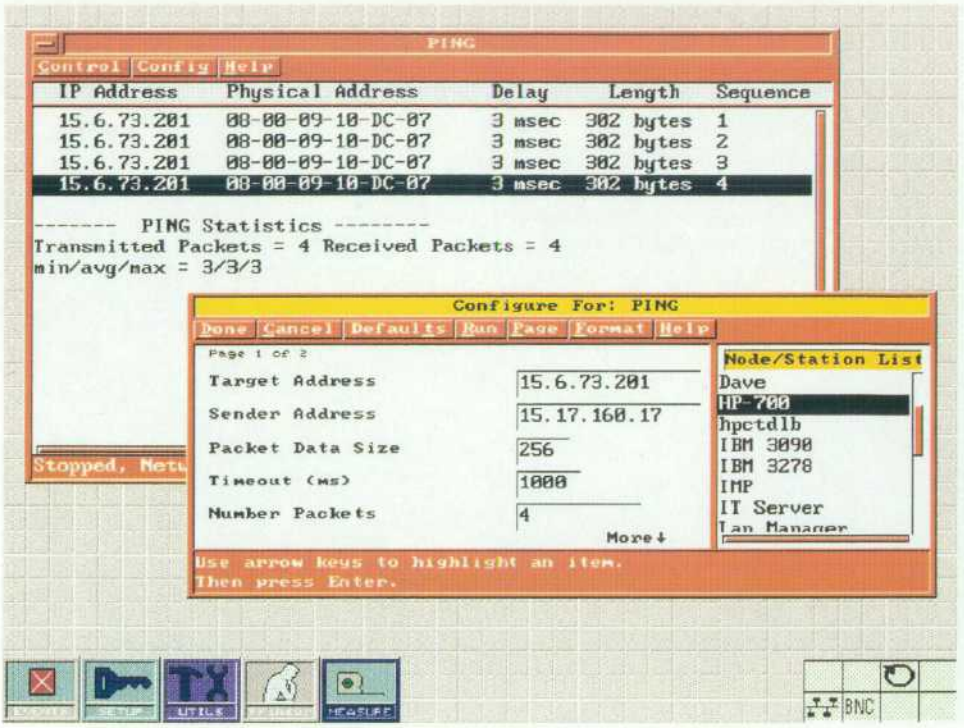


Fig. 3. Ping measurement window.

is rated as low, medium, or high (see the [High] designation given to the fault found in Fig. 2). Problems (faults) that can be conclusively confirmed are given a high confidence and problems that might be one of several possibilities and cannot be diagnosed further are assigned a low confidence.

The Fault Finder provides the user with a very fine level of control over the troubleshooting process. Allowing the user to interact with the Fault Finder was a key design objective. Expert systems that appear as a black box to the user are not appropriate for interactive network troubleshooting. The Fault Finder normally runs in an automatic mode cycling from observations to hypotheses to proven faults. This is very useful for verifying the normal operation of the network, or for a mode of debugging in which the user might look for intermittent problems on a network before they become catastrophic and degrade the network. The Fault Finder also provides several manual modes of debugging, which are useful for reactive troubleshooting to investigate specific failures.

Once the Fault Finder has executed and discovered a fault on the network, the user must perform the final step of troubleshooting—correcting the fault. Simply identifying the fault is not enough if the user does not know how to remedy the problem. The Fault Finder includes a comprehensive explanation facility that explains the troubleshooting process and describes the actions to be taken to fix the problem. Any line entry in any of the three tiles can be highlighted and selected to invoke the explanation facility. The explanation facility is implemented via the knowledge base. Each entry includes a definition, a reason, and an action. Fig. 4 shows an explanation window that explains the necessary action to fix the problem of a station inserting in the network at the wrong speed.

Fault Finder Architecture

The Fault Finder architecture (see Fig. 5) was designed with four main objectives. First, the Fault Finder must be able to operate the instrument in place of the user (the network manager). This means that the Fault Finder needs to initiate and receive results automatically from instrument measurements in a knowledgeable way. Second, the Fault Finder must actively detect and investigate faults in a manner that will allow users to accept its conclusions and understand its actions. Third, the Fault Finder must be able to support knowledge about multiple protocol domains and adapt to varying target networks depending on the needs of our customers. Finally, an inability to complete the diagnosis of one potential fault should not prevent the diagnosis of other potential faults.

The inference engine provides the core knowledge processing capability and sets the stage for relating the other components in terms of their ability to provide information to support inferencing. The blackboard provides a mechanism for allowing multiple sources and sinks* of information to cooperate and allows the user to get information about how data is synthesized in the Fault Finder. Measurements and user input provide two examples of information source modules that are not knowledge-based, but procedure-based.

Inference Engine. An inference engine executes knowledge about a particular domain. During execution, high-level information is synthesized from measurement data, user input, and the knowledge base. The synthesis can be driven by the need to use the high-level data for some other purpose or the availability of sufficient lower-level data to complete the

* Sources generate information and sinks receive information.

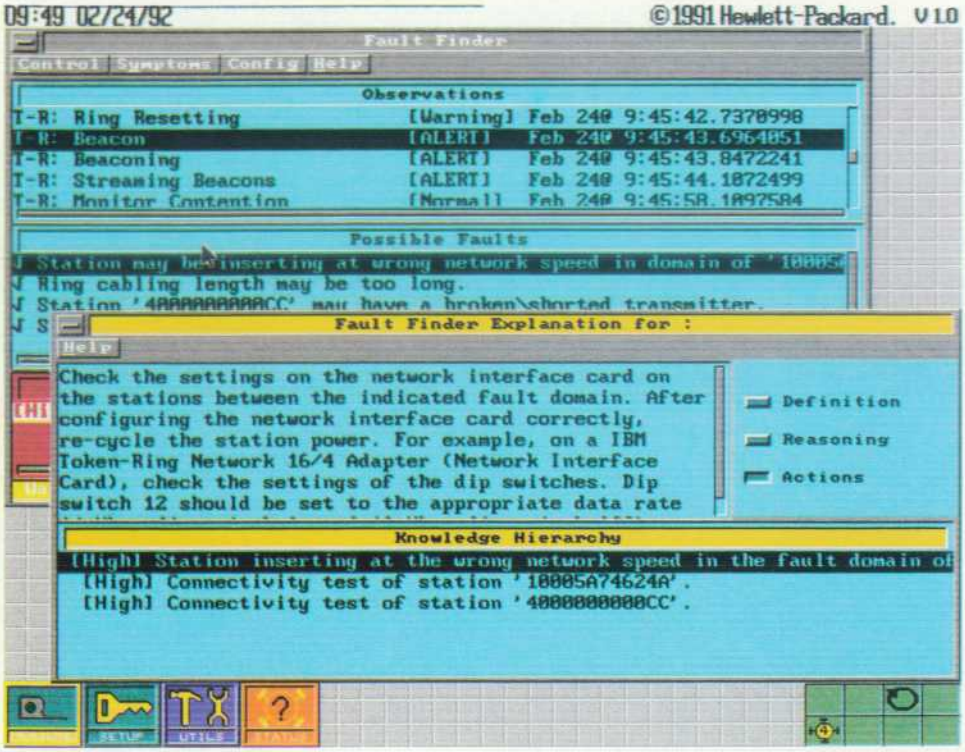


Fig. 4. Fault explanation window.

synthesis. In either case, the knowledge that is executed describes how the synthesis takes place.

The Fault Finder uses a form of knowledge representation known as rules. These rules describe the necessary state of information to be able to synthesize higher-level information. The rules in the Fault Finder have the following three main parts:

- The consequent (describes the information to be synthesized by the rule)
- The antecedent (describes the required preconditions that will allow synthesis to occur)
- The parameters (constrain how the inference engine is allowed to use the rule).

The following rules are used to identify a station inserting in the network at the wrong speed. These rules show the consequent, the problem definition, and the antecedent, which includes the preconditions (forward chaining) that must be satisfied and the information needed to prove (or disprove) the hypothesized problem (backward chaining).

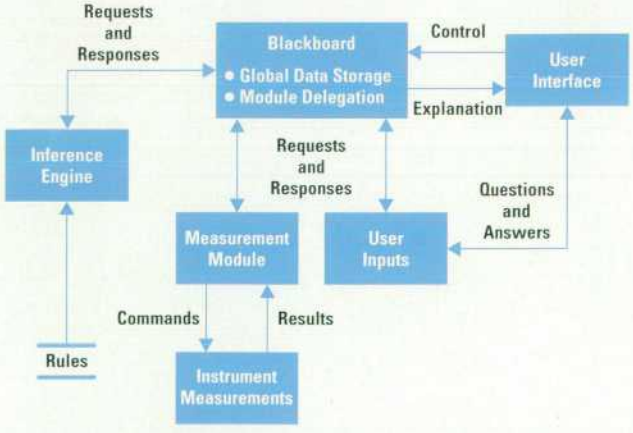


Fig. 5. The Fault Finder's architecture.

```
;Problem description

problem(
  name( InsertWrongSpeedProblem )
  nlsName( 'Inserting Wrong Speed' )
  eventType( #FaultEvent )
  frequency( 50 )
  severity( 50 )
  definition( 'Station inserting at the wrong speed means the network
       interface card (NIC) is not configured to the proper data
       rate for the attached local ring.' )
  solution( 'Check the settings on the network interface card on the
       stations between the indicated fault domain. After configuring
       the network interface card correctly, re-cycle the station power.
       For example, on an IBM Token Ring Network 16/4 Adapter
       (Network Interface Card), check the settings of the dip switches.
       Dip switch 12 should be set to the appropriate data rate (4 Mbps
       dip switch On and 16 Mbps dip switch Off). ' )
  hypoText( 'Station may be inserting at wrong network speed in domain
       of %?%address% and %?%addressNAUN%. ' )
  concText( '%+%Station inserting at the wrong network speed in the fault
       domain of %%?+%addressNAUN%%+% and %%?+%address
       %%+%.%%–%Station
       not inserting at the wrong speed in the fault domain of
       %%?–%addressNAUN%%–% and %%?–%address%%–%.% ' )
  parameters(
    [ address #node #hypothesis " " ]
    [ addressNAUN #node #hypothesis " " ]
  )
)

;Forward Chaining Rule

hypothesize(

;These are parameters
  name( hWrongSpeed )
  cost( 50 )
```

```
  confidence( 100 )
  explanation( 'If a station fails to insert onto the local ring properly,
       then it is possible the station's network interface
       card (NIC) is set to the wrong speed.' )

;Consequent
  logicText( 'InsertWrongSpeedProblem( ?address ?addressNAUN ) :-

;Antecedent
    beaconingMonitor( useBaselines ?time ?beaconAddress
       ?beaconAddressNAUN)
    perform( ?tmp1 ?st "at:put:" #lancTokenRingLastBeaconingAddress
       ?beaconAddress)
    perform( ?tmp2 ?st "at:put:" #lancTokenRingLastBeaconingAddress
       NAUN?beaconAddressNAUN)

    streamingBeaconsMonitor( useBaselines ?time ?beaconStreamAddress
                  ?beaconStreamAddressNAUN ?)

    newActiveMonitor( useBaselines ?time ?addressActiveMonitor
            ?addressNAUNaddressActiveMonitor ?)

    topology (asTokenRingNode [ ?addressStr ] ?address )
    topology (asTokenRingNode [ ?addressNAUNStr ] ?addressNAUN )

    stopAllMeas()
' )
)

;Backward Chaining Rule

backward(

;Parameters
  name( cWrongSpeed )
  cost( 50 )
  confidence( 100 )
  explanation( 'If a station fails to insert onto the local ring properly,
        then it is possible the station's network interface
        card (NIC) is set to the wrong speed. The inserting
        station will attempt the insertion, but will be unable
        to synchronize with the incoming signal and therefore
        remove itself from the local ring. The station may
        try multiple insertions before removing completely.' )

;Consequent
  logicText( 'InsertWrongSpeedProblem( ?address ?addressNAUN ) :-

;Antecedent
    mdbParm (txMeasTimeout ?txTimeout )

    perform ( ?addressStr ?address tokenRingAddress )

    adapterMeas( ?addressStr ?txTimeout ?result1 )
    size( ?result1 ?resultSize1 )
    gt( ?resultSize1 0 )

    perform ( ?addressNAUNStr ?addressNAUN tokenRingAddress )

    adapterMeas( ?addressNAUNStr ?txTimeout ?result2 )
    size( ?result2 ?resultSize2 )
    gt( ?resultSize2 0 )

' )
)
```

The Fault Finder rules are modeled after PROLOG, so the consequent is simply a predicate that represents the goal or information to be synthesized. The predicate has a name that represents the type of information being synthesized and parameters that determine the specific information. For example, in the rule cWrongSpeed the predicate InsertWrongSpeedProblem tells us if the wrong speed is set on the adapter card at some address. This would be synthesized by gathering data via the antecedent predicates and incorporating this data into the consequent.

In the antecedents of the example rules, the conditions are simply ANDed together. Like IF statements in most programming languages, other logical operations can be performed. The inference engine allows patterns to be specified in place of constants, and the condition may succeed multiple times depending on how many pieces of information match the patterns. This allows knowledge to be represented in a general way, without knowing ahead of time how many situations might meet the criteria or the specific names or values of parameters.

The execution of a rule can be driven by forward or backward chaining operations. In forward chaining, the inference engine is presented with one or more network conditions (e.g., network is sluggish). This data will drive the execution of rules that depend on this data. In backward chaining, we start with a result or conclusion to be proved true or false (e.g., station inserting at the wrong speed) and work back through the rules (gathering information along the way) to find the problem or condition causing the given result.

Blackboard. The blackboard allows multiple modules (sources and sinks of data) to work together. It also maintains an information structure that allows greater accessibility and storage of history data about how the information is synthesized or generated.

The blackboard serves as a clearing house for all information in the Fault Finder. It determines which module should be called to perform further information synthesis or data generation. When a module needs information to complete its synthesis, the module requests the information via the blackboard, and the blackboard determines which module can act as a source for that information. When data becomes available asynchronously, the data is distributed to those modules that could perform further synthesis based on the data. The modules are responsible for notifying the blackboard of their specific needs.

Requests and Responses. In the Fault Finder's blackboard, requests for information are posted to initiate information synthesis. For example, if we want the Fault Finder to determine if a particular fault exists, a module will request information about the fault's existence and then direct the Fault Finder to prove (or disprove) the fault. When the Fault Finder is observing the network, it may, on its own, decide to determine if a fault exists. The conditions that indicate the possible existence of a fault cause a request to be placed on the blackboard.

Responses are the result of investigating a request. When a module completes processing a request, one or more responses are placed on the blackboard. Multiple matches of a pattern can generate more than one response. Each response has a level of certainty associated with it, which is determined from the certainty of the information it is based on and the confidence of the rule used to perform the synthesis. In the end, a fault diagnosis can be weighed against other faults to determine a priority for correcting the faults.

Hierarchical Data Abstraction. By tracking requests and responses exchanged via the blackboard, a hierarchy of information created by the system can be maintained (see Fig. 6). The hierarchical orientation of the information facilitates both usability and programmability of the system.

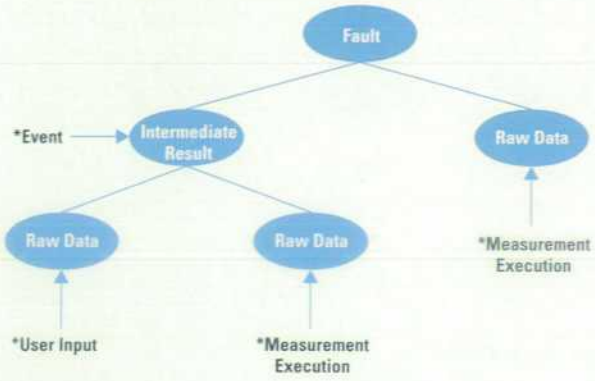


Fig. 6. Hierarchical data abstraction built by tracking requests and responses.

Usability is enhanced by using the hierarchy to demonstrate to the user the steps taken to reach a conclusion, allowing the user to examine why a request was made in the first place. The modules that post requests and responses on the blackboard are required to provide human readable explanations related to these postings. The explanations include how the information was generated, what the information means, and in some cases, what can be done about some problematic situation. Because users have control over what level of decomposition they desire to see, the hierarchy also protects them from the need to look at all of the details of the diagnosis. The user can select an interesting high-level item and pursue some of the low-level details of that item.

The hierarchy is designed to eliminate dependencies between modules. This enhances the ability of knowledge engineers to represent knowledge in a way that is reusable and maintainable. Information from one source can be used by a variety of sinks that synthesize additional information. If a better way of generating the information is determined, the source can be changed without having any impact on the sinks. This will allow the product to evolve over time as our understanding of the problem improves and as other capabilities of the product evolve.

Procedural versus Declarative. The inference engine provides an environment for executing knowledge about diagnosis. This knowledge is represented in a declarative form using rules. The rules represent relationships between facts that transcend the procedures for proving those facts. However, parts of the diagnosis process require the ability to represent the procedural aspects of diagnosis explicitly. A procedural representation can be thought of in terms of a program written in a language such as C, Pascal, FORTRAN, and BASIC.

In the Fault Finder, the instrument measurements and user inputs represent procedural components of the network diagnosis process. The measurements embody complex processes for gathering data about the network. The user inputs allow the user to perform a procedure that is not easily automated. The modules that provide this procedural capability have been designed to interface with the blackboard in the same way as the inference engine.

Measurements. When a request for data on the blackboard can be satisfied by running an instrument measurement, the measurement is initiated and its results are posted on the blackboard. A simple example would be running an adapter status measurement. The data requested is the status of a particular interface card or the ability to contact the node associated with the card. The blackboard forwards this request to the measurement module where the adapter status measurement is handled. The result of running the measurement is that the status and the basic ability to communicate with the node are posted as responses on the blackboard.

When the results get posted, other modules that need this information can proceed with the synthesis of additional information. For example, the failure of a token ring station adapter status may be just one of the conditions of a rule that diagnoses some fault. When the station adapter status results are received by that rule, the rule may proceed with evaluation of the remaining conditions.

User Input. User input is handled very similarly to measurements. When information is requested from the blackboard that the user input module is capable of generating, the blackboard passes the request to the input module handling the user request. A description of the information and how to determine the correct response is provided as part of the user interface interaction. The user will perform the procedures required to determine the correct response and then enter or select an answer. The module that requested the information will then proceed with its synthesis.

Flow of Control. To satisfy the design objectives stated earlier, the flow of control within the system must be carefully controlled. The inference engine has a number of control flow characteristics that can be controlled including forward and backward chaining, cost and confidence parameters for rules, and the urgencies of requests placed on the blackboard. One of the key characteristics of the inference engine is its ability to suspend threads of inference while some of its requests are blocked to pursue other threads of inference.

Multithreaded PROLOG. When the inference engine requests information from the blackboard, there is no guarantee that the information will be available or that the request will be immediately selected as the next request to satisfy. The inference engine must be able to suspend its inferencing related to a request until the response is available. Also, while waiting for a response, the inference engine must be able to initiate other chains of inference to satisfy other requests it receives.

To make this possible, the blackboard allows context information to be stored with each request. This allows a requester to resume synthesis when the requested data becomes available. When the blackboard notifies a module with the new information, the context information is returned to the module. For cases in which multiple responses match a request, the context information is copied to create an equivalent but separate context for each response. This allows all of the backtracking capability of PROLOG to be provided in the blackboard environment. The context information also helps when presenting explanations to the user.

Prioritization. When modules request information from the blackboard, an urgency level is associated with the request. When the request becomes the one with the highest urgency, a module is selected to satisfy the request. The module with the lowest-cost technique for satisfying the request is selected as the exclusive provider of the response. Rules have mechanisms for passing default urgencies for new requests or for increasing or decreasing the urgencies of new requests. Other modules can set the appropriate urgency of requests for their form of synthesis. Each module must be capable of providing an estimate of its cost for any given request.

Tying Components Together. The inference engine is the key to enabling the Fault Finder to operate the instrument in place of a human user. The rules in its knowledge base represent the ability to perform a diagnosis of some fault in a network. The inference engine requests information from the blackboard and causes measurements to be executed or user input to be solicited.

The blackboard is the key to operating in a manner that is understandable and justifiable. Information is stored that allows the user to understand how information is synthesized and why any particular step was taken. The hierarchical nature of the data allows the user to control the amount of information being presented.

The inference engine and the Network Advisor measurements allow the Fault Finder to adapt to a variety of protocol domains. A knowledge base with rules about Ethernet is combined with a measurement set for Ethernet to allow the Fault Finder to find faults on Ethernet networks. The same is true for token ring, TCP/IP, Novell, and other domains. The knowledge for the various domains can be combined to address more complex situations.

The multithreaded nature of the inference engine and the context storage and prioritization mechanisms of the blackboard allow progress to be made in troubleshooting one problem while progress on another problem is impeded. In addition, the prioritization mechanism allows a new and more important problem to take precedence over a less important problem. This is important to avoid investigating small or petty problems while the potential for a disastrous problem exists.

Finally, forward and backward chaining are strategically applied to create the observe, hypothesize, and prove behaviors. Forward chaining rules inform the blackboard that certain information can be used as soon as it becomes available. This sets up the measurements to be made during the observation stage. When the information becomes available, the rule decides if a problem might exist. If so, a request to investigate the problem is created. This request represents upgrading the state of the problem to the hypothesized level. The blackboard then attempts to prove or disprove the problem's existence, which generally triggers the execution of a backward chaining rule. The backward chaining rule will request additional data, which will generally lead to measurement execution and gathering of user input. As a result of this activity, a response is posted on the blackboard, and the fault's existence is proved or disproved. This process may happen for multiple problems during any given session and various certainties will be associated with each conclusion. Users can use these certainties and their own intuition to decide which problem to fix. Fig. 7 summarizes the activities that occur during this fault finding process.

A Fault Finder Example

The example in this section will show how the Fault Finder's expert system capability uses the observe, hypothesize, and prove paradigm to identify and solve a token ring network problem.

A token ring LAN is configured as a logical ring. It consists of a set of computing devices, called stations, connected to the physical wire (see Fig. 8). The logical ring can operate at either 4 Mbytes/s or 16 Mbytes/s, and a station connected to the ring must be configured to the correct operating speed. The stations or spanning devices on the ring are connected to a multistation access unit (MsAU or MAU). These MAUs are usually combined in racks in wiring closets. A MAU port contains a shorted connection (using a relay). When a station is inserted into the ring the station applies a dc voltage to the media interface cable (or lobe) that attaches the station to the MAU. This voltage switches the relay in the MAU and serially connects the station into the ring without affecting the normal operation of the ring.

The operation of a token ring network is composed of many functions. However, for this example only the beaconing function will be discussed. The beaconing function attempts to recover the ring from hard errors. Hard errors, such as a station inserting at the wrong network speed, usually occur

Component	Activity	Phase
Inference Engine (Forward-Chaining Rules)	Indicate data that would initiate inferencing when it becomes available.	Observe
Blackboard	Initiate measurements that will provide indicated data.	
Instrument Measurements	Gather data and post it on the blackboard.	
Inference Engine (Forward-Chaining Rules)	Decide if a fault might exist based on available data.	Hypothesize
Inference Engine (Forward-Chaining Rules)	Post requests to prove or disprove hypothesized faults.	
Blackboard	Call upon appropriate modules to prove or disprove faults.	Prove
Inference Engine (Backward-Chaining Rules)	Request additional data to prove or disprove faults.	
Measurements	Gather data and post it on the blackboard.	
Inference Engine (Backward-Chaining Rules)	Use available data to prove or disprove faults and post conclusions on the blackboard.	

Fig. 7. Rules, activities, and data flows occurring during the fault finding process.

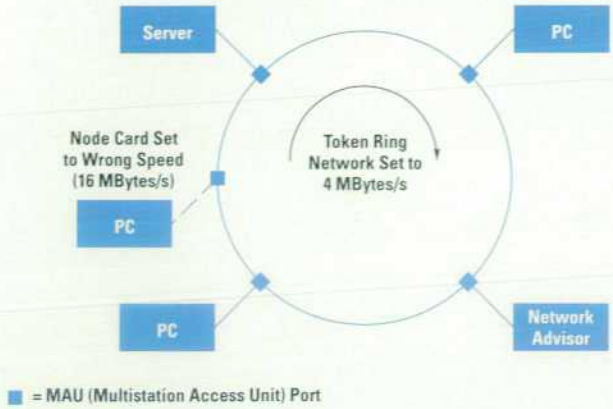


Fig. 8. The token ring layout for the Fault Finder example.

within the station and permanently impair the station's ability to communicate on the ring.

When a station detects an error on its nearest active path it sends a beacon frame containing the address of its upstream neighbor and the type of error encountered. This isolates the fault domain of the problem. The fault domain consists of the transmit path of the upstream neighbor station, the intervening cabling system (cables, MAUs, repeaters), and the receive path of the station. If the upstream neighbor of the beaconing station copies eight of these frames it removes itself from the ring and performs a self-test. If it passes the self-test the station will reinsert itself in the ring, and if it fails, the station will stay off the ring. If the self-test does not resolve the problem, the beaconing station will remove itself from the ring and perform a self-test. If it passes the self-test the station will itself reinsert in the ring, and if it fails, the station will stay off the ring. If the beaconing condition persists even after both stations have removed and reinserted, the condition is considered a permanent beaconing condition and will require manual intervention to resolve.

A station inserting at the wrong network speed is a common problem when a new workstation is installed on a token ring LAN. Specifically, a station inserting at the wrong network speed occurs when the network interface card is not configured properly for the network. The following scenario describes how the Fault Finder is able to troubleshoot this problem.

The scenario begins when a network manager is setting up a new Novell workstation on a token ring network and while attempting to attach to the server via the Novell netx command, the following error message is displayed on the workstation:

"A File Server could not be found."

This message does not necessarily point the network manager in the proper direction to solve the problem and may in fact misdirect the manager.

For this example, three rules and supporting predicates will be used from the knowledge base. These rules include the "inserting at the wrong speed" rule given earlier, and one rule each for broken or shorted transmit or receive wires. The following code shows a portion of the rule for the broken or shorted receiver wire problem. The rule for broken or shorted transmit wire is the same except that the word transmit is used instead of receive.

```
;*******************************
; Broken/Shorted Receiver Problem
;Problem description
problem(
  name( BrokenShortedRxProblem )
  nlsName( 'Broken\Shorted Receiver' )
  eventType( #FaultEvent )
  frequency( 50 )
  severity( 50 )
  definition( 'The network interface card's receiver is bad, the receiver
        minus lead is broken, or the receiver pair is shorted
        together.' )
  solution( 'Run a Network Advisor Lobe test on the lobe wire of the
        specified station. This determines if the problem is the
        wire or the station itself. If the lobe test passes, replace
        the network interface card and reinsert the station.  ')
  hypoText( 'Station %?%address% may have a broken\shorted receiver. ' )
  concText( 'Station %?%address% %+%has%%-%does not have% a
        broken receive minus lead or shorted receive pair. ' )
    .
    .
    .
;Forward Chaining Rule
hypothesize(
  name( hBrokenShortedRx )
  cost( 50 )
  confidence( 90 )
  explanation( 'If monitor contention is not resolved (times out) and
        the ring station enters beacon-transmit mode and transmits
        a beacon MAC frame, then it is possible to have a broken
        or shorted receive pair.' )
  logicText( 'BrokenShortedRxProblem( ?address ?addressNAUN ) :-
    .
    .
    .
;Backward Chaining Rule
backward(
  name( cBrokenShortedRx )
  cost( 50 )
  confidence( 90 )
  explanation( 'The beaconing station will remain in beacon transmit
        mode until the signal is restored by the removal of the
        station with the broken/shorted receive pair
        through the beacon-transmit auto removal test. This removal
        is verified by running a Station Adapter Status measurement
        to determine if the ring station with the broken/shorted
        receive pair has actually been removed.' )
  logicText( 'BrokenShortedRxProblem( ?address ?addressNAUN  ) :-
    .
    .
    .
```

When the Fault Finder begins executing, the forward chaining rules invoke measurements to monitor (observe) the network. The token ring commentator is an example of a monitor measurement. The token ring commentator provides a high-level abstraction of significant protocol events. Significant protocol events are defined as preludes to network performance degradation or network failure. The token ring commentator allows the network troubleshooter to identify network problems without sifting through several pages of protocol decodes.

The following code shows a portion of the module for beaconing events, which are reported to the token ring commentator when a network card inserts in the network at the wrong speed.

```
;
*****************************************************************
; Token Ring Network Events
;
*****************************************************************

event(
  name( beaconMacFramesMonitor )
  nlsName( 'Beacon' )
  eventType( #ProtocolEvent )
  frequency( 50 )
  severity( 50 )
  definition( 'A Beacon MAC Frame is transmitted if a station detects the
        expiration of the claim token timer during the monitor
        contention process. The station will broadcast a Beacon MAC
        frame isolating the domain to itself and its upstream
        neighbor.' )
  solution( '' )
  hypoText( 'Monitor for Beacon MAC Frames ' )
  concText( '%+%Station %%?+%address%%+% transmitted a beacon
        MAC frame. %%-%No beacon MAC frames encountered.% ' )
  parameters(
    [ useBaselines #string #hypothesis '' '' ]
    [ address #string #conclusion '' '' ]
    [ addressNAUN #string #conclusion '' '' ]
  )
)

event(
  name( beaconingMonitor )
  nlsName( 'Beaconing' )
  eventType( #ProtocolEvent )
  frequency( 50 )
  severity( 50 )
  definition( 'The ring is considered beaconing if a station has transmitted
        8 consecutive Beacon MAC Frames. ' )
  solution( '' )
  hypoText( 'Monitor for the Ring Beaconing ' )
  concText( '%+%Station %%?+%address%%+% is beaconing station
        %%?+%addressNAUN%%-%The ring is not beaconing%. ' )
  parameters(
    .
    .
    .

event(
  name( streamingBeaconsMonitor )
  nlsName( 'Streaming Beacons' )
  eventType( #ProtocolEvent )
  frequency( 50 )
  severity( 50 )
  definition( 'The ring station has been transmitting Beacon MAC frames.' )
  solution( '' )
  hypoText( 'Check for the Ring Streaming Beacons ' )
  concText( '%+%Station %%?+%address%%+% is streaming beacons at
        station %%?+%addressNAUN%%-%The ring is not streaming
        beacons%. ' )
  parameters(
    .
    .
    .

event(
  name( newActiveMonitor )
  nlsName( 'New Active Monitor' )
  eventType( #ProtocolEvent )
  frequency( 50 )
  severity( 50 )
  definition( 'The new active monitor indicates the ring has recovered and
        is proceeding with normal operation.' )
  solution( '' )
  hypoText( 'Check for Ring Recovery ' )
  concText( '%+%New active monitor is station %%?+%address%%-%No
        new active monitor MAC frames encountered%. ' )
  parameters(
    .
    .
    .
```

The rules are then blocked pending measurement results, which satisfy the rules' preconditions. Once the results are received they are posted on the blackboard. A network interface card attempting to attach to a token ring network at the wrong network speed will cause a temporary beaconing condition on the ring. The token ring commentator measurement will identify beaconing on the ring and abstract the beaconing condition into four different stages. The first stage, beacon, identifies that beaconing has been initiated on the ring. The second stage, beaconing, indicates that beaconing has occurred long enough for the upstream station to remove and perform a self-test. The third stage, streaming beacons, indicates that beaconing has occurred long enough for the beaconing station to remove and perform its own self-test. The fourth stage, catastrophic, indicates a permanent beaconing condition. This particular beaconing condition causes the upstream station and the beaconing station to remove themselves from the ring.

The token ring commentator measurement observes the first three stages of beaconing and posts the observations to the blackboard. The observations are displayed in the Fault Finder's Observations tile shown in Fig. 2. Following the beaconing condition, the token ring commentator measurement also observes that a new active monitor is elected. This observation is used by the Fault Finder to conclude that the beaconing condition was temporary and that the ring has recovered. Since the observations posted on the blackboard satisfy the preconditions specified for the three rules mentioned earlier (broken/shorted transmit wire, broken/shorted receive wire, and inserting at wrong network speed), the problems can be hypothesized.

The problems hypothesized by the Fault Finder are a result of inferencing through the antecedent part of the rules. The possible problems are displayed on the Fault Finder's Possible Faults tile to show the user the current problems the Fault Finder is investigating (see Fig. 2). This feature is provided because the Fault Finder may have enough information to hypothesize a problem, but might not be able to prove that the problem exists. This may occur because:

- The Fault Finder cannot obtain the required information through measurements
- The Fault Finder cannot obtain the required information from the user
- The knowledge base does not have the ability to prove (or disprove) the problem.

The hypothesized problems are prioritized to allow a more important problem to take precedence over a less important problem. Therefore, the Fault Finder will investigate the excessive ring length problem first because this problem could potentially effect the entire network while the other problems are most likely localized to a single user.

The Fault Finder is able to obtain information on its own about the state of the network to prove (or disprove) hypothesized problems. This is performed by the rules' requesting information (via the inference engine) from the blackboard. The blackboard requests the data from the appropriate measurement modules and the results of the measurements are posted on the blackboard to allow the inference engine to continue and eventually prove (or disprove) the hypothesized problem.

In this example the hypothesized problems (broken transmit wire, broken receiver wire, or inserted at the wrong speed) are proved (or disproved) by determining which device (if any) was beaconed off the ring as a result of the problem. This is determined by transmitting a token ring adapter status MAC frame to the suspected devices. The addressing information to determine which devices to query is taken from the observations made during the temporary beaconing condition. The rules will execute and configure the adapter status measurement to obtain the required status information from the device. The response or lack of a response from the adapter status measurement will be posted on the blackboard and used for further inferencing. In this particular example, neither of the devices was permanently removed from the token ring network. Therefore, the Fault Finder will conclude that there was not a broken transmitter wire or a broken receiver wire, but that a station with the wrong speed was inserted between the specified upstream and downstream stations. Notice for this particular problem the confidence level is indicated as [High] as shown in Fig. 2.

When the Fault Finder discovers a problem on the network, the user is notified. This notification provides the user with information about the problem, a definition of the problem, and the reasoning and required actions to solve the problem. This information is provided in the Fault Finder's explanation facility shown in Fig. 4. For a station inserting at the wrong speed, the required actions tell the user to change the setting on the network interface card, and provide an example of how to perform this task on an IBM Adapter II network interface card.

This problem (inserting at the wrong network speed) fits well into the observe, hypothesize, and prove paradigm. The Fault Finder can observe or passively monitor the network for significant events to hypothesize possible problems. The real strength of the Fault Finder, however, is its ability to run instrument measurements automatically, to obtain status information from different network devices, and to prove that a problem exists.

Conclusion

Protocol analyzers provide powerful measurement capability that allows experienced LAN troubleshooters to solve many network problems. The Fault Finder provides the next generation in LAN troubleshooting tools. It automates the process of troubleshooting, allowing network managers to focus their efforts on those problems requiring human attention. It incorporates the knowledge of expert troubleshooters into its rule base, allowing network managers to take advantage of a powerful problem solving instrument. Finally, the Fault Finder uses the same troubleshooting model as expert troubleshooters—observe, hypothesize, prove.

Acknowledgments

This work has been the result of discussions with many people. Many thanks to Cliff Frost at the University of California at Berkeley, Bruce Hitson at Teknekron Infoswitch Corporation, and Randy Strickfaden, Peter Haddad, Jeff Hintzman, and Simon Lewis at Hewlett-Packard Laboratories. Many people have contributed to the design and implementation of the Fault Finder. Our special thanks to Dave Fish, Mark Smith, Tom Wisdom, Paul Kingsley, and Bill Marbaker.

References

1. R.S. Englemore-and T. Morgan, *Blackboard Systems*, Addison Wesley Publishing Co., 1988.

2. V. Jagannathan, R. Dodhiawala, and L.S. Baum, *Blackboard Architectures and Applications*, Academic Press, 1989.

3. I. Bratko, *PROLOG: Programming for Artificial Intelligence*, Second Edition, Addison Wesley Publishing Co., 1991.

Bibliography

1. B.L. Hitson, "Knowledge Based Monitoring and Control of Distributed Systems," *Technical Report no. CSL-TR-90414*, Stanford University, 1990.

The User Interface for the HP 4980 Network Advisor Protocol Analyzer

A PC-based, object-oriented software architecture forms the underpinning for the HP 4980 Network Advisor's user interface.

by Thomas A. Doumas

The HP 4980 Network Advisor protocol analyzer's user interface provides LAN troubleshooters with a clear, concise, and consistent presentation of measurement results. The user interface is built on a graphical, window-based system. The user interacts with a number of system windows to access and control the features of the instrument. This interaction is through pull-down menus, pushbuttons, list boxes, and dialog boxes associated with specific features. Support for servicing these user interactions is provided by a layer of software called the measurement architecture. The measurement architecture software and other system software are collectively called the general-purpose environment. The general-purpose environment software is written in the object-oriented Smalltalk language and runs on a PC.

Working in consort with the general-purpose environment is another environment called the analysis and real-time (ART) environment, which runs on a RISC-based hardware platform and provides the services for interfacing to the Network Advisor's front panel and the network under test. A high-level view of the general-purpose and ART environments is shown in Fig. 1. The ART environment is described in detail in the article on page 29.

The following features are provided through the Network Advisor user interface:

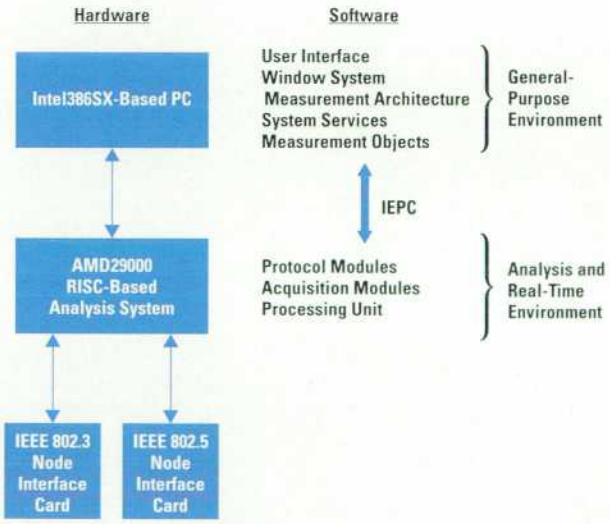


Fig. 1. A high-level view of the main hardware and software and components of the Network Advisor. IEPC is the interenvironment process communication channel.

- Simultaneous execution of measurements. The user can execute multiple measurements simultaneously. For example, users can start a traffic generation measurement to produce a specific network load and simultaneously monitor the frames with protocol decodes and statistics measurements.
- Graphical display. The Network Advisor user interface is enhanced by a graphical display. This 16-color, VGA display system provides a platform for displaying statistical information with presentation tools such as gauges, graphs, and bar charts. The network statistics measurement provides the user with line graphs correlating multiple network parameters and gauges that change color to indicate threshold events.
- Consistency across measurements. All Network Advisor measurements are controlled with a common user interface. The user executes, configures, pauses, and stops all measurements with the same mouse clicks or keystrokes. Each measurement has a set of standard menu items for controlling these common features.
- User-definable measurement presentation. The Network Advisor presents the measurement display by grouping the measurements into categories and subcategories. The standard set of categories is first indexed by protocol stack* and then by measurement type (e.g., statistics, timing, performance, etc.). Users can create their own categories and subcategories containing their choice of measurements. This feature allows the Network Advisor measurement selection window to be customized for specific tasks. Since measurements can appear in multiple categories, new categories do not interfere with existing categories.
- Online help system. The Network Advisor software includes an online help system. The help system provides help on the use of Network Advisor features and help on specific data-communications topics such as network protocols.

Measurement Organization

The Network Advisor presents the user with functionality oriented around the measurements on the network. Some examples of measurements include individual protocol decodes, protocol stack decodes, traffic generation, network

* A protocol stack is a group of protocols that work together to provide a service for network communication, and represents one possible choice of protocols for the seven layers of the ISO OSI Reference Model. For example, the ARPA stack defines the protocols FTP, SMTP, and telnet for the application layer, TCP for the transport layer, IP for the network layer, and various protocols (e.g., IEEE 802.3 and 802.5) for the data link layer, and leaves the other layers undefined.

performance summary statistics, automatic node discovery, and the node generating the most errors.

Traditional protocol analyzers provide access to their functionality by grouping features into a small number of predefined, fixed functional areas. For example, the HP 4972A LAN protocol analyzer groups functionality into the broad areas of decodes and statistics. The statistics functionality group is composed of a complex menu tree which gives the user access to features as diverse as network performance statistics, connection matrices, and traffic generation. This menu tree presents the user with a variety of parameterization menus along the way. For example, the decodes functionality group presents the user with parameterization menus for selecting protocols, protocol layers, and display formats.

The Network Advisor's functionality is accessed through measurements. Each measurement is self-contained and has a set of configurable parameters that are specific to that measurement. All measurements use the same user interface style for parameterization. The presentation of the measurements is controlled by the categories and subcategories that act as view filters. If the standard categories are not intuitive or useful for a particular situation, the user can create custom categories.

This Network Advisor measurement organization provides the user with presentation functionality. Users can select the measurement needed without distractions from unrelated data. This is an improvement over the collections of functionality in fixed pieces and the different user interfaces provided in traditional protocol analyzers. Fig. 2 depicts this difference in instrument organization. The upper portion of the figure depicts the Network Advisor concept and the lower portion depicts traditional instrument organization.

General-Purpose Environment

The software architecture for the general-purpose environment is shown in Fig. 3.

The main areas in the general-purpose software environment include:

- Applications. These are the modules that provide features such as protocol decodes and statistics to the user and handle the displays and input from the user.
- Frameworks. These are groups of classes that provide the foundation on which many of the user interface features are built.
- Measurement Architecture. This is a set of global objects and classes that control shared resources and provide access to system functions. Shared resources include the hardware in the ART environment used to capture and hold data coming from the network under test.

The software in the general-purpose environment is implemented in the object-oriented Smalltalk language (see "Object-Oriented Design and Smalltalk" on page 24).

User Interface Frameworks

The Network Advisor user interface is built using multiple layers of frameworks. A framework is a group of classes that implement commonly used features such as printing, window system control, and paging and searching through data. A framework is generally used without modification by the layer of software above it. In some cases the framework classes are subclassed for slight behavior modifications. Classes implemented at higher layers of the system do not "tunnel" through lower layers to access layers below their adjacent layer. This rule enhances the maintainability of the system.

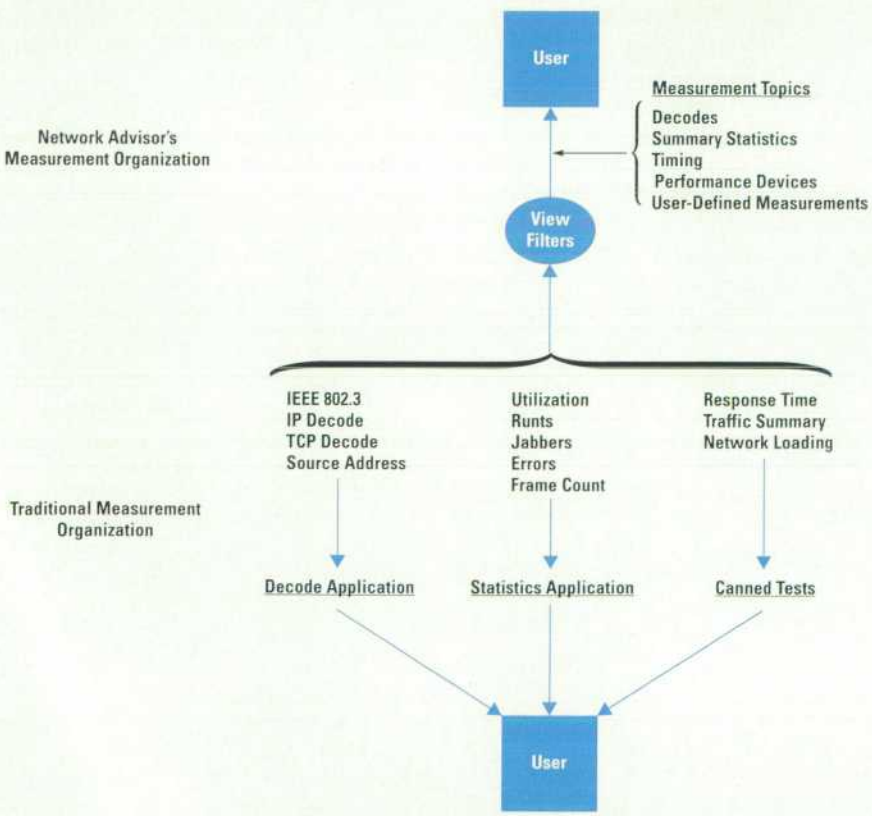


Fig. 2. Differences in measurement organization between traditional protocol analyzers and the Network Advisor.

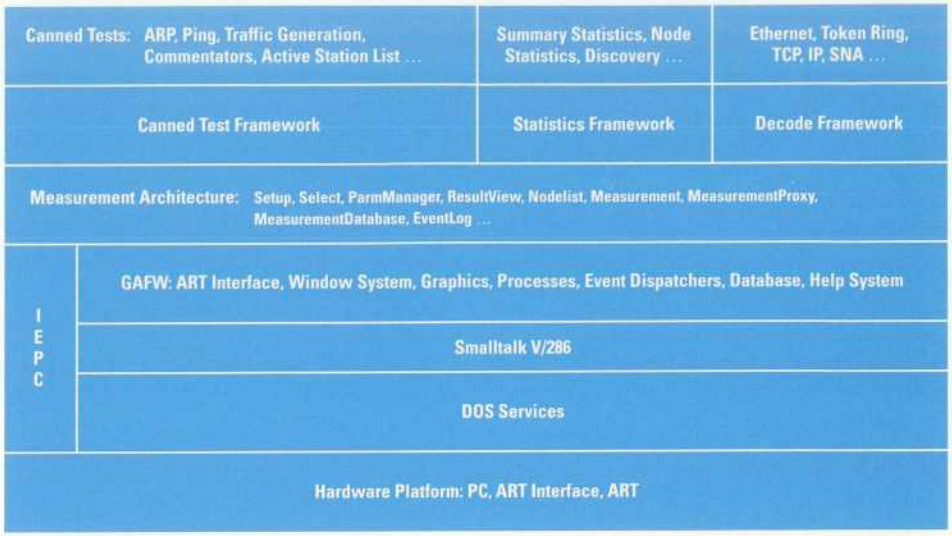


Fig. 3. The software architecture for the general-purpose environment. IEPC is the interenvironment process communication channel.

Generic Application Framework (GAFW). The core system services of the Network Advisor are provided by a group of classes called the generic application framework. These classes implement the windowing system, ART interface, error handling, and so on.

Specific Application Frameworks. The specific application frameworks provide a set of classes to implement a class of measurements that have common features. The decode framework is a specific application framework upon which all the protocol decode measurements are built. This framework is different from the others because it supports a post-processing mode of operation. In addition, the decode framework provides a data throttling protocol* for run-time decode displays that are not required to maintain real-time display of received frames. The statistics framework, which is another specific application framework, integrates multiple statistical measurements into a single composite measurement. It provides different ways of showing statistical data such as generic graphs, gauges, pie charts, and bar charts. It also provides configuration capability for each component measurement.

Canned Test Framework. The canned test framework supports all of the canned tests such as ARP,** ping, traffic generation, protocol commentators, and active station list. This framework focuses on programmatic control of the front-end data transmission interface and real-time display of results.

Measurement Architecture

The measurement architecture is a software platform that defines and implements a set of standard features and interfaces to system functions. These standard features and interfaces are implemented as global objects and thus are available to all measurement objects in the Network Advisor. Software developers can use the classes of objects in the measurement architecture as they are, or modify their behavior with subclassing. The objects in the measurement architecture provide the classes to create the following user interface features.

* A protocol in which the decoder sends a message to the ART environment to retrieve the next eight frames of data.

** ARP (address resolution protocol) is used to find the physical address for a specific IP address in an ARPA protocol stack.

Object-Oriented Design and Smalltalk

Object-oriented designs are based on the data that is present in the system. The object-oriented model defines objects that encapsulate data and provide all defined operations (methods) that act on the data. The entire system is modularized on the basis of the data structures. This is in contrast to procedural designs, which focus on algorithms and features. The main benefit of the object-oriented approach is that the data provides more stability over time than algorithms because only the methods in the object can modify data, whereas with procedural designs, data structure changes and global access to data affect the stability of data.

The major elements of the object-oriented model are: abstraction, encapsulation, hierarchy, and modularity. Abstraction is defined as the description of a system that focuses on the details that are significant while hiding the details that are not significant. Abstraction describes the external behavior of the object or system. The concept of encapsulation means that the data structures of an object are accessed only through a publicly defined interface to the object and that the implementation details of the object are hidden. This gives the programmer the freedom to reimplement the object for improving performance or repairing defects without worrying that some user of the object is dependent on the specific data structure or the implementation of the methods that operate on the data. The hierarchy of the object-oriented design is an ordering of the abstractions that define the system. In Smalltalk, the class hierarchy defines which classes can inherit functionality. For example, the class **Dictionary** inherits from the class **Collection**. Modularity refers to abstractions grouped into discrete units. The modules should be loosely coupled so that changes in one module will not require modification in others.

In Smalltalk, abstractions are defined in classes. A class contains data and methods that operate on the data. A program is built by creating instances of the classes and tying the instances together to create the desired solution.

To implement the user interface and the other classes in the general-purpose environment, we used SmallTalk/V286 from Digitalk Inc., and a development environment called Envy/Developer from Object Technology International. Envy/Developer provides a network-based (Novell Netware) team programming environment with tools for tasks such as source code control, revision control, debugging, and software production.

Bibliography

1. T. Kraemer, "Product Development Using Object-Oriented Software Technology," *Hewlett-Packard Journal*, Vol. 40, no. 4, August 1989, pp. 87-100.

2. P. Munsch and S. Witten, "Object-Oriented Design in HP IVI," *Hewlett-Packard Journal*, Vol. 41, no. 5, October 1990, pp. 29-30.

Features Selection. The user interface measurements window presents the user with all of the available measurements.

The menu items in the measurements window give the user the ability to open a measurement to run, to configure a measurement, to stop all measurements, and to run multiple measurements. In addition to measurement control, the user can import measurements created on another Network Advisor.

Measurement Configuration. The Network Advisor provides a set of user-modifiable parameters to control specific features of a measurement. These parameters can affect the presentation of data, and in some cases they can affect data transmitted on the network. Users access the configurable parameters of measurements with the Config menu item, which appears in each measurements window. The parameters are available to developers via a programmatic interface.

User-Defined Measurements. The Network Advisor measurements can be cloned into new, user-defined measurements. This feature allows users to configure a measurement for a specific task and then create a new measurement using the same configuration.

Display Management. All window objects use the display manager to display themselves when they are ready. The display manager keeps track of the location of all windows and suggests a size and placement for new windows. The display manager provides a similar function for the icon area.

Measurement Control. All measurements can be started, stopped, or paused. In addition, all measurements can modify the source of data (e.g., capture buffer or the network under test).

Intermeasurement Communication. Because the results of one measurement might be of interest to another measurement, the measurement architecture defines a standard communication path for intermeasurement communication. The requesting measurement only needs to register for results using a programmatic interface to receive the results from another measurement.

Database. The measurement architecture provides a database facility for the Network Advisor. Any measurement or system function can define and use the database facility. The node list browser is an example of database use.

Interface to Analysis System. The ART and general-purpose environments interact through the interenvironment process communication (IEPC) channel. The IEPC channel manages bidirectional communication between the ART and general-purpose environments by buffering and dispatching messages in both directions. All commands to the ART environment are Forth strings (see "The Forth Interpreter" below). The measurement architecture provides a high-level interface that allows programmers to build a Forth command and send it to the ART environment with a single message. The high-level IEPC interface supports simple commands (status response only), complex commands (multiple responses), command response timeouts, error handling, and priority commands. When a command is sent, an object in the general-purpose environment is dynamically assigned an IEPC port number. This number identifies the sender of the command and the receiver of the response. Communication with global objects is supported with fixed destination ports.

The Forth Interpreter

In the Network Advisor, the user interface software on the PC communicates with a Forth interpreter in the analysis and real-time environment on the pod.* The Forth interpreter is used to configure and control the ART environment. Commands such as **start** and **stop** are sent to the ART environment as ASCII Forth strings. These strings are passed to the Forth interpreter for execution.

Since the ART environment is written in C++ and the Forth interpreter is written in C, we needed a way to interface the Forth interpreter to the C++ code. We decided that the best way to do this would be a call to a virtual function. This would allow objects to inherit Forth interfaces. Since most objects in the system are derived from a C++ class we defined as the root class, we decided that this was the place to define the virtual function. We also wanted to modify the Forth source code as little as possible.

We defined a global function **callForthFunc()** that takes two parameters: a pointer to an object and a pointer to the Forth stack. This function is written in a C++ module and can call the virtual function. It casts the object pointer to a root pointer and calls the virtual function **forthFunc()** passing the stack pointer. By indexing the stack pointer, parameters can be passed between Forth and C++.

By convention the first element of the stack is used as an index to tell **forthFunc()** what function to perform. If **forthFunc()** does not implement the requested function, it will call **forthFunc()** in the inherited class. This call chaining creates an inherited Forth interface.

At system initialization, the ART environment stores a pointer to its global record in a Forth variable. This global record is an object derived from the root class. The **forthFunc()** for this class implements functions such as instantiating an application, returning pointers to other objects in the system, and configuring system parameters. Supplying Forth with this one pointer allows it to make calls and gain access to the rest of the system.

Development Phase

The Forth interpreter was also used in the development environment. The ART environment and its applications were developed on workstations. The front-end code was simulated using a disk file. Frames were read out of the file and passed to the applications. The Forth interpreter was used to control the flow of these frames to the applications.

The Forth word **play**** took the number on the top of the stack as the number of frames to **play** to applications. The Forth word **step** was defined to do a **1 play**. This allowed frames to be played into the ART environment to examine the internal data structures between frames.

The Forth interpreter was also used as a debugging tool in the target environment. A user interface to the Forth interpreter called **Forth Window** was created on the PC. From this window, Forth commands could be sent to the Forth interpreter on the pod. This allowed us to get and set system variables, dump parts of memory to the debug port, query memory use, and so on.

Robert L. Vixie
Development Engineer
Colorado Telecommunications Division

* A pod is a plug-in module for the Network Advisor that contains the interface hardware and the real-time analysis processor system.

** The **play** command tells the ART environment how many frames to process.

Error Handling. The measurement architecture provides a variety of automatic error handling mechanisms. Errors generated by the ART environment, applications, system functions, Smalltalk, and DOS are all handled by these mechanisms. In many cases the Network Advisor can recover from an error and continue executing measurements. In other cases, the Network Advisor aborts itself and exits to DOS. In this case, data is saved in an ASCII file on disk so that it can be examined later to find the defect.

Hardware Configuration. The analysis hardware is controlled by the analyzer setup interface. The user can modify the hardware configuration via the setup window. Measurement objects can access the same parameters through a programmatic interface. The hardware configuration parameters are categorized into two groups: network interface independent and network interface dependent. The network interface independent parameters include capture buffer size, capture mode (circular or run until full), and capture filters (capture, exclude, or stop). The network interface dependent parameters include configuration commands for the IEEE 802.3 Ethernet network interface card and the IEEE 802.5 network interface card.

Node Lists. The node list feature allows the user to create and maintain node lists. The node lists are used by most measurements in the system to map physical network addresses to mnemonics that are meaningful to the user. The node list can be created automatically with a measurement object that monitors the network traffic to discover nodes. A set of utilities converts node lists from a variety of formats into Network Advisor format.

Event Log. The event log is a database of significant events that the Network Advisor has observed. These events are grouped into the following categories: protocol, threshold, topology, fault, and instrument.

Native Language Support (NLS). The Network Advisor software has built-in support for localization. The text displayed throughout the Network Advisor is provided by NLS dictionaries. Through the use of NLS dictionaries, the text can be localized without modification of the code.

PC Configuration. The Network Advisor software provides a set of configuration functions that are available to the user via the PC configuration window. These functions allow the display timeout to be set, measurements to start automatically when the analyzer is invoked, and DOS file operations to be carried out.

Measurement Execution

When the user requests via a menu selection to see certain measurement data, a number of objects are instantiated to perform the measurement, store the data, and display the results. Fig. 4 shows the system before a measurement object is instantiated and Fig. 5 shows the situation after a measurement object is instantiated. Some or all of the following events take place during measurement execution:

- To execute a measurement, the user selects the measurement in the MeasurementSelectView window and then chooses the Run menu item from the MeasurementSelectView menu (① in Fig. 5).

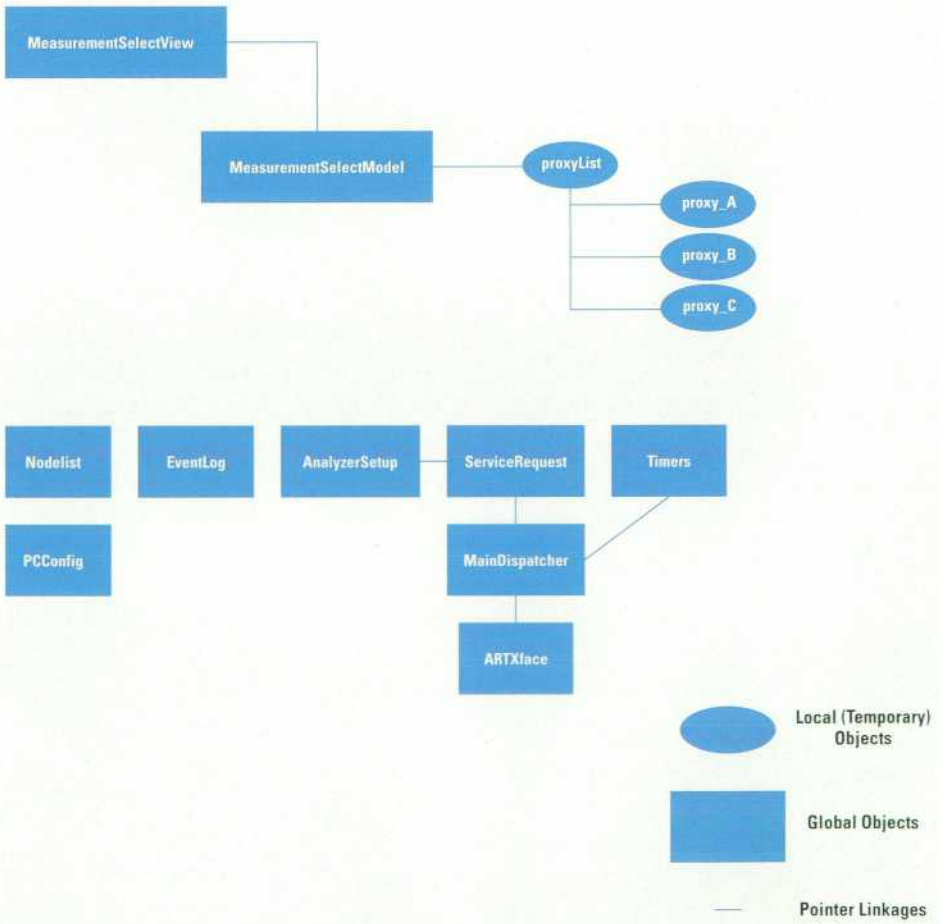


Fig. 4. Initial condition of objects in the system before the execution of a measurement.

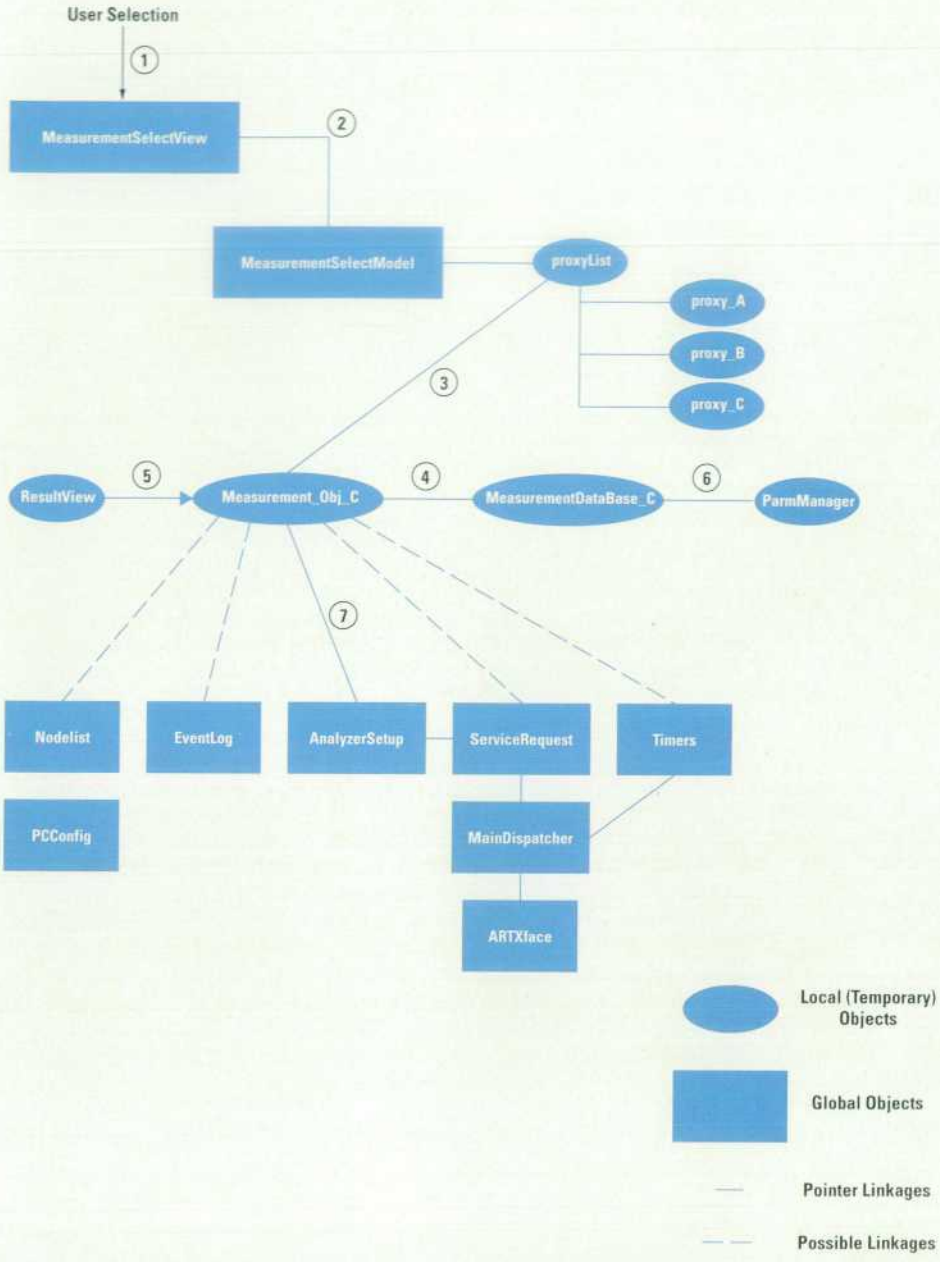


Fig. 5. Linkages and interactions when a measurement object (Measurement_Obj_C) is created to perform the measurement operations.

- After Run is selected, a message is sent to the MeasurementSelectModel indicating which measurement to run (② in Fig. 5). The MeasurementSelectModel holds a list of objects that act as proxies for the actual measurement object. The proxy for a measurement object holds the name of the class to instantiate for the specific measurement. It also holds the name of a measurement file that contains the data for the measurement. The proxy instantiates the appropriate class and then sends an initialize message to the instance (③ in Fig. 5). From here, the measurement object takes over and completes its own initialization. The measurement object reads its database from disk and creates the MeasurementDataBase object (④ in Fig. 5). The MeasurementDataBase holds the definitions and current values of the user configurable parameters for the measurement. In addition, the MeasurementDataBase contains the specification for the view that the measurement object is to build for displaying the measurement results. With this information the ResultView object is created (⑤ in Fig. 5).
- The ParmManager interacts with the MeasurementDataBase to allow the user to configure the measurement (⑥ in Fig. 5). Examples of configurable parameters include the sample period for a statistics measurement, timestamp mode for a decode, or percent utilization from a traffic generation measurement. The ParmManager object is instantiated when the user requests the configuration window for the measurement object.
- A measurement object might require that the node list be loaded from the disk-based databases into memory. If this is the case, then the measurement object sends a message to the global node list object to accomplish this.
- The measurement object might also post events to the event log as part of its startup sequence. An example would be the time the measurement started.
- Some measurement objects automatically configure to certain hardware or ART states. For example, the decode measurements will start the data capture if the data source indicated in AnalyzerSetup is the network under test. If the data

source is the capture buffer, then the decode measurements will request enough frames to fill the ResultView. The statistics measurements warn the user if the data source is not the network under test since statistics only execute in real time.

- After the measurement object is linked into the system and fully initialized, it will start executing. To do this, the measurement object sends a message to AnalyzerSetup and requests that the front-end measurement be started (⑦ in Fig. 5). AnalyzerSetup will get a unique handle (identifier) for the measurement and start the data capture. AnalyzerSetup keeps track of which measurement objects have requested a start measurement so that when the measurement objects request a stop, the front end will not be stopped until all measurement objects have requested a stop. In this way multiple measurements can run and stop independently.
- Measurement objects can send messages to the ART environment using the ServiceRequest object, which provides a high-level interface for sending Forth messages and receiving responses from the messages.
- When the user chooses to close a measurement, the measurement object breaks all of its references to system resources. This process includes removal of the ResultView object window from the window scheduler causing the window to disappear. The Smalltalk memory manager will then be able to recover the newly available memory.

Process Model

The process model is an important part of the general-purpose environment because it supports the simultaneous execution of multiple measurements while still providing a responsive user interface (see Fig. 6). Smalltalk supports separate processes with their own stacks of send messages. Processes all run within the same Smalltalk memory so they are more like threads than real processes with memory protection. The process scheduler is not time preemptive but rather selects a new process to run when an interrupt occurs or when the currently running process blocks, yields, or finishes.

The general-purpose environment runs two distinct processes. One process is the user interface process and the other is the background process. The user interface process

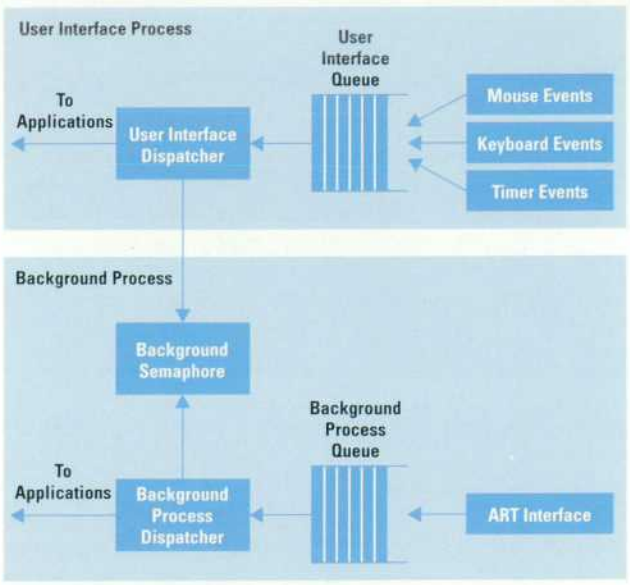


Fig. 6. Process model and dispatcher.

runs at a higher priority than the background process. The relative priorities are set this way so that the user interface can interrupt the background process and provide the user interface with good responsiveness.

The user interface process processes all keyboard, mouse, and timer events. It also blocks on a keyboard semaphore until it is awakened by the keyboard interrupt service routine.

The background process processes events that are generated by the ART system. Examples of background events are statistics and decode data units, front-end control information, and analysis control information.

Each process maintains a separate queue and dispatcher for storing the events generated for that process. This way, a large queue of background events cannot cause the user interface events not to be processed. This is the key to user interface responsiveness.

The Network Advisor Analysis and Real-Time Environment

The user interface and protocol decode applications of the HP 4980 Network Advisor use the services of a software platform that provides real-time protocol analysis and an interface to the network under test.

by Sunil Bhat

The analysis and real-time (ART) environment of the HP 4980 Series Network Advisor protocol analyzers is a software platform that has all the necessary services to support real-time network protocol analysis applications. To a lesser extent it also supports postanalysis of captured data. The ART environment is one of two major environments that represent the Network Advisor's software architecture. The other environment is the general-purpose environment. The general-purpose environment provides support for the general-purpose programming of the Network Advisor. Specifically, the general-purpose environment is responsible for the user interface, file management, and all other services that are essential at the user level. The general-purpose maintains all persistent information like node lists and setup configurations, and treats the ART environment like a device. Both of these environments are depicted in Fig. 1.

Some aspects of the ART design were leveraged from the CONE[1] (common OSI network environment) architecture, which provides a network-specific operating system for the HP OSI Express card and an environment for implementing OSI protocols. Specifically, the logical buffer services for encapsulation, segmentation, and reassembly of data were ported from the CONE implementation.

At the software level, the general-purpose and ART environments communicate with each other by an interenvironment process communication (IEPC) mechanism. At the hardware level, the ART environment executes on the AMD29000 RISC-based processor while the general-purpose environment executes on the Intel386SX-based processor and they communicate via a shared memory module. The general-purpose environment controls and configures the ART environment by sending messages via the IEPC mechanism, and the ART environment transfers the results of its analysis to the general-purpose environment via the IEPC mechanism.

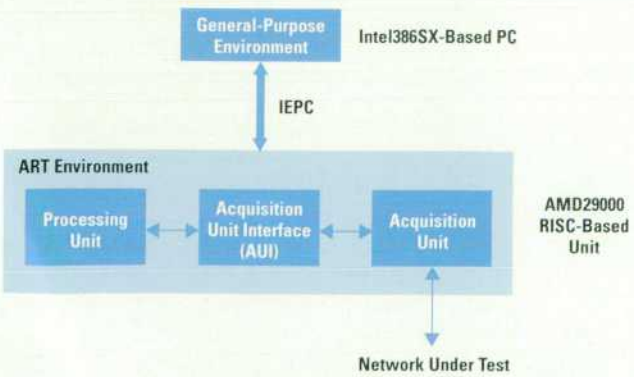


Fig. 1. High-level ART and general-purpose system architecture. IEPC is the interenvironment process communication channel.

Typically, any application on the Network Advisor has an analysis module that operates in the ART environment. This module does in real time all application-specific analysis based on relevant data from the network or network counts maintained by the hardware. There is also a corresponding module in the general-purpose environment that provides the user interface for the application. The application-specific user interface controls and configures its analysis module by commands sent across the IEPC. It also processes and displays the results sent by its analysis module to the user in some suitable format—graphical, tabular, or simple text. The Network Advisor allows multiple applications (also called measurements) to be active simultaneously. Therefore, at any given time there can be numerous applications active, each with its own analysis and user interface modules in the ART and general-purpose environments communicating across the IEPC.

The general-purpose and ART environments constitute the top-level logical entities of the Network Advisor's software architecture. The implementation of the Network Advisor software follows this logical design very closely. The IEPC implementation is split between the general-purpose and the ART environments. The general-purpose environment is written primarily in Smalltalk and the ART environment is almost entirely written in C++.

This article describes the architecture and high-level design issues of the ART environment. The user interface and more details about the general-purpose environment are described in the article on page 22.

ART Subsystems

As shown in Fig. 1, the ART environment consists of two subsystems: the processing unit and the acquisition unit. The processing unit contains hardware independent functions and the acquisition unit encompasses all hardware-specific low-level functions. The processing unit is designed to be hardware independent and can therefore be ported with relative ease onto other hardware platforms.

The acquisition unit is responsible for interfacing to the network under test and all the relevant hardware counters and timers. While connected to the network, the acquisition unit

captures real-time data and stores it in a buffer called the *capture buffer*. It also reports status information and does all other hardware specific housekeeping.

The processing unit is primarily responsible for real-time event processing. These events are typically data events stored in the capture buffer by the acquisition unit. The events could also be commands sent from the general-purpose environment or events generated within the processing unit. The processing unit also supports event post-processing. This allows the user to capture a full buffer of data that represents an interval of network activity, and then analyze it in postprocessing mode.

The Acquisition Unit

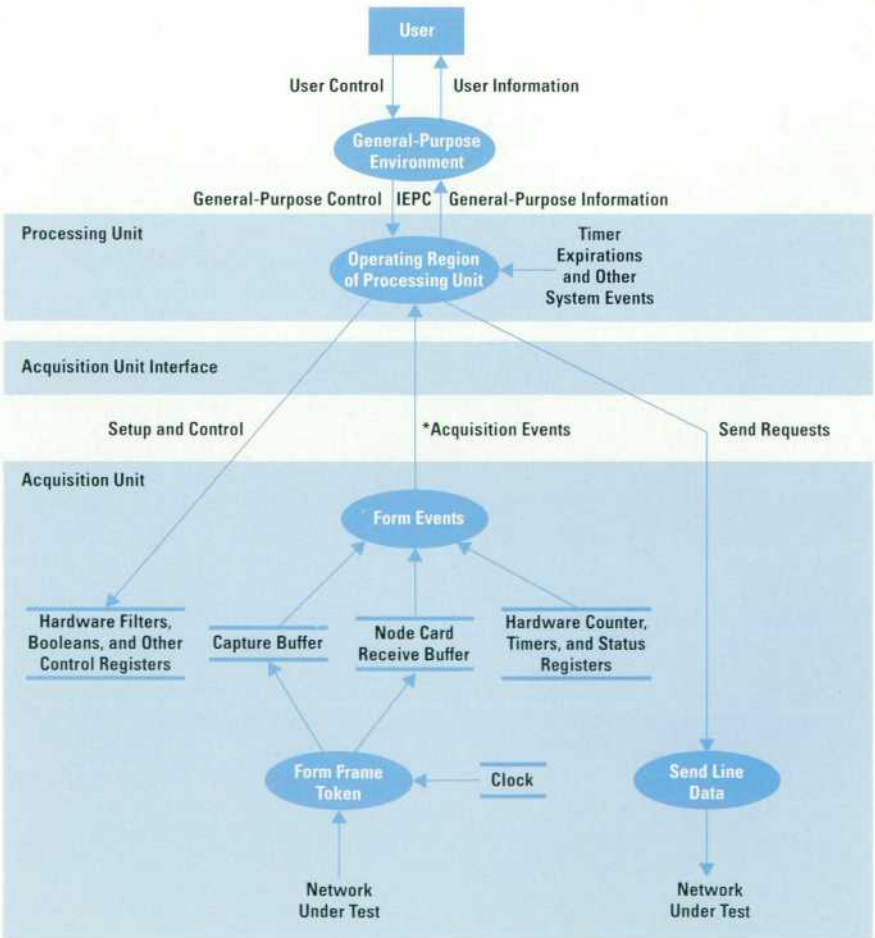
The data flow of the acquisition unit with respect to the rest of the system is shown in Fig. 2. The acquisition unit accesses the front-end hardware that interfaces to the network under test. The front-end hardware transfers MAC-level* frames from the network under test and stores them in the capture buffer along with a timestamp, length, and status information for each frame. These timestamps help protocol analysis modules correlate data with time. In the ART environment, frames that are stored in the capture buffer typically represent events. The introduction of a frame from the capture buffer to the processing unit is called a frame arrival event. Whenever an event is generated, it gets appended to the event buffer, which presents an event stream to the processing unit.

The processing unit uses the event data stream, control messages from the general-purpose environment, and timer information to produce two distinct data flows. The first data flow is the analysis information that is sent from the processing unit to the general-purpose environment via the IEPC. This information flow consists of the results from protocol analysis modules such as protocol decodes and network statistics executing in the ART environment. The results are packaged in an application independent form called *analysis items*. Analysis items are described in the article on page 34.

The other data flow results from send requests generated by the general-purpose environment. This flow, which can be a single frame or a sequence of frames, provides stimulus to the network under test. A traffic generator is a prime example of an application that can request entire sequences of frames to be transmitted repeatedly to create user-specified traffic on the network under test.

There is another distinct data path through the acquisition unit that is made up entirely of frames addressed to the Network Advisor. These frames are stored in the node card receive buffer and the acquisition unit reports them to the processing unit as node card arrival events. This data path,

* MAC stands for media access control, the lowest level of the protocol stack. Thus, MAC frames are the frames actually transmitted on the physical network media.



*Acquisition Events = Frame Arrival Events, Node Card Events, and Counter Read Events.

Fig. 2. Acquisition unit data flow.

along with the ability to send requests to the network, provides the basic node card functionality that enables the processing unit to make the Network Advisor behave as a valid node on the network and not just a monitoring environment. This functionality is essential for supporting remote capabilities as well as interactive applications like ping and ARP. At the lowest level the front-end hardware provides the physical interface to the network and is responsible for sending frames to and receiving frames from the network.

Acquisition Unit Interface

Access to the functionality of the acquisition unit is provided by the acquisition unit interface which marks the boundary between the processing unit and the acquisition unit. The interactions across this interface are summarized in the following sections.

Control and Configuration. The processing unit controls the acquisition unit. Among other things, it issues start and stop directives to initiate and terminate capture of data from the network under test. MAC-level frames are captured from the network in either continuous or autostop mode. In the continuous mode the capture buffer is viewed as a circular buffer that wraps around until the processing unit issues a stop command. In the autostop mode data is captured until the capture buffer is full, which triggers an automatic stop. Based on user selection, the processing unit configures the acquisition unit to operate in one of these modes. Apart from direct control, the processing unit also configures acquisition-specific parameters like the size of the capture buffer.

Most of the protocol information is available within the leading portion of a MAC-level frame, which is made up of protocol headers for each level of data encapsulation. The acquisition unit provides the user, through the acquisition unit interface, with the ability to specify the number of leading bytes of a frame that should be stored in the capture buffer. In such a case, the frame is said to be sliced to a specified length. This allows the user to store only the relevant portion of the frame, which effectively increases the capacity of the capture buffer.

Receive Data. The processing unit receives data events from the acquisition unit. These events are MAC-level frames from the capture buffer. For each frame captured by the acquisition unit, an information header is added by the front-end hardware. This header information includes a timestamp, the received length of the frame, and other status information detected by the front-end hardware (e.g., CRC errors).

Event Formation. The acquisition unit is responsible for the formation and reporting of all events or exceptions to the processing unit. As the frames get stored in the capture buffer, the acquisition unit generates frame arrival events to the processing unit. Similarly, the acquisition unit generates a node card arrival event for each frame received that is addressed to the Network Advisor. It also reports the hardware counts (counter read event) maintained by the front-end hardware for statistical applications. Any change in the run mode of the acquisition unit is reported to the processing unit as a run-mode change event.

Timestamping. The processing unit environment simulates time using discrete time information provided by the acquisition unit. All received data contains a timestamp. In the absence of data, periodic clock-tick events are sent to the processing unit. On the current hardware platform, the period for this event is 100 milliseconds, which defines the worst-case resolution of time in the processing unit environment. It should be noted that the hardware timer chips are initialized to the PC time at bootup. This way all analysis is correlated with a time base. Also, all analysis items sent to the general-purpose environment have a timestamp for synchronization of the general-purpose and ART environments.

Trapping and Filtering. The processing unit programs the acquisition unit trapping and filtering functionality using the set of filters and Boolean expressions provided by the hardware front-end. A filter can be specified to match a range of values for each of the first 127 bytes in a frame and the frame status byte in the frame header. A Boolean expression is the result of any logical combination of filters. A frame along with its hardware assigned header is stored in the capture buffer only if it satisfies at least one Boolean expression. The header provides a status field that reflects the results of the Boolean expressions matched by the frame. For example, if we wanted to capture all frames sourced by node A and destined for node B as well as all broadcast traffic on the network, we would set two filters. We would specify the source and destination address bytes in the first filter to match A and B respectively, and set all remaining bytes to match any value (i.e., don't care setting). The second filter would have the destination address bytes set to all ones (for Ethernet) while the rest are don't cares. Finally, we would set a Boolean expression to be the logical OR of the result of the above specified filters. The Network Advisor also has the capability to generate traps from software-matched Boolean expressions or external signals. Currently only software traps are implemented. The acquisition unit provides an indication to the processing unit when a trap condition has occurred.

Status. The acquisition unit provides status information to the processing unit about items such as the size of the unprocessed capture buffer and the length of the send queue. This status information is used by the processing unit for its internal activities such as flow control.

Send Data. The acquisition unit provides services for the processing unit to send data frames to the network under test. Each frame destined for the network is defined by the processing unit as an object called a send object. Entire traffic scenarios can be described as a list of send objects. This enables the acquisition unit to support traffic generator applications.

The Processing Unit

Exception services constitute the core of the processing unit, which is an event-driven system. The basic control structure of the processing unit is shown in Fig. 3. The various entities in the processing unit communicate with each other by means of exceptions. In the context of the processing unit, exceptions are events. Some exception types in the

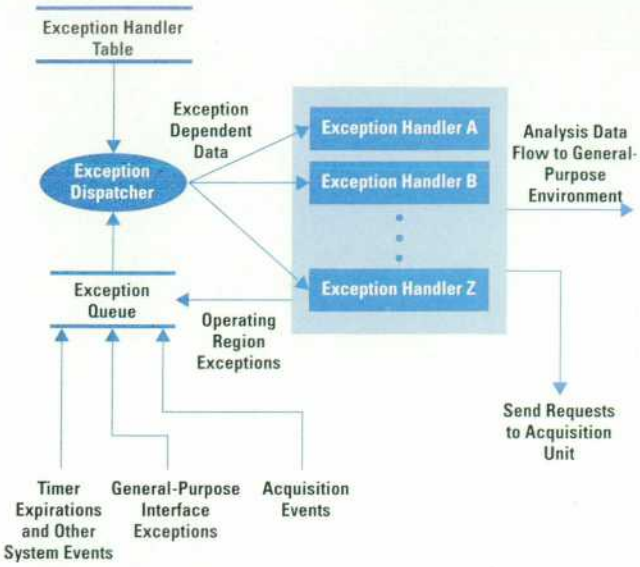


Fig. 3. Processing unit data flow.

system, like the arrival of a frame or the expiration of a second, are well-known. Applications can also dynamically allocate new exception types and generate them for their own use. Because of the event-driven nature of the processing unit, actions are triggered on the arrival of exceptions.

Protocol analysis applications executing in the processing unit environment are essentially actions that exist without any control of their own. These applications are called the exception handlers. Control passes to a particular exception handler when an exception occurs that it has shown interest in. This model is similar to environments that handle windowing applications. It is very effective in handling situations in which there are typically a number of events from different sources occurring asynchronously.

Exception Dispatching. The central entities in the processing unit are the exception dispatcher and an exception queue, which stores events arriving from different sources. The dispatcher dequeues exceptions from the queue and invokes all interested handlers that have previously registered for the exception. The dispatcher, the exception queue, and all other associated data structures constitute an operating region in the processing unit because together these items can be viewed as a process. The processing unit was designed to support multiple regions with different priorities. However, for simplicity the current implementation has a single region. Therefore, there is a single dispatcher that services all exceptions received at the processing unit's periphery. The decision to implement a single region in the processing unit greatly simplified its implementation, and also improved its response time.

To dispatch events to the appropriate handlers, the dispatcher uses a table that has an entry for each type of exception. Each entry in this table contains a linked list of zero or more exception handler tokens. A handler token is basically a pointer to an exception handler function or routine. Any application in the processing unit registers for specific exception types by enabling a handler token. This has the effect of adding the handler token to the handler lists for those exceptions the application is interested in. The handlers for a given exception are simply determined by indexing into the exception handler table using the exception token.

Each exception in the exception queue has a parameter token associated with it that contains all exception-specific data that needs to be passed to the handlers when the exception is serviced.

Event Synchronization. The dispatcher operates in a synchronous fashion in that only one exception is processed to completion before the next one is picked from the exception queue. The processing unit model is a cooperative nonblocking model. This means that a handler cannot be preempted before it completes processing. Therefore, the burden is on all applications in the processing unit to finish processing in a reasonable time.

The synchronous nature of the processing unit has a number of advantages for protocol analysis applications, which include:

- In a synchronous system there are no race conditions between various modules.
- Since each event introduced into the system is processed completely, there is no need to reorder events based on timestamp information.
- The correlation of analysis from different modules with respect to events and time is greatly simplified.

Timer Services. One of the central concepts with regard to correlation of analysis from different modules, based on events, is the notion of time. The processing unit maintains two time sources. One is called *real time*, which is driven by the periodic clock ticks from the hardware. On the current hardware platform these time ticks are 100 milliseconds apart. This defines the resolution of the real-time source. The other time source is *line time*. This is time simulated by the processing unit using the timestamps of the data frames received in the capture buffer. The line time is advanced each time one of the frames is introduced to the processing unit as a frame arrival exception. In the absence of frames, real time is used to update line time.

Using these two sources of time, the processing unit supports timers and the associated notion of timeout. Timers are represented by the data structure called *timer token*. Each timer token has its timeout value and a handler token for its handler function. The processing unit dispatches the timer token to its handler function when a timeout occurs. Any application module in the processing unit can create and use timer tokens. Standard timer functions like restarting, both in an absolute and a relative sense, and canceling a timer are all supported by the timer services of the processing unit.

For both real-time and line-time sources, all timer token expirations will occur chronologically. The processing unit guarantees that a timer token with a timeout of N seconds will not expire before current time plus N seconds, and no later than an event with a timestamp greater than current time plus N seconds. During run time, line time will track real time. In the case of postprocessing, line time is simulated using data frame timestamps alone.

Conclusion

The ART environment was designed to be an optimized environment for supporting real-time protocol analyzer applications. In addition to the basic ART environment, an extensive debugging and simulation environment was designed to support the development and preliminary testing of the processing unit on the HP-UX* operating system before integrating it with the acquisition unit running on the AMD29000 processor. This strategy allowed us to develop code in parallel with the development of the target hardware. It also reduced the integration effort required once the target hardware was available.

The success of the ART environment design is reflected in the relative ease with which applications can be written using ART environment services. In fact, different applications require different services, and as a result, the ART environment has been extended to provide frameworks for similar types of applications. The decode framework, which supports all protocol decodes and the canned test framework, which supports customized measurements for specific protocol analysis, are good examples of ART extensions. It should be noted that these frameworks in the ART environment have corresponding frameworks in the general-purpose environment.

The processing unit was designed to be hardware independent. This decision enabled us to provide the Network Advisor with functionality for Ethernet, token ring, and FDDI media. Since the acquisition unit interface is an application program interface to the hardware-specific software, we need only provide an acquisition unit for the hardware we are interested in. The rest of the basic ART environment and its extensions remain the same.

Acknowledgments

The design, implementation and testing of the ART environment and its supporting, debug, and simulation environments have been a team effort from start to finish. Numerous people were directly involved in this effort and many more provided inputs during development, and so it is hard to mention them all. I would, however, like to thank Bob Pinna and Bob Vixie who were major contributors to the design and implementation of the processing unit and the basic decode framework that supports decode applications in the ART environment. Thanks also to Jim Quan for providing the acquisition unit functionality for Ethernet/802.3 and token ring media. Many thanks to Jerry Morris for providing the basic operating system functionality for the AMD29000 and leading the integration effort and the port to the target hardware. Thanks to Rohit Mital for porting the logical buffer services from CONE. Finally, thanks to Betsy Perkins and Ed Moore for successfully managing this endeavor through some rough and uncertain times.

References

1. S. Dean, D. Kumpf, and M. Wenzel, "CONE: A Software Environment for Network Protocols," *Hewlett-Packard Journal*, Vol. 41, no. 1, February 1990, pp. 18-28.

Network Advisor Protocol Analysis: Decodes

The decodes feature of the Network Advisor allows users to traverse from a high-level summary of protocol information to a bit-level interpretation of the protocol data.

by Rona J. Prufer

The decode portion of protocol analysis involves the recognition and interpretation of the syntax and semantics of the different types of network protocols. The HP 4980 Network Advisor is different from traditional protocol analyzers in that it attempts to interpret data from the network under test and provide answers to protocol problems, not just reams of data. Two of the Network Advisor's key features are a flexible user interface and the number of decodes it can handle.

Design and Development Considerations

The considerations associated with designing the decode platform involved deciding how to:

- Present information to the user
- Divide protocol knowledge between the analysis and user interface environments
- Make a contribution to industry-standard decode practice
- Provide a productive environment for decode developers.

Experience from previous products and user feedback answered many of the user presentation issues. The solution to dividing protocol knowledge between environments came from a definition of the division of responsibilities between the protocol analysis environment and the user interface environment. A contribution to decode practice was made by including knowledge of the network protocols and determining and providing information to the user about a network's health. Finally, a productive environment was provided in which developers needed minimal system knowledge, allowing them to focus on protocol-specific issues.

Presenting Information

Presenting information to the user involved understanding the expertise of our potential customers. Experienced network managers know the protocols and most of the significant fields contained in the protocol fields. These users need to see a high-level view of the data and have the ability to focus on the specific problem when they determine that there is a problem. At the other end of the spectrum are novice users who know little about protocol fields but need to have enough information to ensure that the network is

15:11 06/18/92 ©1991 Hewlett-Packard. V1.0

IP Summary Decode

Control | Config | Actions | Format | Other displays | Help

Frame	Time	Source	Destination	Next Protocol	Data Si
39	07:26.994	15.6.72.18	15.6.72.153	Stream	48
!40	07:26.995	15.6.72.18	15.6.72.153	Stream	48
41	07:26.996	15.6.72.18	15.6.72.153	Stream	48
42	07:26.997	15.6.72.18	15.6.72.153	Stream	48
43	07:26.999	15.6.72.18	15.6.72.153	Stream	48
!44	07:27.000	15.6.72.18	15.6.72.153	Stream	48
!45	07:27.001	15.6.72.18	15.6.72.153	Stream	48
46	07:27.002	15.6.72.18	15.6.72.153	Stream	48
47	07:27.003	15.6.72.18	15.6.72.153	Unknown	48
!48	07:27.004	15.6.72.18	15.6.72.153	ICMP	48
49	07:27.005	15.6.72.18	15.6.72.153	IGMP	48
50	07:27.006	15.6.72.18	15.6.72.153	GGP	48
51	07:27.007	15.6.72.18	15.6.72.153	Stream	48
!52	07:27.009	15.6.72.18	15.6.72.153	TCP	48
!53	07:27.010	15.6.72.18	15.6.72.153	UDP	48
54	07:27.011	15.6.72.18	15.6.72.153	Unknown	48
55	07:27.012	15.6.72.18	15.6.72.153	Unknown	48
56	07:27.013	15.6.72.18	15.6.72.153	Unknown	48
57	07:27.014	15.6.72.18	15.6.72.153	Unknown	48
!58	07:27.015	15.6.72.18	15.6.72.153	Stream	48
!59	07:27.016	15.6.72.18	15.6.72.153	Stream	48
60	07:27.018	15.6.72.18	15.6.72.153	Stream	48
61	07:27.019	255.255.255.255	15.6.72.153	Stream	48
!62	07:27.020	15.6.72.18	15.6.72.153	Stream	48
63	07:27.021	15.6.72.18	255.255.255.255	Stream	48

Capture Buffer limits 1 - 119.

Fig. 1. Summary view of an internet protocol (IP) in the DARPA protocol stack.

working. For these two very diverse users we found that there were three views of the protocol decode that would satisfy most requirements: summary view, detail view, and data view.

Summary View. The summary view has several uses of the same format. The summary decode screen has one line per frame showing frame number, time, and three to four other significant fields (depending on the field sizes). This view can be used on an individual protocol or on a protocol stack. For example, Fig. 1 shows the summary decode for the internet protocol (IP) in the DARPA* protocol stack. This summary shows the source and destination addresses in a dot-decimal format, the name of the next protocol layer, and the size of the data being passed to the next protocol layer. The summary view is also useful for seeing traffic on the network. By changing the format slightly, the user can see a network summary that shows the MAC** source and destination addresses along with a top-down list of the protocols contained in that frame (Fig. 2). Another common use of a summary display is to show a stack-specific overview. For instance, the summary for the AppleTalk stack shows the source and destination MAC addresses and the AppleTalk protocols contained in that frame (Fig. 3).

Detail View. The detail view is a full-sized window that shows one packet or protocol per display. The detail-view format has three columns. The first column lists the names of the fields in a packet, the second column contains the current value in the field, and the third column describes the meaning of the field or the value in column two. In Fig. 4, which shows a detail view of an IP packet, the first column shows an ordered list of the fields that are in the packet. The first item on the list shows the version of the IP packet, which according to the second column is 4. The second item shows that the internet header length field of the IP packet has the value 5, which indicates 32-bit words (column three). The precedence field has the value 000... which corresponds to routine precedence (as opposed to an urgent precedence). Following the ordered fields in the protocol is the derived information about the packet. For instance, there may be an indication about how much data a packet is passing up to the next layer, or information about the reassembly process or protocol-following process.*** The detail display can also be used to show the fields of an entire protocol stack. For instance, it can show the Ethernet fields, the IP fields, and the TCP fields together in one display.

Data View. The third view is of the data contained in a packet. Again, the flexibility exists to show all the bytes of the packet or just the bytes associated with a single layer. This display format lets the user see data in a format that may have more meaning. For instance, there may be users who want to see data in EBCDIC or hexadecimal formats (Fig. 5). Another variation shows the entire packet of data with protocol headers separating the different protocol layers.

Different Environments

The decode design is split between the two major functions of the instrument. Displaying strings and values and formatting are handled by the general-purpose environment, and protocol meaning is determined by a module in the analysis and real-time (ART) environment. The general-purpose environment provides mechanisms for handling the Network Advisor's user interface and the ART environment provides services for interfacing to the network and transporting data to and from the general-purpose environment. For greater

* Department of Defense Advanced Research Projects Agency.

** MAC stands for media access control, the lowest level of the protocol stack. Thus MAC frames are the frames actually transmitted on the physical network media.

*** Protocol following is tracing the different states a connection goes through to transfer information.

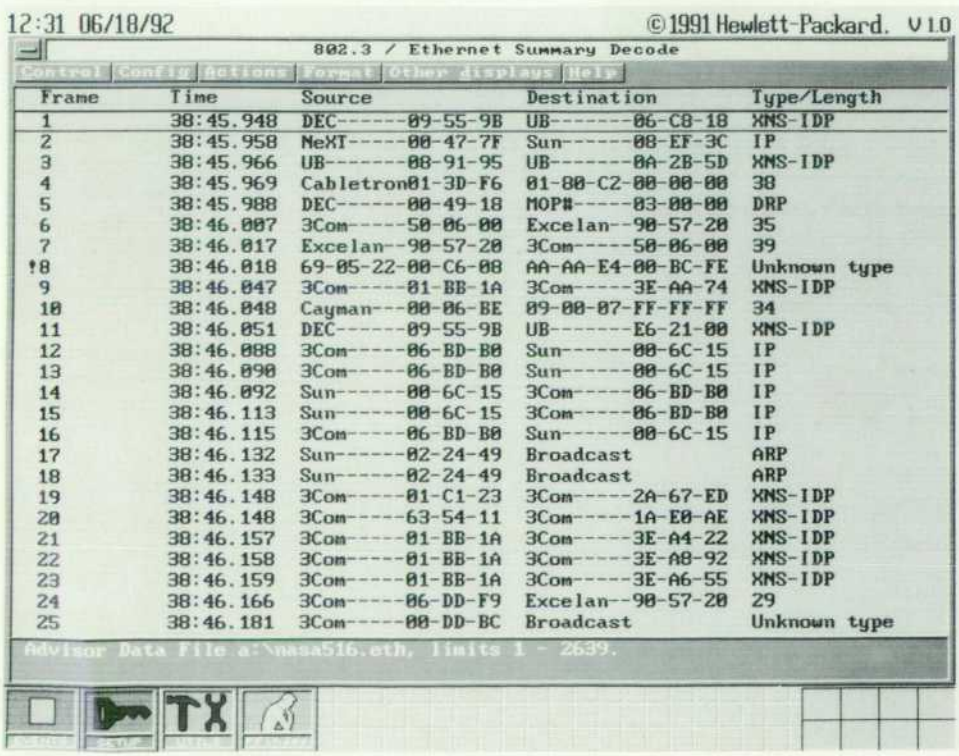


Fig. 2. Summary view showing IEEE 802.3 Ethernet decode.

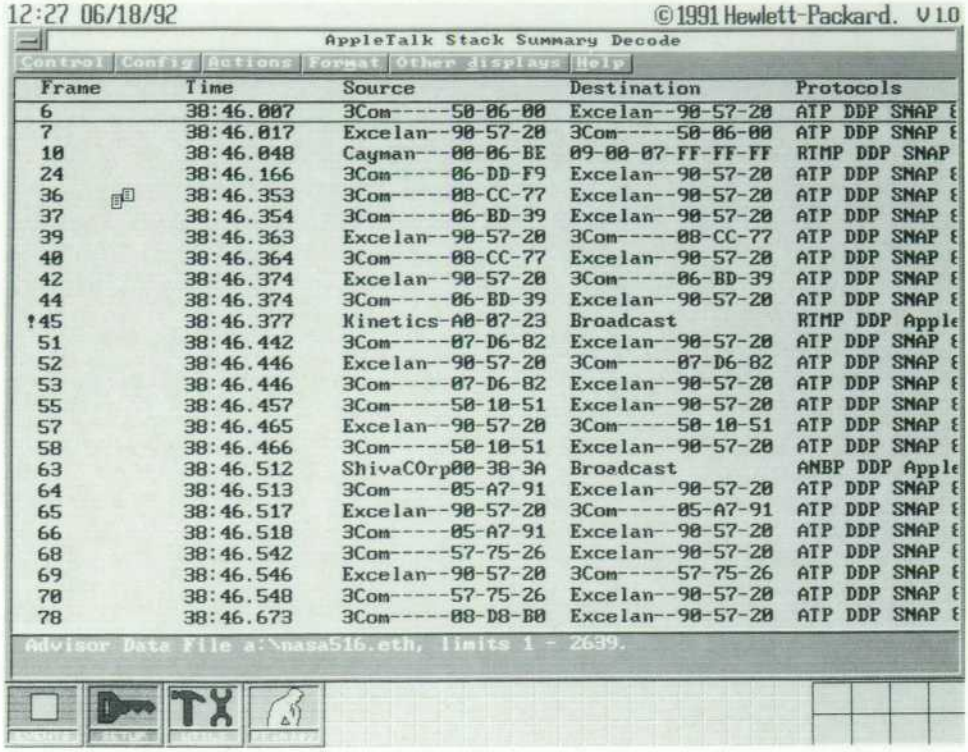


Fig. 3. Summary view for an AppleTalk stack.

flexibility all the incoming network data, after being interpreted and put in a special data structure, is sent from the ART environment to the general-purpose environment. This allows the user to select which format is more useful and not have to wait for information to be processed again in the ART environment. The services provided in the ART environment and the relationship between the ART and general-purpose environments are described in the article on page 29. The user interface is described in the article on page 22.

The data structure that is passed between these two environments is called an *analysis item*. This structure was chosen because it allows many different protocols to be described in one format. An analysis item contains two or more analysis data units (ADUs) as shown in Fig. 6. There is a one-to-one correspondence between an ADU and a protocol decode.

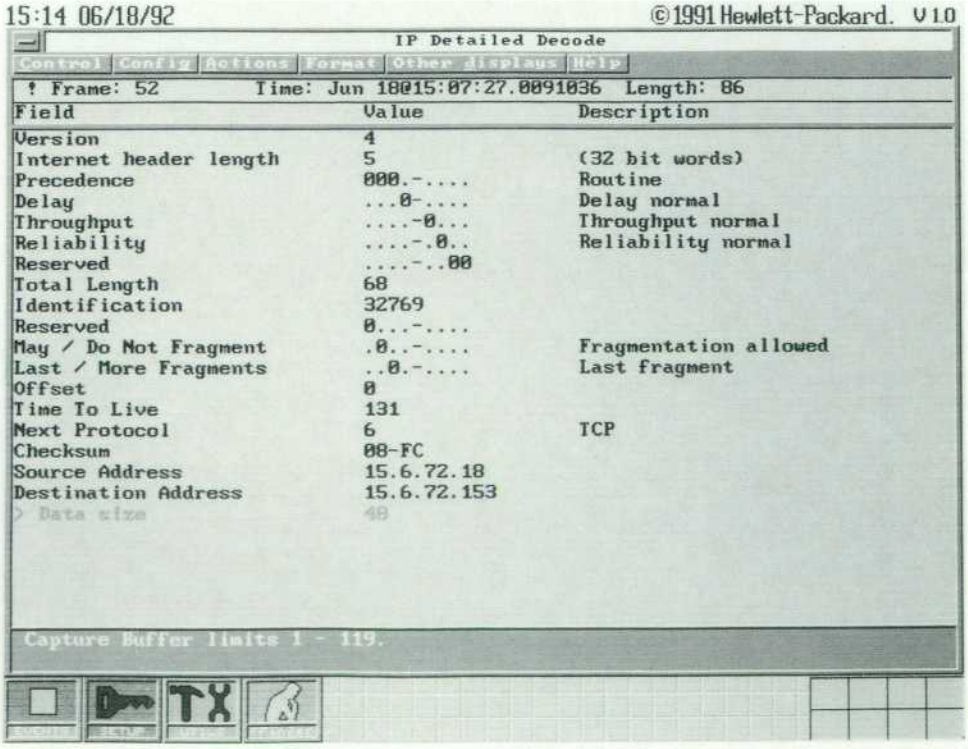


Fig. 4. Detail view of an IP packet.

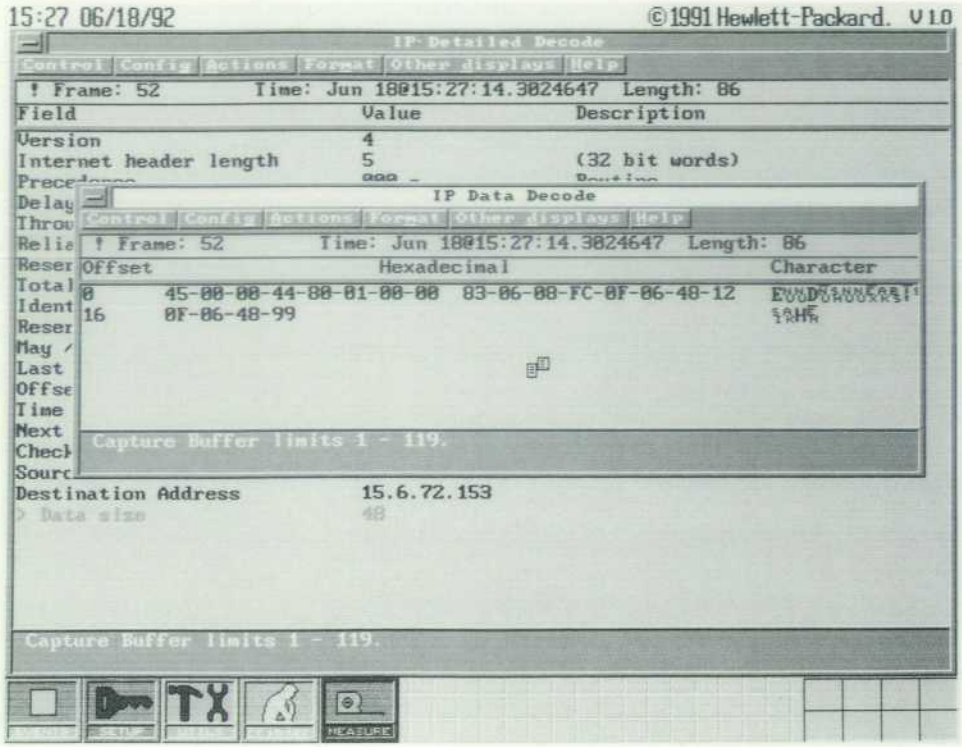


Fig. 5. Data view in hexadecimal format.

Each ADU contains three sections. The first section is data for the associated protocol layer; it is referred to as the hex. The second section is the syntax for the fields in the protocol. For instance, each field in the protocol has a unique syntax record associated with it. The last section is a semantics section, which contains additional information derived from analysis of the protocol.

The syntax section is further divided into a series of records. These records are patterned after the ASN.1 encoding of a tag, a length, and a value field.[1,2] Each field in the protocol is assigned a unique number (the tag). In cases where several protocols share the same field, such as a destination address, this tag is shared so that it is always called Destination Address in the display of that field. The second field in the ADU is an offset field. It indicates where the bytes or bits exist in the hexadecimal field of the protocol. If the most-significant bit is 1 then the field is interpreted to be the number of bits, and if the first bit is a 0, it is the offset in bytes. The third field is the length of the protocol field. Again, the most-significant bit indicates a bit or byte length. The last field is called the status/error/warning field. It is one of the two major contributions of the decodes to the Network Advisor. It tells the user more information about a specific field.

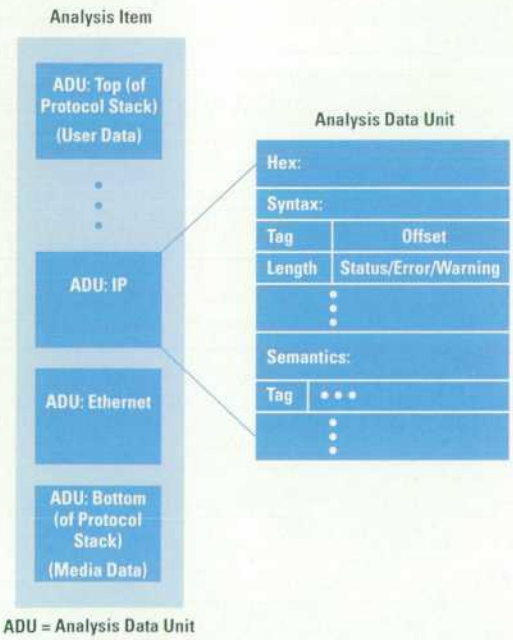


Fig. 6. An analysis item data structure.

The semantics section of the ADU has a more free-form format. It has a unique tag that describes the information and the length of that information. Like all information passed between the two environments, the data in the semantic section must adhere to the long word* boundary rules. Examples of the information passed in the semantics section include reassembly information, resequencing information, and protocol-following information.

Decode Development in the ART Environment

The object-oriented C++ design of the ART environment gives a decode designer four classes to work with. These include the data analysis class, the module record registration class, the path entry class, and the path SAP** entry class. A GNU-EMACS editor function fills in the class names and formats a module for simple default class protocol when a protocol module is first checked out for design. This provides the decode designer with a foundation on which to add the features of a specific protocol.

Data Analysis Class. The data analysis class contains a function called makeAdu that allocates and formats the ADU for a

* A long word is a four-byte word.

** A SAP (service access point) is an addressable point at which protocol services are provided for the next-higher protocol layer.

particular protocol. It allocates memory space for the hex, syntax, and semantic sections of the ADU. In most protocols, the fields are known and do not change. Only in the case of options or dependent fields are there extra fields that need to be considered. Because of the large set of known fields, space can be allocated and field values (tag, offset length, and status/error/warning) can be defaulted. When the makeAdu routine is executing, it moves through each field in the frame data and analyzes it for significant facts. For instance, in the IP protocol, the version number is checked. If the version is not 4, the status/error/warning field of the version syntax record shows a number signifying a bad version. Another example is the IP internet header length. If this value is longer than the bytes available, an error is signaled in the status/error/warning field of the internet header length syntax record.

The status/error/warning check can be as elaborate as the designer wants to make it. In some protocols the difference between a good value and an unacceptable value is not straightforward. In other protocols there are no fields that are open to interpretation. For instance, in Novell's NetWare Core Protocol (NCP) the completion code values are listed as having five valid values. An update to this specification for future releases of NCP may add more values, so our decode cannot definitively call invalid any other value used for a completion code.

By using object inheritance, different features can be added to a decode in the semantic section of the ADU. The developer can inherit several functions into the data analysis class for conveying additional protocol information to the user. An example is the dataSize record. This record indicates how much information is being passed up to the next layer in a protocol stack from the current layer. Another example is the calculation of a correct checksum. If the protocol has a bad checksum, it needs to be displayed along with the correct value. This information can also be sent in a semantic record.

Module Record Registration Class. The module record registration class allows the decode to be called from almost any place in the decode process, allowing a protocol to be encapsulated. This situation is common in the case of the DARPA internet protocol. This IP can be called directly over Ethernet by putting the type field value of 0x0800 in the thirteenth and fourteenth bytes of the Ethernet header. It can also be called from the SNAP protocol which is above the IEEE 802.3 and 802.2 protocols. The SNAP protocol maps the field value 0x0800 to IP just as the type field of Ethernet does. Both of these protocol combinations call the same IP decode module.

The module record registration class allows the decode writer to specify the decode modules called from a particular protocol. For example, in the IP module record registration class, a value of 5 in the next protocol field maps to the system calling the transport control protocol. Another example is 17, mapping to the user datagram protocol from the IP module record registration class.

Path Entry Class. The path entry class joins the ADU items together to form a chain of ADUs collected into one analysis item. The data passed in the analysis item might contain an Ethernet ADU, an IP ADU, a transport layer ADU, and perhaps a file transfer protocol ADU. In addition, standard top and bottom ADUs are added to the package to take care of user data such as a file transfer (the top ADU) and media information (the bottom ADU).

In addition to formatting the information sent to the general-purpose environment and eventually the display, the path entry class also takes care of forming the paths in the ART environment. These paths uniquely define address-specific mappings. For example, consider a protocol stack in which MAC address A is talking to MAC address B and the IP address for A is C and the IP address for B is D (see Fig. 7). In this stack C will be above A and D above B. The path entry class will tie A to B, A to C, B to D, and C to D so that one side of a protocol conversation can always see the state of the other side.

Path SAP Entry Class. The path SAP entry class allows connection-oriented protocols to keep track of the two sides of a transaction and to store additional information in a common area. A scratch space can be allocated to keep track of the state of the connections, the sequence numbers, and any other information that would help follow a connection and share information between the two sides of the conversation.

A strict layered approach to protocol analysis was chosen for the decode development model because it allows dynamic allocation of protocol analysis decodes and frees the developer from the need to depend on knowledge about any other protocol above or below a protocol module. For instance, the IP module knows that several different protocols can be sent information via the next protocol field. However, because of the layered approach in the ART environment, the IP module does not have to know that any of these protocol decodes actually exists. The module will send the data to the next layer, and if there is no decode for the data, the data automatically gets put into the top protocol module's hex field in the ADU. This indicates that there are no more consumers for the data and signals the ART environment that an analysis item is complete and can be sent to the general-purpose environment for display. The IP module is then ready to process the next frame.

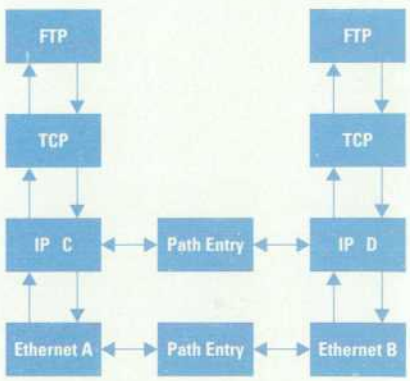


Fig. 7. Connections formed between ADUs by the path entry class to show connections within and between nodes.

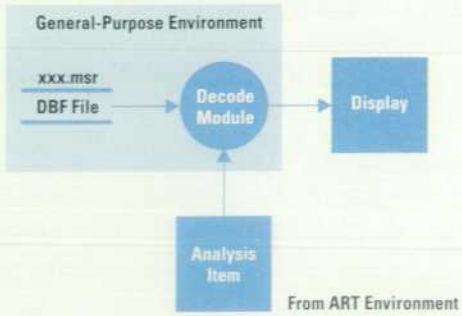


Fig. 8. Components and data flows involved in decoding and displaying data in the general-purpose environment.

Using a prototype template for these four classes and two very diverse protocols as the first ones to be developed in this environment, we were able to test some of the decodes. As more and more decodes were developed, improvements to the templates were made. For instance, a function call was added to set a bit indicating to the general-purpose environment that a frame contains an error. Also, subclasses were developed that are useful to different protocol stacks. An example of this is a class for handling reassembly. This class is used in IP and the ISO network layer decode.

To test decodes in the ART environment, the input to a decode was simulated using tools that take an input file describing the scenario to be tested and provide a stream input to test the ART code on an HP-UX* operating system. This means that all the test cases a protocol can expect to see can be put into a file and tested as the code is developed.

The ART environment is designed so that all protocol analysis can be done once in one place. This design allows the display modules to be concerned only with the presentation of data and not with any protocol-specific knowledge.

Decodes in the General-Purpose Environment

The decode modules in the general-purpose environment were designed around the three main methods of displaying information described earlier. These three displays allow the user to traverse from a high-level summary of information to the lowest bit-level interpretation. Since they are relieved of any protocol analysis duties, the general-purpose decode modules concentrate on taking information from the analysis items from the ART environment and the format information from a database to provide the optimal display of information to the user (see Fig. 8).

Like the ART environment, the general-purpose environment was designed using an object-oriented approach. Because the ADUs consist of the same three sections, there is generic code in the general-purpose environment that parses each of the three sections. These parsing modules are optimized depending on the what the user is viewing. For instance, if the user is looking at a summary view with only one stack open, the parser can traverse the analysis item pulling from each ADU only the information needed for a summary view. This optimization leads to a significant increase in the performance of the display system.

For the decode developer, display format information is input via a database file written in DBF (database format). This ASCII file, which is usually written near the end of the decode development cycle, contains information about mapping numbers contained in the ADU to strings displayed in the decode windows. This file is parsed into Smalltalk Collection and Dictionary classes, which are used at run time in the general-purpose environment. This implementation has two very important advantages to the decode developer. First, no knowledge of Smalltalk is required to develop the decode modules. One less environment to learn helps to reduce the nonprotocol development time and fully uses the protocol analysis skills of the protocol expert. The second advantage is that the decode developers are not required to do run-time debugging of the decode modules. This drastically reduces the variety of bugs that occupy the protocol expert's time.

The DBF file format has several sections. The first section is setup information. This includes the name of the decode (e.g., IP), the network media (e.g., IEEE 802.3), and the names of the online help files that can be referenced.

The second section in the DBF file contains data for mapping the constants sent in the ADUs to the strings that appear in the different protocol views. This mapping information is split into four sections: mnemonics, errors, warnings, and status messages. There are some strings common to all these sections. For instance, if there is a checksum error, one common string to indicate this condition is included in the system errors file. In addition to mapping strings, the mnemonics section also has a provision for declaring the format in which to display the value. Some common display formats are convertToDecimal, convertToBinary, convertToDotDecimal, convertToHex, and convertToAddress. With the convertToAddress format, user names are mapped to addresses and put into the database in place of network addresses.

A significant contribution is the ability to display the protocol fields in a manner that makes sense for each field. For instance, in the IP protocol the first field is the version number. It is four bits long, and instead of showing the user the four bits, it is displayed in decimal as the specification is written (a 4 is easier to read than 0100). Another example of this is also in the IP protocol. The precedence field is in the second byte of the protocol and is three bits long. To a user who is not familiar with this field, the values mean very little. However, to a Network Advisor user, the value column shows that the field is three bits long and specifies the three bits. In addition to this, the third column gives a high-level interpretation of these bits (Routine precedence in Fig. 4).

The next section in the DBF file contains the strings for column headers for the summary, detail, and data view windows.

When the fields in the DBF have been filled in, the DBF file is then put through a parse routine. This compilation is quite robust in terms of catching syntax errors and typing mistakes. The result is a file of type xxx.msr, which is a measurement file used at run time by the decode modules in the general-purpose environment to display the decode information.

The first pass through the development of the decodes in the general-purpose environment revealed many areas for optimization. Because the input to the decode modules is an ASCII file and the modules are developed with an object-oriented language, a second pass through the system was

done late in the development cycle to find the common elements among the protocols. For example, a number is now used to represent common field names, errors, and warnings. Also, the detail display uses every syntax record so there is no need to specify its format in the ASCII file, and many of the same semantic records are used by protocols so a standard display method was developed for these records.

The same testing approach used in the ART environment was used in the general-purpose environment. Files of ADUs (or analysis items) were generated in the ART environment and transferred to a PC running the general-purpose environment. These files were then fed to the decode window as if it were running from data stored in the capture buffer in the ART environment. All of the test cases used to test the ART code were used to test the general-purpose portion of the product. In addition to testing in separate environments, test files were sent to an HP 4972A protocol analyzer and put onto the IEEE 802.3 media for live capture by the hardware. The results from testing with the HP 4972A were compared with results from previous tests to verify consistency of results and regression testing.

The Results

The general-purpose development effort for decodes was reduced to less than 20% of the total decode development time because we used protocol templates, eliminated the need for our protocol expert to develop in Smalltalk, and used a robust parser that caught many of the syntax errors in the DBF file. Debugging time was also reduced because the general-purpose decode files are one step removed from the real-time processes.

By dividing the decode implementation into two environments and identifying conventions between common protocol decode tasks, the development time for new protocol decodes was significantly reduced. All major protocol stacks have been decoded and the embedded protocols have been accounted for automatically. There is a great deal of code reuse between different protocol stacks because of the inherited functionality provided by using object-oriented environments.

There are a few exceptions in which the strict vertical communication between protocols had to be subverted. For instance, in the SNAP decode, if the next protocol field indicates AppleTalk protocols, the lowest layer is examined to see if it is a token ring network or Ethernet. A modification is then made in the ADU to send this to either a TokenTalk or an EtherTalk decode based on the lowest layer. Another example is in the Novell protocol stack in which a reply frame contains a frame number and a packet type for the corresponding request frame.

Acknowledgments

The author would like to specifically thank Jim Hammond for his unending customer support and Bob Vixie for his help with getting the ART environment working. Special thanks also go to Gary Roberson and Bob McCabe for their review of this article.

References

1. *Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)*, ISO 8824: 1987(E).

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

Mechanical Design of the HP 4980 Network Advisor

The package design for the Network Advisor was guided by the electrical, mechanical, and ergonomic requirements of a PC-based protocol analyzer.

by Kenneth R. Krebs

The HP 4980 Network Advisor package consists of 31 injection-molded parts, 15 sheet-metal parts, 19 cables, nine printed circuit boards, two custom-machined parts, a custom power supply, flexible and hard disk drives, a color or monochrome LCD (liquid-crystal display), and numerous other custom and standard parts. Its hinged, fold-up, flat-panel display and fold-down keyboard are designed to make it easy to use the Network Advisor either on a desktop (see Fig. 1 on page 6) or in a floor-standing position (see Fig. 1), while providing maximum portability when closed. It has interchangeable network interface modules that mount to the underside of the instrument. The overall package measures 5.9 inches high by 14.3 inches wide by 16.8 inches deep and weighs 25 pounds fully loaded.

Fig. 1. The Network Advisor in a floor-standing position.

Design Decisions

Mechanical design for the Network Advisor was driven by several major decisions made very early in the product definition phase of the project. The first of these was to make the instrument DOS-compatible and, therefore, PC-based. Because of the dominance of the DOS operating system in the LAN testing market, our customers demanded DOS compatibility in our products.

While we did not intend to market the Network Advisor as a PC that does protocol analysis, but rather as a protocol analyzer with an embedded PC, we did feel that the ability to run standard PC applications (e.g., word processors and spreadsheets) would be a marketing benefit. Therefore, we needed a full-screen, 80-column display. Since the VGA standard was emerging as the choice of the future, we chose it for our instrument. A second result of the DOS decision was the requirement for a full-function, full-size PC keyboard and internal flexible and hard disk drives.

A second decision (a result of the first design decision) was the need to be able to install at least one and preferably two standard, off-the-shelf, full-length, low-profile PC cards.

The third major decision was the choice of a flat-panel technology over a CRT. The VGA decision dictated a CRT too bulky and heavy to meet our portability requirements. Also, CRTs have some manufacturing disadvantages we wished to avoid (e.g., pincushioning, alignment, high voltages, and shielding). We also felt that the market perception of flat panels as a leading-edge technology would be beneficial.

We investigated several flat-panel technologies including electroluminescent, gas plasma, and liquid-crystal. Electroluminescent and plasma displays were costly, had high power dissipation and lacked sufficient grayscale shades. After investigating several types of LCD, we chose a cold-cathode, backlighted, film-compensated LCD as having the best combination of brightness, contrast ratio, cost, and weight. Just before our tooling release, Sharp Inc. introduced a 10.4-inch-diagonal, TFT (thin-film transistor) active matrix, color LCD, which is larger and thicker than the monochrome LCD we had chosen. After a redesign effort to accommodate the larger display, the display housing injection mold was designed with inserts to allow for both color and monochrome versions.

Because there are several different networking technologies (e.g., token ring network, Ethernet, fiber distributed data interface (FDDI)), our instruments need different hardware sets for data acquisition and analysis and different external connector types for connection to the network. In the past we accomplished this by offering different interface modules (pods) cabled to the base instrument (mainframe). Considering how to handle different network technologies led to our fourth major design decision which was to integrate the pods into the mainframe so that nothing external would be required or would hang off the instrument and get in the way during operation. These integral pods needed to be easy to install and remove. The difference in networking technologies also required that the network line and event status LEDs on the front panel (up to 12 pairs with each pair consisting of one red and one green LED) be easy to relabel since different network types have different numbers and types of lines and different nomenclatures.

Another important design constraint was the requirement that the user be able to operate the instrument conveniently on a desktop and on the floor. Frequently our customers need to make their network connections in a small control room, a closet, or around the back of a patch panel where tabletop space is unavailable.

Since many of our customers are third-party network service providers who travel with the instrument to their customers' sites, the instrument had to be truly portable and rugged. This meant that it had to fit under an airline seat and weigh less than 33 lb (we set a target of 20 lb and achieved 25 lb with a fully configured instrument). This also meant that we needed a carrying case not only for the instrument but also for appurtenances such as interface cables and different interface modules.

Many of our customers are network managers who do not routinely move their instruments from place to place. Therefore, we wanted a configuration that would work well with an external color monitor, preferably one that could support the monitor on top of it so as to use as little desktop space as possible.

Lastly, we wanted a package that made a true contribution to manufacturability by being very easy to assemble and service. Therefore, we wanted to eliminate as many fasteners

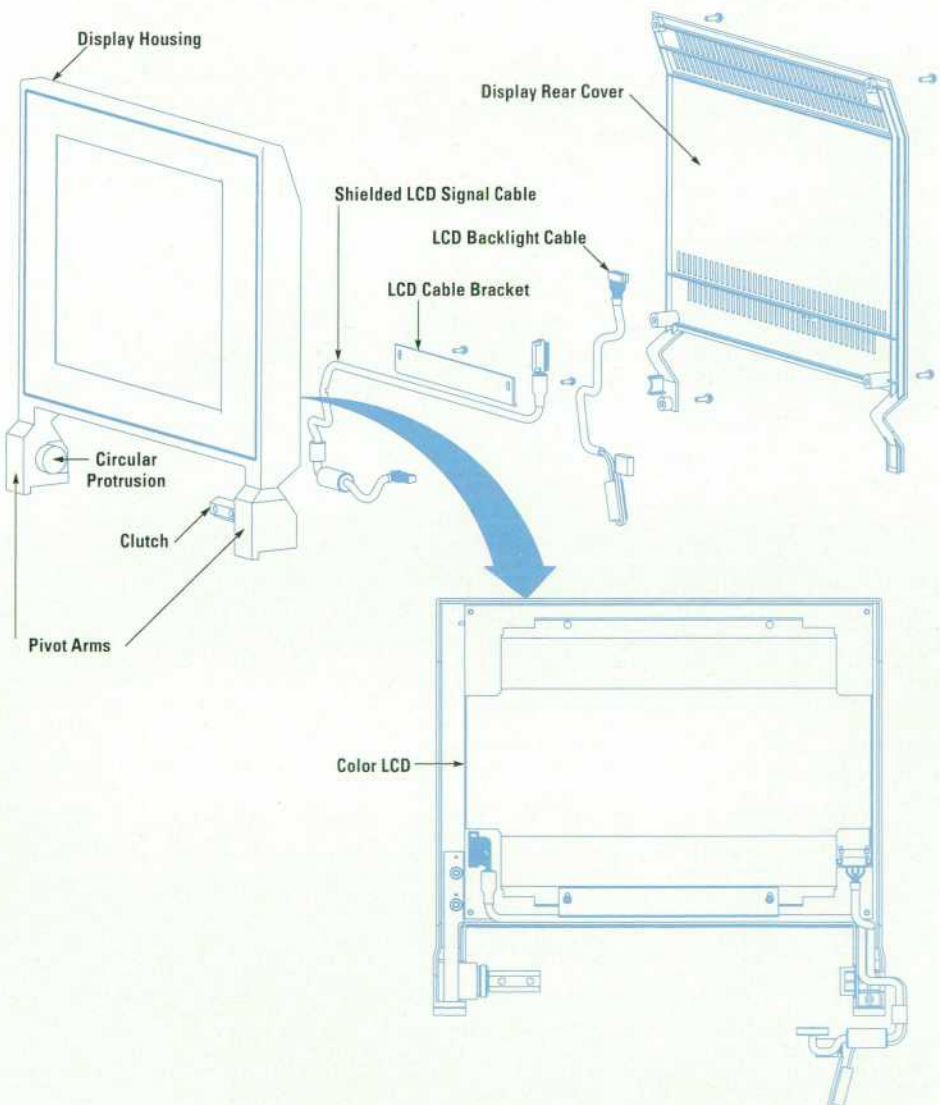


Fig. 2. Display housing and other components for the color LCD panel.

and small parts as we could. We also wanted this package to be as flexible as possible for maximum longevity and reuse.

Design Concept

The design decisions described above resulted in the following implementations for the Network Advisor package.

Display Housing. The choice of a VGA flat-panel display dictated a design concept in which the display rotated or otherwise moved into viewing position. To make it stationary on the front panel would have made the instrument too large or required an upright package similar to the HP Integral portable scientific computer—a design ill-suited for floor operation or for use with an external monitor. Rotating the display up and back into position as is done with a laptop was equally unsuitable. Rotating the display up and forward puts it in good viewing position above the keyboard for desktop operation, and rotating it a bit farther puts it in good position for floor-standing operation with the instrument standing on its rear feet. The display housing unit has two pivot arms at its bottom corners (see Fig. 2). The right pivot arm carries the friction clutch that holds the display in any position. The left pivot arm routes and protects the display signal and backlight cables. In addition, the left pivot arm has a short, molded circular protrusion which acts as a shaft. This shaft gets captured between semicircular features molded into the front panel and a front-panel rear cover piece which, when the pieces are assembled together, form a bearing (see Fig. 3). This bearing captures the shaft for rotational support of the left side of the display housing unit. In its folded-away position, the display resides in a recess molded into the plastic top chassis.

Even in its folded-away position, the front of the display is exposed, and a cover piece is required to protect the display during travel. This protective lid has side latches that snap onto the top chassis. The side latches are pressed inward to release the lid, whereupon it is rotated up and back (see Fig. 4). After the display is raised, the lid is again closed and latched. The lid also acts as the platform to support an external monitor when the display is left folded away (a user must software-select which display is on at any one time to keep the internal display from overheating when closed). A large, polycarbonate label covers the underside of this lid to hide the extensive ribbing needed for stiffness.

Fig. 4. The protective lid used to protect the display during travel. Also shown is the location of the detachable pod assembly which snaps into the underside of the mainframe of the Network Advisor. The pod assembly is discussed later in this article.

Keyboard. The full-size keyboard is hinged onto the front panel where, in its closed and latched position, it protects itself and the front panel. It unlatches and rotates into position on two custom shoulder screws. Nylon-inserted lock nuts captured in each end of the plastic keyboard housings keep the shoulder screws from backing out against the keyboard rotation. A crescent spring washer sandwiched between a flat washer (to protect the plastic from galling) and the underside of each shoulder screw head provides friction

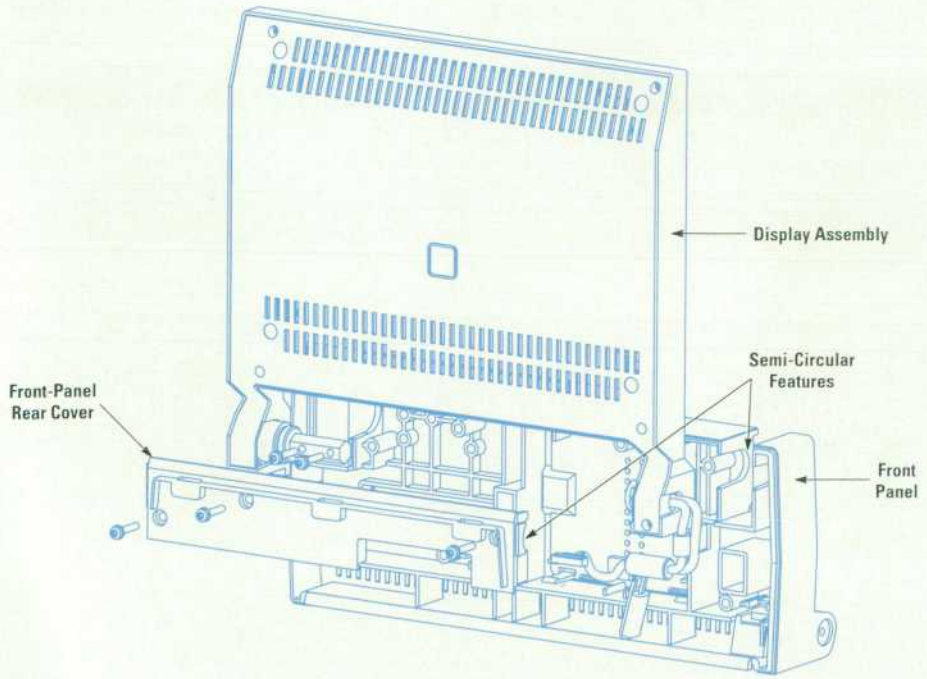


Fig. 3. Front panel and front-panel rear cover showing the semicircular features molded into both parts, which form a bearing for holding the display housing in place.

to hold the keyboard in any position for typing (see Fig. 5). Normally the keyboard is simply rotated until it rests on the desktop at a seven-degree angle for comfortable typing. When the instrument is standing vertically on its rear feet for floor operation, the keyboard rotates around to a stop so that it is in a comfortable position for typing while sitting in front of the instrument (see Fig. 1). While not easily detachable, the keyboard is removable in the event that a user wishes to use a standard 101-key keyboard instead of the one provided. An adapter cable is provided for this purpose.

More than any other feature, the keyboard, folded in a vertical position when closed, determined the height of the instrument. The keyboard and the need for a full-length PC card determined the width of the instrument. The overall length was determined by the components inside the instrument, with consideration given to how tall the package stood in its floor-standing mode.

Top and Bottom Chassis. The bottom chassis acts as the foundation on which all other subassemblies rest (see Fig. 6a). It measures 3.2 inches high by 14 inches wide by 14.2 inches long and is molded in a 600-ton press. To reduce the number of screws, many molded features act either as snap-fit receptacles or anchors for other parts and assemblies. Major assemblies such as the front panel, handle, and rear panel drop vertically into molded grooves and are held in place until the installation of the top chassis captures and anchors them by the addition of four screws. The top chassis (Fig. 6b) is roughly the same size as the bottom and is molded in the same press. Tabs on the top chassis mate to slots in the front panel and front-panel rear cover to anchor that assembly and provide good seam contact for EMI suppression.

Pods. The interchangeable interface modules (pods) house the data acquisition and analysis hardware and the connectors for the physical interface to the network. A plastic molded housing forms the bottom section of the pod container. A pair of painted and screened sheet-metal covers are lap-joined together using flathead screws, countersunk punched holes in one piece, and press-in floating fasteners in the other to form the top section of the pod container (see Fig. 7). This was done to compensate for assembly tolerance stackup and to allow for easy assembly of different network connector types and combinations. The printed circuit boards mount to each other via board-to-board connectors and snap-mount, press-in standoffs. The printed circuit board subassembly then snap-mounts to press-in standoffs in

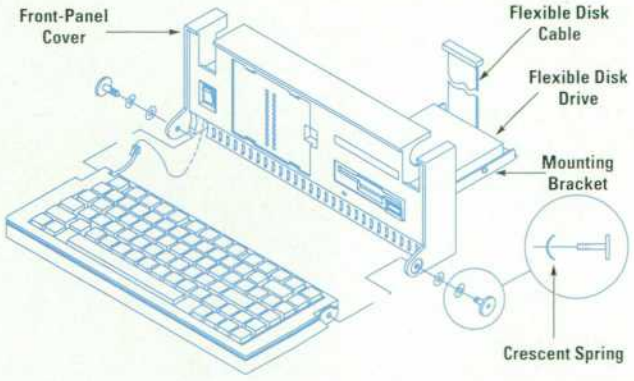
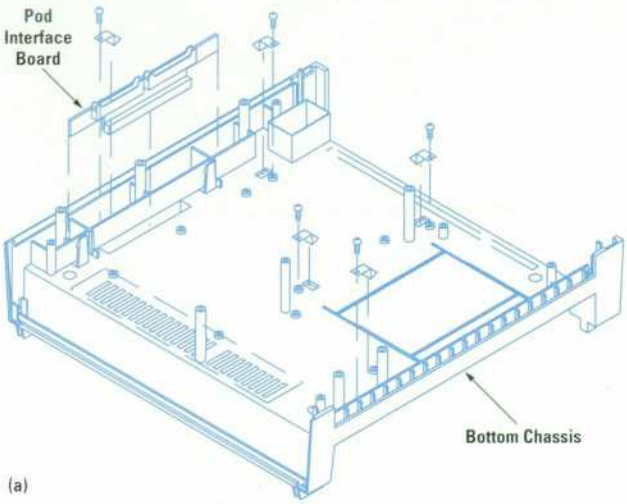


Fig. 5. Keyboard connection to the front panel.



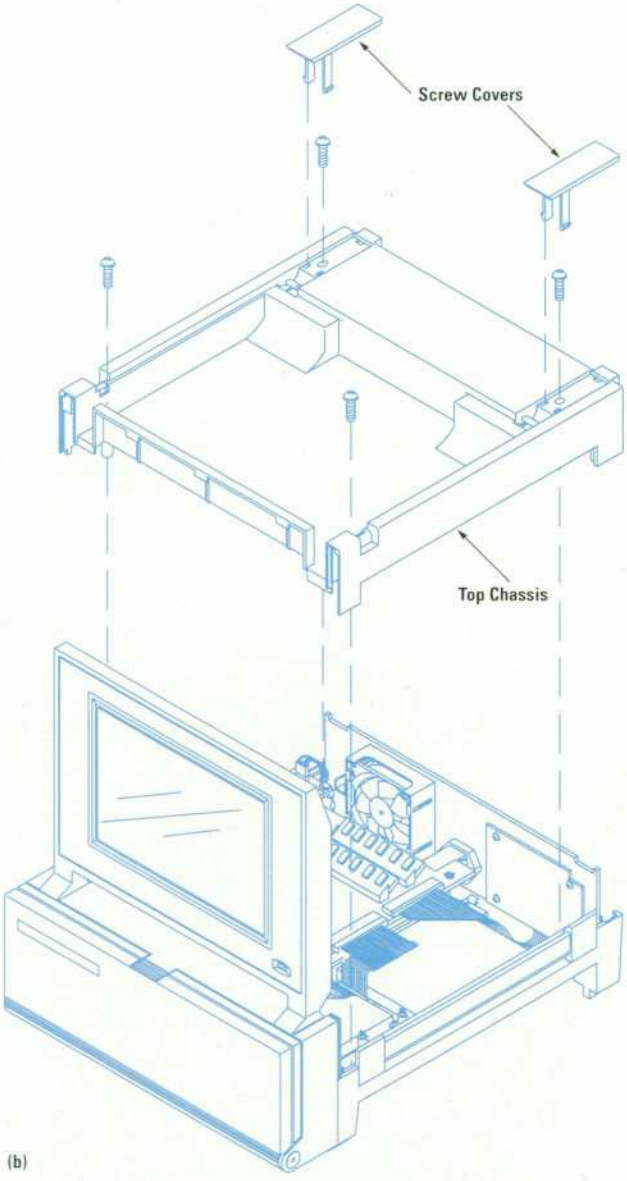


Fig. 6. (a) The bottom chassis of the Network Advisor. (b) The top chassis of the Network Advisor.

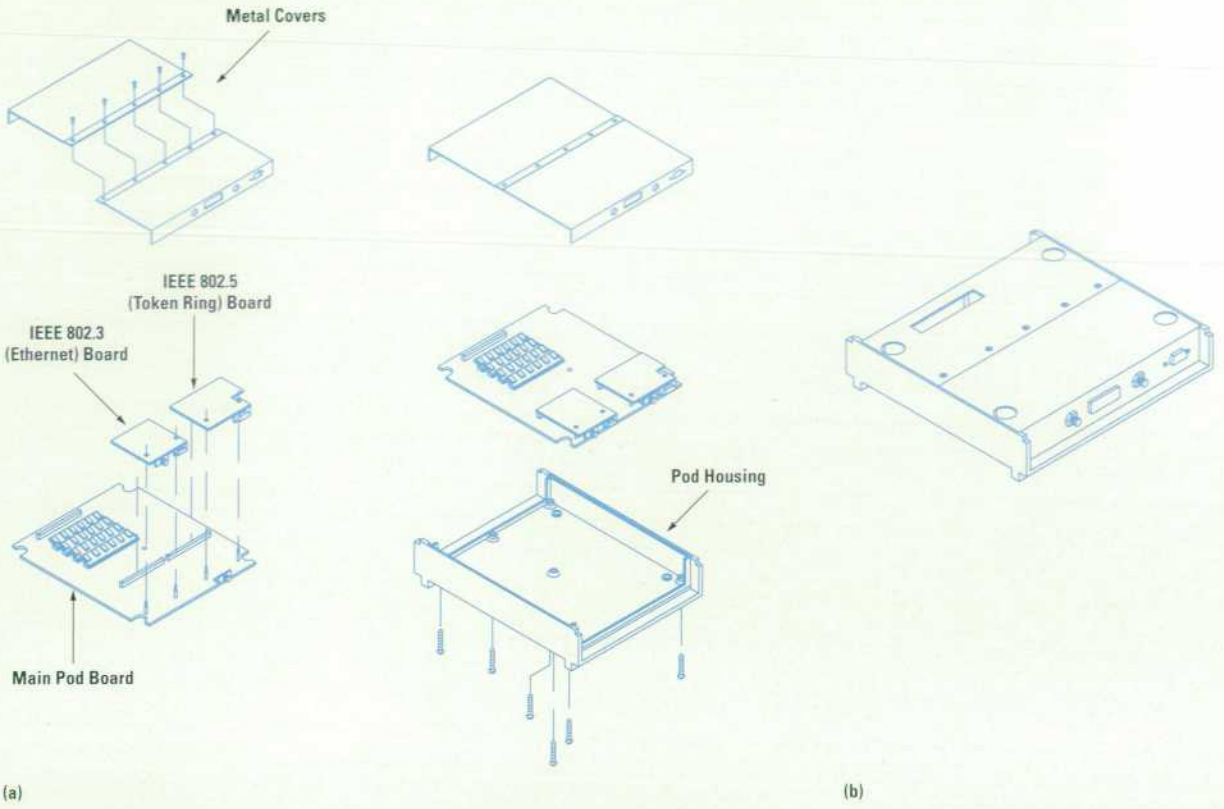


Fig. 7. Pod assembly. (a) The three main components of the pod and how they go together. (b) The assembled pod.

the sheet-metal covers. The whole assembly then screws to the plastic pod housing.

An assembled pod mounts on the underside of the mainframe (see Fig. 4) using four quarter-turn fasteners. The quarter-turn studs are captured, but float freely in the pod housing and mate to receptacles that are ultrasonically installed in the bottom chassis. All of the standard PC signals are bused from the mainframe to the pod through a 140-pin connector pair which automatically mate during pod installation. The pod cannot be installed incorrectly. The network interface connectors are mounted along the side of the pod, which puts them in easy reach for connection to the network, especially in floor-standing operation. The exposed sheet metal front panels of the pod through which these connectors protrude are recessed to keep the connectors protected. However, this configuration reduced the internal width available in the pod for printed circuit boards. A full-length, standard PC card was too long to fit in this width. Therefore, two plastic molded end caps were tooled to allow for the inclusion of the PC card. In this case, the sheet metal, no longer exposed, is reduced to a single, flat, unpainted piece which helps retain the end caps and close up the pod. The pod can accept either one full-length and one 3/4-length, low-profile PC card, or our custom LAN data acquisition boards, but not both simultaneously. Because we offer some products in the form of PC cards for use in customers' existing network computers, these cards can also be put into pods for use in the Network Advisor. One example of this is the HP 4957PC wide area network analyzer, which is on a PC card. With this feature, a customer can install an HP 4957PC into a pod to make the Network Advisor into a WAN analyzer.

Bottom Feet. Along the bottom of the pod housing are molded two long rails that "join" the front and rear bottom feet molded into the bottom chassis. These rails are 0.5 mm shorter in height than the chassis bottom feet. This ensures that while the bottom feet are always the ones to bear the weight of the instrument, they do not catch the back of a user's leg when the instrument is carried by its padded, flexible side handle.

Rear Panel. The molded rear panel mounts the line filter/fuse module, the fan, and the external PC ports printed circuit board, which provides two serial ports, one parallel port, and one video port (see Fig. 8). The area behind which the PC ports board mounts is formed using mold inserts to allow easy changes in the number and configuration of connectors (e.g., adding HP-IB or SCSI). A fan grill and cage are molded into the panel so that the fan can be snapped into the panel without the use of fasteners. A molded-in feature acts to capture the RAM backup battery in the bottom chassis when the rear panel is installed. All text and graphics (connector labels, line information, safety warnings) are molded-in on mold inserts to make them easy to change or repair. Also, text is indented in the plastic (raised on the inserts) and molded into shallow recesses to accept labels to cover the text if it becomes necessary (e.g., labels for different language versions). A long label was added to accept new warning and regulatory messages as they become necessary.

Tooling and Molding

HP's Palo Alto Fabrication Center (PAFC) molds most of the Network Advisor's plastic parts. Mobay's Bayblend FR1441 polycarbonate/ABS blend was chosen as the basic material

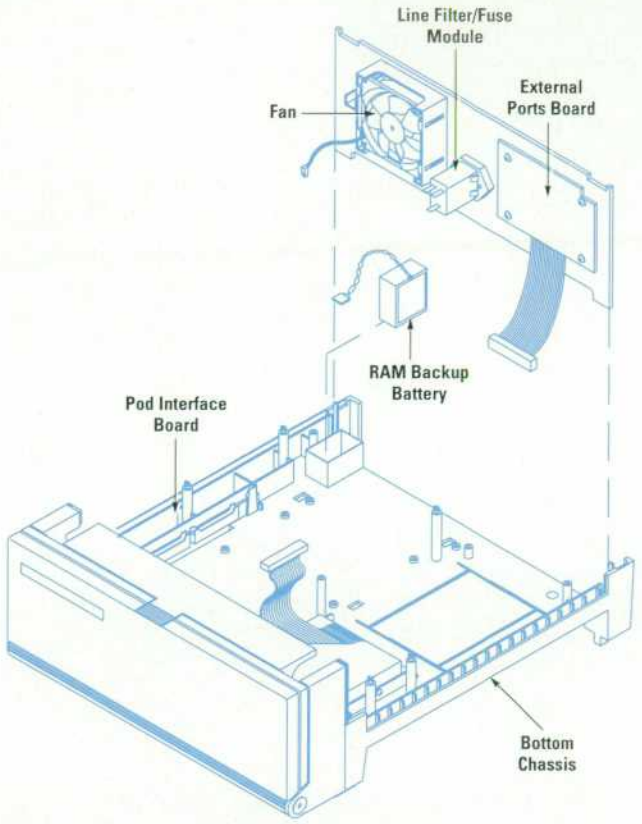


Fig. 8. Components attached to the rear panel of the Network Advisor.

for its combination of strength, moldability, appearance, and price. Two parts, the display cover lid and the display housing rear cover (both large, thin parts), required FR1439, which has a higher percentage of ABS, for increased moldability to prevent warping and to reduce blush. Two small parts (molded by a second vendor), the snap-on side feet and the snap-on rear mounting screw covers, use GE's Lexan 920 straight polycarbonate. While we feared a noticeable gloss difference between the blended and straight materials, our fears were not realized and the match has been good. The snap-on display lid latches also use this material for strength reasons. They have a molded-in cantilever spring that provides the force to return them to their home positions after unlatching. Straight polycarbonate with its higher allowable strain rates and creep resistance was needed for these springs. The friction clutch arm bears the brunt of the display housing support and torque when moving the display. For this reason, it is molded in a 40% glass fiber filled polycarbonate for strength and stiffness. The keycaps are molded in polyester to allow sublimation printing of the three keycap legend colors.

All external cosmetic surfaces are textured using an HP frosted grain III specified as 0.001 inch deep on typically 2.5° draft.* Some surfaces, however, were designed with only 1.5° draft. On these surfaces, the texture is specified as 0.0005 inch deep to avoid possible ejection problems.

* A draft is a slight angling of the vertical walls of an injection mold to allow the molded part to separate from the mold.

As mentioned previously, the display housing and display housing rear-cover molds were designed with two sets of inserts to accommodate the color or monochrome LCD display. The color LCD has the larger active area opening and requires different mounting boss height and location.

The handle-mounting piece and the front-panel rear cover piece are about the same size and weight. As a result, they were put into a family mold to save on both mold cost and part cost. This later had the disadvantage of changing two parts when only one, the handle mount, needed its thickness reduced. Additional work was then needed on the front-panel rear cover to compensate for the reduction in the mold separation line.

During the lab prototyping phase, we soft-tooled the entire box using urethane molds for parts over 40 square inches and aluminum tools for the smaller parts (a process limitation). The vendor worked directly from IGES** translations of our 2D, undimensioned HP ME 10 drawings. The parts had molded-in color and texture but were soft and capable only of limited structural and temperature testing. The smaller parts, molded in aluminum tools, were molded in their proper materials, which allowed thorough structural testing of these parts. While this process produced high-quality parts that were very useful in our evaluation, the process was costly and took much longer than the vendor estimated.

With an all-plastic enclosure, we knew that shielding technology would become a critical factor. From the outset, we planned on vacuum-deposited aluminum for this job. This technology was available in very few places. Also, the size of some of our parts (requiring up to 600-ton presses) limited the molder selection greatly. However, one molder had up to 1000-ton presses and the ability to do vacuum deposition in-house. This vendor selection later proved to be unfortunate for two major reasons. First, they relied heavily on cutting cavities in the parent steel. Many of our parts have deep ribs and intricate features which, later experience would show, would have been more easily polished and otherwise tuned had they been formed using mold inserts. While more costly up front, inserted molds would have saved us time and money during the tool tryout and tuning phase of our product development. In some cases, attempts at changing the parent steel, especially polishing of deep ribs, caused mold damage that could not be fixed, leaving some parts permanently out of specification. In these cases, unfortunate and costly design changes were necessary.

The second reason was that we found out very late in the tool tryout phase that the molder could no longer supply us with molded parts. During the ensuing turmoil we initiated our contingency plan of moving the molds to PAFC, which had recently installed a new 550-ton press capable of molding our larger parts. This transition would have been significantly more painful if our materials department had not planned for it.

** IGES (Initial Graphics Exchange Specification) is a file format that is used to describe models created with CAD programs.

Environmental Testing

The 2-to-3-μm-thick vacuum-deposited aluminum coating proved to be an ineffective shielding technology with which we experienced flaking problems, especially during humidity testing. Our search for an alternative included nickel-based and copper-based paints, electroless copper plating, and zinc arc spraying (ZAS). After tests and evaluations of the controllability of the processes, we opted for ZAS, a capability that PAFC had in-house. While this has proven to be more costly than vacuum-deposited aluminum, it has allowed us to meet our targeted CISPR 11* radiated emissions specification. Initially, because the silicon ZAS masking fixtures are soft and flexible for easy cleaning, the force of the spray would sometimes blow the mask away from the part allowing zinc overspray to get on the cosmetic surfaces. Reinforcing the masks brought the process under control.

Many of our part interfaces were designed with 2-to-3-μm-thick aluminum coating in mind and therefore had very small clearances. The move to 0.003-to-0.007-inch-thick zinc caused some clearances to become interferences. In some cases, this required tool modifications to make parts thinner. Fortunately, in most cases this could be accomplished by simply shaving the core-side parting line and readjusting shutoffs if necessary. In other cases, costlier tool modification requiring welding and recutting was needed. However, in one case welding was impractical and zinc had to be masked from affected areas. While shielding effectiveness was reduced, the ZAS process, coupled with other internal electrical and cabling modifications, gave us enough margin to pass the CISPR 11 specification.

The package was tested to the HP class B1 shock and vibration specification. It was tested in closed-up-for-travel, desktop (display up, keyboard down), and floor-standing configurations in both color and monochrome display versions. No unacceptable damage was sustained.

Cooling of the package is provided by an 80-mm-square tubeaxial fan with a maximum airflow of 38 ft^3/min. Ambient air is drawn in at the rear of the pod, across the pod electronics, and up through the top of the pod into the bottom chassis. Once inside the mainframe, it joins air coming from the base of the front panel, goes across the power supply and the PC system printed circuit board, and exhausts through the rear panel. Our worst-case pod, dissipating approximately 50 watts, experiences an air temperature rise of only 16°C. The temperature rise in the mainframe is 12°C above the system board and 19°C above the power supply.

All of these temperatures remain well below what we can tolerate given the temperature limitations of the disk drives and the LCD. The maximum allowable ambient operating temperature of the hard disk drive and color LCD is 40°C, and that of the flexible disk drive and monochrome LCD is 45°C. The minimum allowable storage temperature of either display is –25°C. Therefore, the full HP class B1 temperature specification had to be waived in deference to these components. The monochrome LCD experiences extreme response sluggishness at very low temperatures and washes out at high temperatures (adjusting the contrast control does not help). The color LCD experiences no such performance or visual degradation at the temperature extremes.

The supersoak and condensation portions of our humidity testing were not done because the LCDs and disk drives do not allow them. The polarizers on the LCDs cannot tolerate standing water for any length of time. However, we have an optical film applied to both LCDs for anti-glare and protection from scratching and chemical contamination. The other humidity testing done on the instrument presented no problems.

While our altitude testing indicated no problems, we learned a valuable lesson regarding altitude effects. The large label that covers the ribbing on the underside of the display cover lid entraps air in the dozens of hermetically sealed pockets it forms. When an instrument built at a 6000-foot altitude (like Colorado Springs) made its way to sea level, these pockets collapsed a little causing an unacceptable dimpling effect in the label. As an interim remedy, we had to machine vents into the top of one rib in each pocket to allow the pressure to equalize. Later, 122 pins were added to the mold to provide these vents.

ESD testing has shown no hard failures up to the 25-kilovolt limit.

Acknowledgments

I would like to recognize the other major contributors to the mechanical design effort. Ernie Hastings, the other product designer on the project, had primary responsibility for the keyboard and interface module designs. Mike Walker brought his industrial design experience and expertise and his model making abilities to the effort. Jim Young of HP Labs came to us for three weeks to lend his industrial design talents and to help us play "what if" on ME 30, and John Hamand, who spearheaded our materials efforts, played a critical role in keeping the tooling process on track and in ensuring that the switchover to PAFC as our molding vendor went smoothly.

* Comité International Spécial des Perturbations Radioélectriques (International Special Committee on Radio Interference)

The Microwave Transition Analyzer: A New Instrument Architecture for Component and Signal Analysis

The microwave transition analyzer brings time-domain analysis to RF and microwave component engineers. A very wide-bandwidth, dual-channel front end, a precisely uniform sampling interval, and powerful digital signal processing provide unprecedented measurement flexibility, including the ability to measure magnitude and phase transitions as fast as 25 picoseconds.

by David J. Ballo and John A. Wendler

As signal processing capabilities advance, modern microwave and radio frequency (RF) systems are becoming more and more sophisticated. Pulsed-RF signals, once used only for radar applications, are increasingly being used in communication systems as well. These signals routinely have complex modulation within the pulse, especially frequency and phase variations (see Fig. 1). Operating frequencies and bandwidths continue to increase, placing additional demands on the components of the systems.

Engineers responsible for the design and testing of such components and systems often need to measure them under the same dynamic conditions as those in which they are used. For example, it may be necessary to measure a device's response to phase coding or linear frequency chirp inside an RF pulse.

Measurements with traditional frequency-domain instrumentation are often insufficient to characterize and understand fully the operation of components in dynamic signal environments. Before the microwave transition analyzer introduced in this article, no single instrument could handle the diverse range of measurements required for dynamic testing at microwave frequencies. In addition to the new measurements it makes, this analyzer can perform many of the measurements previously requiring the use of network, spectrum, dynamic signal, and modulation analyzers, as well as oscilloscopes, counters, and power meters.

Importance of the Time Domain

A key benefit of the microwave transition analyzer is that it brings time-domain analysis to RF and microwave component engineers. In addition to its use in pulsed-RF testing, the time domain is essential to characterizing and understanding nonlinear devices because one can clearly and intuitively see the relationship between the input and output signals. As an example, both signals in Fig. 2 would appear identical if displayed on a spectrum analyzer. Even if the phase of the harmonics were known, the differences between the signals would not be immediately obvious. When viewed in the time domain, however, it is clear that signal 1 is clipped (the output of a limiter, say), while signal 2 has crossover distortion (what might be seen at the output of a Class-B amplifier, for example). Without the time domain, engineers have had to guess at the underlying causes of observed frequency-domain behavior. The ability to view microwave signals in the time domain has also proved to be extremely valuable to designers that are using CAE microwave design simulators, such as HP's MDS. Now simulations based on circuit models can be easily compared to actual measurements in both the time domain and the frequency domain.

Historically, most measurements on high-frequency nonlinear devices have been performed in the frequency domain. Often, this has been because of inadequacies in time-domain instrumentation. When frequency-domain information is of prime concern, spectrum analyzers are superb in their ability to display harmonic, modulation, and spurious signals with a large dynamic range. However, without the phase of the frequency components, the time-domain signal cannot be reconstructed. Network analyzers are excellent for performing linear, small-signal, frequency-domain testing, but they are limited in their ability to characterize nonlinear devices. The addition of harmonic and offset sweep capability in network analyzers has helped, but the time-domain perspective is still missing.

For envelope analysis of pulsed-RF signals, spectrum analyzers offer some limited time-domain capability. Recently, network analyzers have been adapted for pulsed-RF time-domain testing as well. Because of the architecture of these instruments, the intermediate frequency (IF) bandwidth imposes an upper limit on the measurement bandwidth. The result is minimum measurable edge times of greater than 100 ns. The microwave transition analyzer's architecture does not have this restriction. Edge speed is limited only by the RF bandwidth. Consequently, magnitude and phase measurements on pulses with rise times as fast as 25 ps are possible. Fig. 3 shows an example of a microwave transition analyzer measurement.

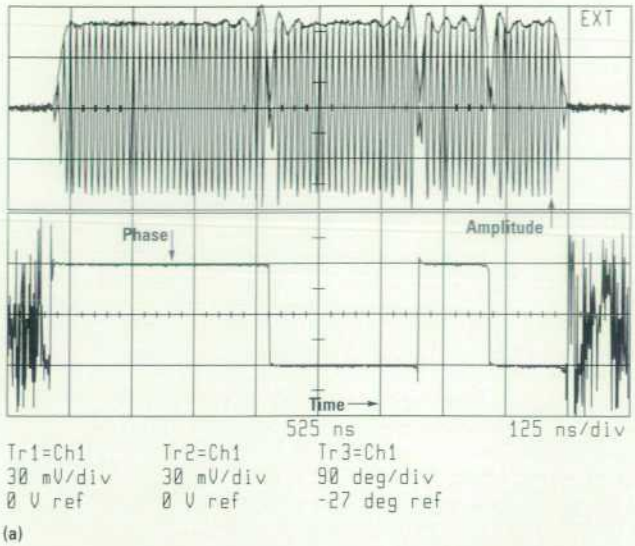


Fig. 1. Examples of complex modulation. (a) A phase coded RF pulse. The waveform and magnitude demodulation are shown in the upper half. The carrier's phase with respect to a CW reference is shown in the lower half. (b) Frequency modulation inside an RF pulse. The waveform and magnitude demodulation are shown at the top, the frequency demodulation is shown in the middle, and the magnitude spectrum of the pulse is shown at the bottom.

The ability to measure narrow pulses in the time domain can also be used to determine the impulse response (and therefore magnitude, relative phase, and group delay) of frequency translation components such as mixers and receivers. By stimulating these devices with a narrow pulse of RF energy, time-domain distortion can be directly observed. Often, it is the time-domain distortion that is of interest, even though it may be specified indirectly as magnitude and phase flatness versus frequency. By transforming the input and output pulses to the frequency domain with the built-in fast Fourier transform (FFT) and computing their ratio, the transfer function is obtained. From this, familiar results of magnitude and group delay versus frequency can be displayed. Network analyzers are only able to measure the phase and group delay of frequency translation components relative to a reference or "golden" device.

It is much easier to measure nonlinear devices at low frequencies than at RF and microwave frequencies. At low frequencies, general-purpose oscilloscopes readily show time-domain behavior, and dynamic signal analyzers provide both magnitude and phase in the frequency domain. The only tool available for high-speed time-domain measurements before the microwave transition analyzer has been the high-frequency sampling oscilloscope. Initially, sampling oscilloscopes were purely analog instruments, and in the past few years have incorporated digital storage and other enhancements such as markers. However, these instruments have not enjoyed widespread acceptance from RF and microwave engineers for several reasons. The first is the difficulties involved in achieving reliable external triggering at high frequencies and small signal levels. High-speed sampling oscilloscopes have enjoyed the most success for use with digital signals where voltage levels are generally large and triggers are not difficult to obtain. Secondly, traditional sampling oscilloscopes are not very sensitive, especially compared to network and spectrum analyzers. The microwave transition analyzer incorporates selectable filters to decrease noise without limiting the signal bandwidth. The resulting increase in sensitivity combined with internal triggering across the full RF bandwidth greatly aids in the measurement of small signals.

Excellent sensitivity also helps overcome a limitation of sampling oscilloscopes for high-input-impedance measurements ($\gg$50 ohms). Until recently, it has been very difficult to obtain probes with low enough parasitic capacitance to be useful at microwave frequencies. Companies now offer solutions for high-frequency passive probing, but signal attenuation is significant. This signal attenuation is not a problem for the microwave transition analyzer because of its high sensitivity. This has been especially beneficial for probing monolithic microwave integrated circuits (MMICs) at the wafer level.

Finally, the operation of high-speed oscilloscopes has not been optimized for RF and microwave applications, where terminology is often different from that used in digital design. The user interface of the microwave transition analyzer uses units and formats that are familiar to RF and microwave engineers. For example, log-magnitude displays of pulsed-RF signals are readily available, and marker annotation can be in dBm or dBc as well as volts.

Microwave Transition Analyzer

The HP 71500A microwave transition analyzer (Fig. 4) is a two-channel, sampler-based instrument with an RF bandwidth covering from dc to 40 GHz. The instrument is called a transition analyzer because of its ability to measure very fast magnitude and phase transitions under pulsed-RF conditions. However, this name does not encompass the full

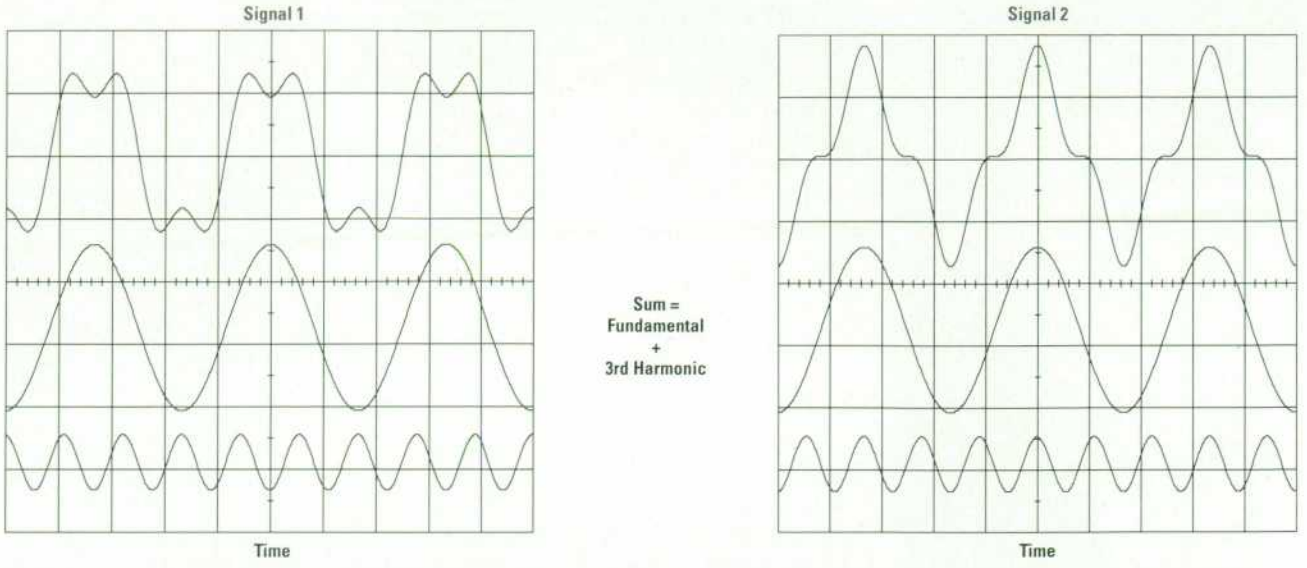


Fig. 2. The importance of phase information in nonlinear design. Signals 1 and 2 would appear identical on a spectrum analyzer display.

range of its measurement capability. The microwave transition analyzer can best be described as a cross between a high-frequency sampling oscilloscope, a dynamic signal analyzer, and a network analyzer.

Like a digital sampling oscilloscope, the microwave transition analyzer acquires a waveform by repetitively sampling the input, that is, one or more cycles of the periodic input signal occur between consecutive sample points. However, unlike an oscilloscope, the sampling instant is not determined by an external high-frequency trigger circuit. Instead, the sampling frequency is synthesized, based on the frequency of the input signal and the desired time scale. A synthesized sampling rate is an attribute that the microwave transition analyzer shares with dynamic signal analyzers. Also in common is an abundance of digital signal processing capability. The FFT, for example, allows simultaneous viewing of the time waveform and its frequency spectrum. However, unlike a dynamic signal analyzer, the microwave transition analyzer

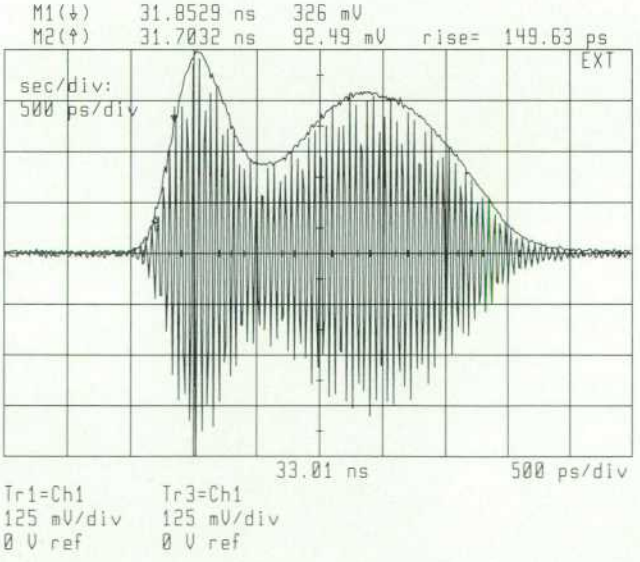


Fig. 3. The microwave transition analyzer can measure edge speeds on modulated waveforms as fast as 25 ps.

does not have an anti-aliasing filter at its input. The sampling frequency is automatically adjusted to achieve a controlled aliasing of the frequency components of the input signal. Finally, like a network analyzer, the microwave transition analyzer can be configured to control a synthesized signal source for the characterization of devices over frequency or power ranges. It can also receive a frequency that is offset from or a harmonic of the source frequency, and it can provide frequency and power sweeps at a particular point within a pulse of RF, on pulses as narrow as 1 ns.

Architecture

Fig. 5 shows a simplified block diagram of the microwave transition analyzer. The analyzer has two identical signal processing channels. Each channel samples and digitizes signals over an input bandwidth of dc to 40 GHz. The channels are sampled simultaneously (within 10 ps), permitting accurate ratioed amplitude and phase measurements. A single synthesized low-noise oscillator drives a step recovery diode, the output of which is split into two pulse trains that drive the microwave samplers. The microwave samplers and the analog-to-digital converters (ADCs) are run at the same frequency. The maximum sampling frequency is 20 MSa/s (20 million samples per second).

The signal at the output of the samplers is processed by a 10-MHz-bandwidth low-pass IF strip. The IF (intermediate frequency) circuitry includes a programmable shaping amplifier to compensate for the sampler's IF response roll-off, 60 dB of step gain to optimize the signal level into the ADC, and variable low-pass filtering to remove noise and sampler feedthrough. The trigger circuitry is at the end of the analog path. Triggering on IF signals (instead of RF input signals) allows the microwave transition analyzer to be internally triggered to 40 GHz. Enhancements to the hardware trigger are available through the use of digital signal processing.

Periodic Sampling

The mathematical analysis of periodic functions was begun in the early 19th century by Jean-Baptiste-Joseph Fourier. Fourier's theorem introduced the techniques for decomposing

Fig. 4. Named for its ability to measure very fast magnitude and phase transitions under pulsed-RF conditions, the HP 71500A microwave transition analyzer (top instrument) is part high-frequency sampling oscilloscope, part dynamic signal analyzer, and part network analyzer. The HP 71500A consists of the HP 78020A microwave transition analyzer module and the HP 70004A mainframe. The bottom instrument shown here is the HP 83640A synthesized sweeper.

any periodic waveform into a sum of harmonically related sinusoids. The Fourier series is a frequency-domain representation of the original time function and is used to simplify the description and provide insight into the function's underlying characteristics.

The sampler in the microwave transition analyzer is driven by a constant-frequency sampling signal. Because the sampler drive is periodic, Fourier analysis can be used to understand the sampler's operation. Often, periodic signals or systems responding to periodic signals are described and analyzed in the frequency domain. Transformations between the time and frequency domains replace convolution operations in one domain with multiplication in the other.

Filtering, a convolution operation in the time domain, is more easily interpreted as frequency-domain multiplication. Alternatively, a mixer multiplies two signals in the time domain, but the result is expressed as frequency-domain translation, a convolution operation. Why convolution is the analytical mechanism for realizing frequency translation is explained in "Frequency Translation as Convolution" on page 61.

An ideal sampler driven by a periodic sampling pulse can be considered a switch that briefly connects the input port to the output port at a periodic rate. When the switch is closed, the output signal is the input signal multiplied by unity. When the switch is open, the output signal is grounded, that is, the input signal is multiplied by zero. Thus, the signal at the sampler's output is formed as the product of the input signal and the periodic pulse defining the switch state as a function of time. As in the mixer example on page 61, time-domain multiplication results in frequency-domain convolution. The frequency spectrum of the sampler's input signal is convolved with the spectrum of the periodic pulse to produce the spectrum of the sampler's output (IF) signal.

The frequency spectrum of a periodic pulse is composed of delta functions at the fundamental repetition frequency and all multiples (harmonics) of this frequency. This infinite set of impulses in the frequency domain, sometimes called a frequency comb, inherits a magnitude and phase profile according to the time-domain pulse shape. A narrow, rectangular pulse imparts a sin(f)/f roll-off characteristic to the frequency comb. The first null of the response occurs at a frequency equal to the reciprocal of the pulse width and the 3-dB attenuation frequency occurs at 0.443 times this value. Fundamental to wide-bandwidth sampling is achieving a very narrow sampling pulse or aperture. The sampling aperture in the microwave transition analyzer is less than 20 ps.

The sampling front end of the microwave transition analyzer converts the high-frequency input signal to a low-frequency IF signal suitable for digitization and subsequent numerical processing. Depending on the application, three different interpretations of the sampling process are possible: frequency translation, frequency compression, and a combination of translation and compression.

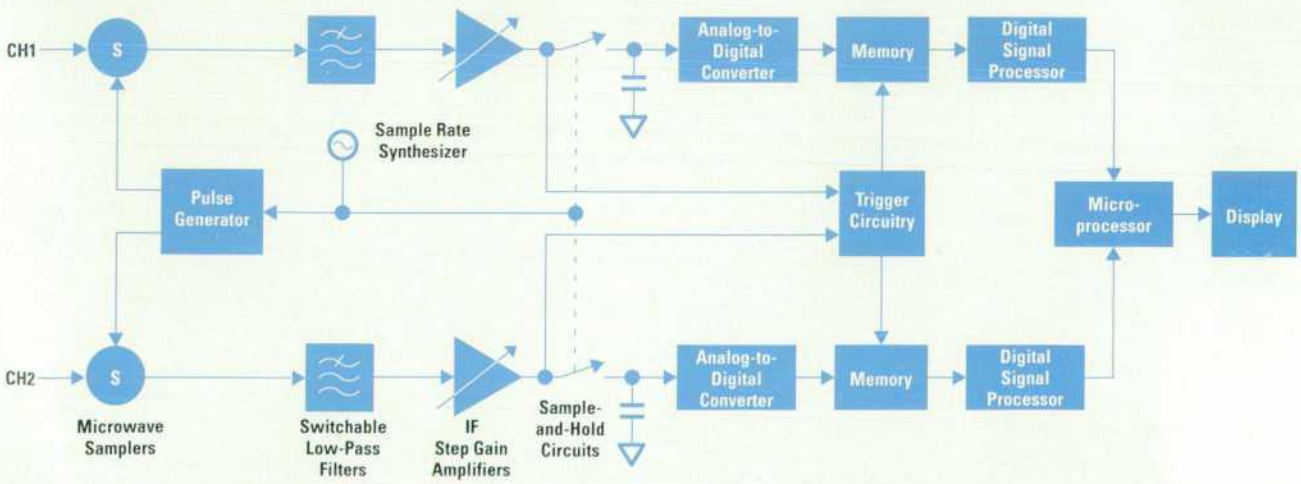


Fig. 5. Simplified block diagram of the HP 71500A microwave transition analyzer.

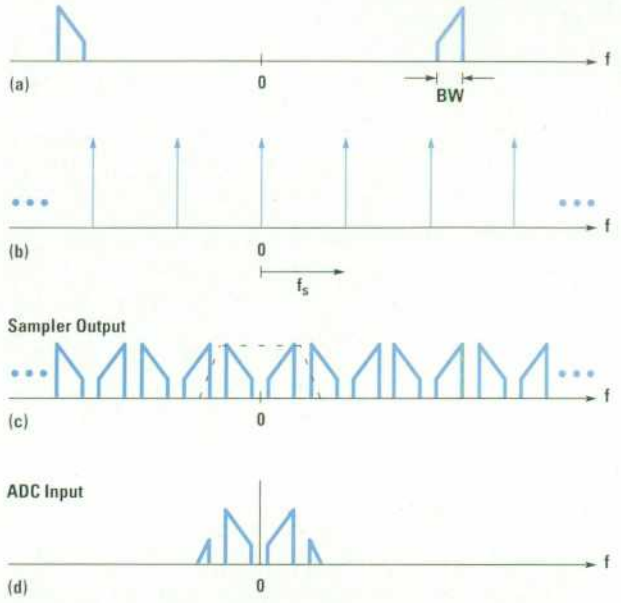


Fig. 6. Sampling used to translate a frequency band. (a) Input spectrum. (b) Sampling comb. (c) The sampler output spectrum is the convolution of the waveforms in (a) and (b). (d) Filtered output.

Frequency Translation

Nonrepetitive or single-shot events can be captured by sampling the input signal at a rate greater than twice the input bandwidth. This is known as the Nyquist criterion. However, maintaining this criterion does not imply that the sampling rate must be greater than twice the input signal's highest frequency. If the RF bandwidth of the sampler is adequate, narrowband information on a high-frequency carrier can be captured by low-frequency sampling, as long as a sampling rate of approximately twice the modulation bandwidth is maintained. Sampling the high-frequency signal translates the signal to baseband.

Samplers are often used in place of mixers for frequency conversion—for example, in the front ends of many general-purpose network analyzers. In the case of translation only, a given narrow frequency band is converted to baseband by an appropriate choice of sampling frequency. Fig. 6 diagrams the conversion process. The spectrum of the input signal is shown in Fig. 6a and the frequency comb of the sampling pulse is shown in Fig. 6b. The sampling frequency, that is, the spacing between the teeth of the frequency comb, is chosen such that the input spectrum lies appropriately positioned between adjacent comb teeth. The convolution result is shown in Fig. 6c.

Two important considerations in the choice of sampling frequency can be seen from these diagrams. First, the input signal bandwidth (Fig. 6a) must be less than one half the sample rate. Second, the sample rate must be chosen so the input spectrum is entirely contained in a frequency range bounded by the nearest sampling harmonic and the frequency halfway to the next higher or lower harmonic. If these criteria are not met, the sampler will translate or alias more than one component of the input spectrum to the same output frequency, causing uncorrectable errors. The maximum sampling rate of the microwave transition analyzer is 20 MSa/s. The rate is continuously adjustable (in 1-mHz steps) down to a minimum rate of 1 Sa/s and can be phase-locked to an external 10-MHz reference.

The signal at the output of the sampler is amplified and low-pass filtered before analog-to-digital conversion. This filtering virtually restores the original input spectrum, but it is now centered in the much lower IF range (Fig. 6d). Because the filter transition from passband to stopband is not immediate, some undesired high-frequency energy may be included in the signal presented to the ADC. In this case, the bandwidth of the signal at the ADC exceeds half the sample rate. Aliasing occurs as the highest-frequency components are folded back on top of the original translated spectrum by the sample-and-hold circuit of the ADC. However, unlike the aliasing problems mentioned in the previous paragraph, the effects of this aliasing can be predicted and corrected in software because the aliased components represent redundant information.

In summary, using a sampler with a bandwidth many times the sample rate allows the capture of single-shot events in the modulation on a high-frequency carrier (see Fig. 7). The analysis bandwidth is limited to half the sample rate.

Frequency Compression

A second, fundamentally different perspective of the sampling process is useful in the measurement of periodic high-frequency signals. Traditionally, these measurements have required trigger-based repetitive sampling techniques. In the microwave transition analyzer, precision RF trigger circuitry is not used. Periodic sampling alone is used to convert a strictly periodic input with harmonic components spread across a very wide bandwidth to a low-frequency signal with harmonic components spread over the narrow IF range. This is accomplished by choosing a sampling frequency that converts each component of the input signal into the IF such that the harmonic ordering, magnitude, and phase relationships of the original input are preserved in the IF signal. The sampling process effectively compresses the wide-bandwidth input signal into a low-frequency signal at the IF.

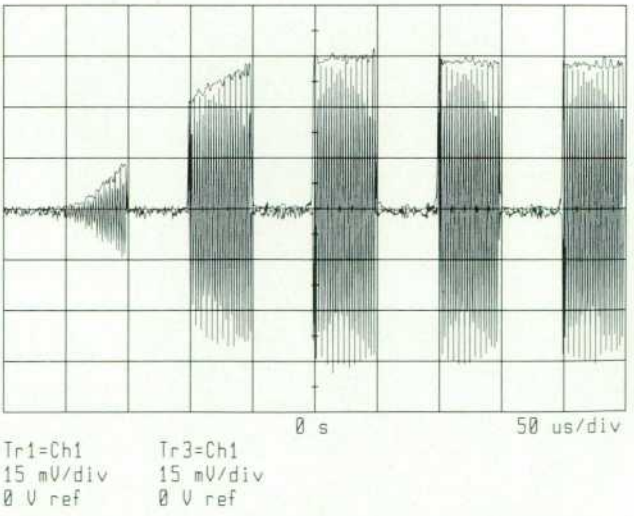


Fig. 7. Turn-on characteristic of a synthesizer's output amplifier. This single-shot measurement was internally triggered on the signal that originated from the enabling of the RF output of the synthesizer. The carrier frequency is 5 GHz.

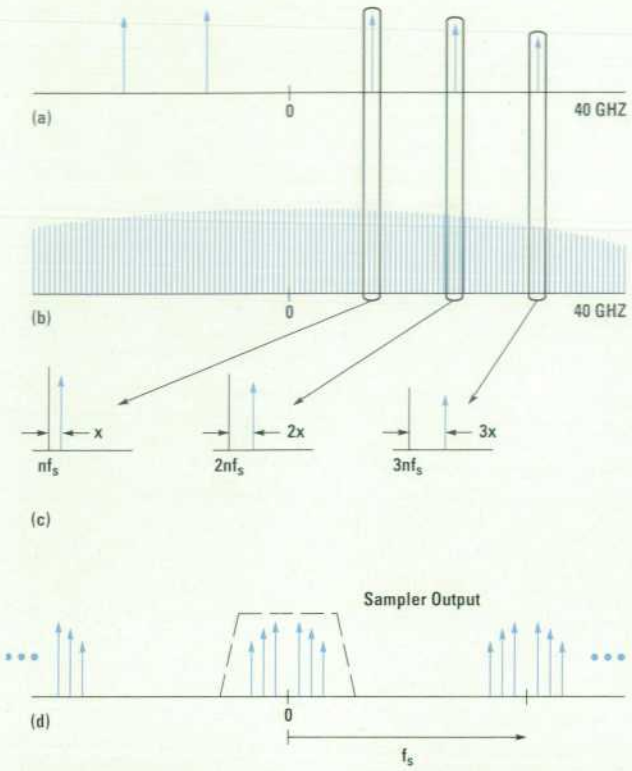


Fig. 8. Sampling used to frequency compress a periodic input signal. (a) Input signal spectrum. (b) Sampling comb. (c) Expanded frequency scale showing the relationship between the input and the sampling signal components. (d) The sampler output signal is the convolution of the waveforms in (a) and (b).

Fig. 8 illustrates the concept in the frequency domain. The input spectrum and frequency comb of the sampling pulse (including the RF response roll-off) are shown in Figs. 8a and 8b. Fig. 8c provides a close-up view of the relative positioning of the comb lines with respect to the input signal. The sampling rate is chosen such that a given harmonic (the nth) is positioned x Hz below the input's fundamental frequency. Then, the (2n)th sampling harmonic will be positioned 2x below the input's second harmonic, the (3n)th sampling harmonic will be 3x below the input's third harmonic, and so on. Fig. 8d shows the result of the convolution. Each harmonic of the input is converted to a corresponding harmonic of the low-frequency signal at the IF.

The sampler does not have infinite bandwidth, and the sin(f)/f roll-off of the sampling comb attenuates the IF responses that correspond to input components at the higher frequencies. Small amounts of attenuation may be compensated for in software, however, after the signal is digitized. The combination of a very narrow sampling aperture and software corrections allow the microwave transition analyzer to specify a flat response to 40 GHz.

Viewing this process in the time domain, the sample interval is set to be a multiple of the input period plus a small amount equal to the effective time between points (Fig. 9). Since the sampling interval is not an exact multiple of the input period, the sampling instant moves with respect to the input at a prescribed increment as the samples are acquired. The effective time between points is determined by how close the sampling frequency is to a subharmonic of the input frequency.

Compression Factor. The signal at the IF is a replica of the input signal, but at a much lower fundamental frequency. When this signal is digitized and displayed, the waveshape matches that of the input. The time range indicated on the display is calculated by dividing the real time (sample period times trace points) by the compression factor (input frequency × 1/x, where x corresponds to the fundamental frequency at the IF—see Fig. 8):

$$\text{Time Span} = \frac{(\text{Sample Period})\ (\text{Number of Trace Points})}{(\text{Input Frequency})/x}.$$

When the microwave transition analyzer is used for repetitive sampling, the input signal must be strictly periodic, and the period must be known to high accuracy. If the frequency that the analyzer assumes for the input signal is near but not exactly equal to the frequency of the signal being measured, the IF will be shifted in frequency by an amount equal to the difference. The resulting measurement will show an erroneous time scale, the error equal in percentage to the frequency error of the IF signal. Thus, a small RF inaccuracy can result in a very large time-scale error. The ability to frequency-lock the microwave transition analyzer's sampling rate to the signal being measured (by sharing a common reference frequency with the stimulus), removes this source of error. The resulting time scale accuracy is specified to 1 ps—better than any current trigger-based oscilloscope.

Triggering. To keep the display "triggered," low-frequency trigger circuitry is connected to the IF signal and used to initiate the storage of a data record relative to a rising or falling edge. Data samples in the buffer before the trigger occurrence are displayed as negative time (pretrigger view). Through the combination of periodic sampling and a low-frequency trigger circuit, the microwave transition analyzer is able to trigger internally on periodic signals across the full 40-GHz input bandwidth and offer negative-time capability without delay lines.

IF Filtering for Noise Reduction. As mentioned earlier, the signal at the output of the sampler is low-pass filtered before analog-to-digital conversion. In Fig. 8d the bandwidth chosen for this filtering is less than half the sampling rate. Any IF components above the band edge of the filter correspond to input harmonic components beyond the specified input bandwidth of 40 GHz and may be filtered off. Filtering the IF signal to a bandwidth narrower than half the sampling rate means that not all of the noise across the 40-GHz input bandwidth is converted to noise on the IF signal. Thus, noise is removed from the displayed signal without affecting the

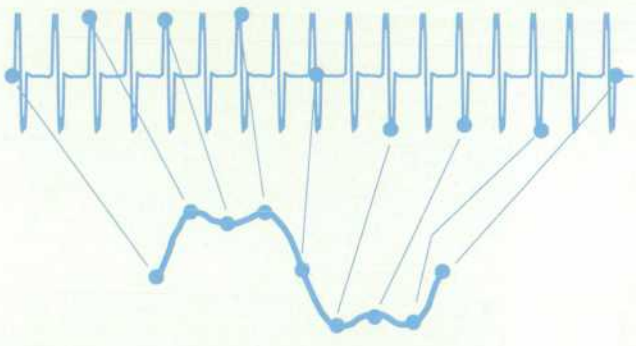


Fig. 9. Periodic sampling in the time domain.

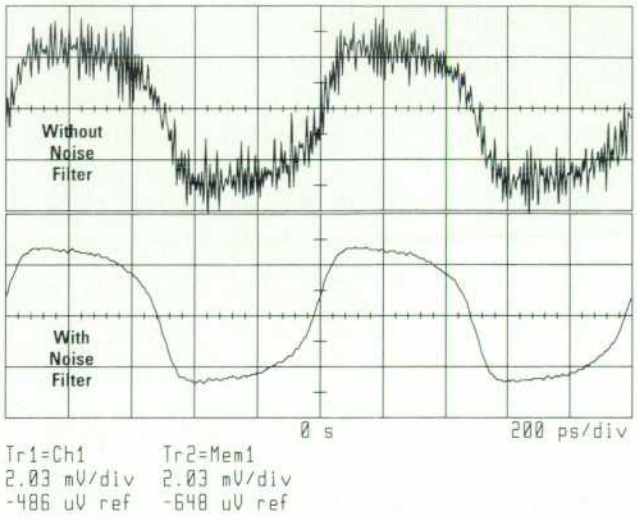


Fig. 10. Filtering the IF signal removes noise but retains the underlying wave shape.

waveshape. The result is cleaner displays and improved sensitivity (by more than 20 dB) compared to conventional trigger-based sampling oscilloscopes (see Fig. 10).

Translation and Compression

The perspectives of translation and compression are combined to analyze the third use of the microwave transition analyzer's sampling front end. The application is measuring signals composed of broadband, periodic modulation on a high-frequency carrier. Examples include pulsed-RF signals with narrow pulse widths or fast edge speeds. Proceeding as before, the spectrum of the input signal and the frequency comb of the sampling pulse are shown in Figs. 11a and 11b, respectively. Fig. 11c has an expanded frequency scale showing the relative positioning of the input's spectral lines and those of the sampling pulse. Two variables, x and y, are introduced in this figure, and are related to the concepts of compression and translation, respectively. The sampling frequency is chosen such that the signal's pulse repetition frequency (PRF) is slightly greater (x Hz) than a multiple of the sampling rate. In other words, the time between sampling instants is slightly greater than an integral number of input pulse repetition periods. As can be seen from the diagram, the frequency separation between a given signal component and the nearest sampling harmonic increments by x Hz when considering the next-higher signal component. Consequently, the spacing of the corresponding components in the sampler's output signal is x Hz, resulting in a compression factor of PRF/x.

In Fig. 11c, the spectral center of the input signal is shown to be offset by y Hz from the nearest sampling harmonic. Therefore, the signal at the output of the sampler is centered at y Hz, as shown in Fig. 11d. If the offset y is allowed to decrease by a change in the input carrier frequency, the sampler output components are translated toward one another as indicated by the dashed arrows. If y becomes too small, the components will partially overlap and distort the spectrum. Likewise, if y is increased, the sampler's output components move opposite to the directions indicated and will overlap as y approaches half the sampling rate.

For a given pulsed-RF input signal with an arbitrary carrier frequency, the values of x and y cannot be independently controlled by adjustments in the sampling rate alone. If the sampling rate is set to achieve the desired compression factor (PRF/x), there is no remaining degree of freedom for adjusting the spectral offset (y) to avoid overlap. One solution is to provide a mechanism for automatically adjusting the carrier frequency under control of the microwave transition analyzer. In many cases, the microwave transition analyzer is used in a stimulus-response configuration similar to that of a network analyzer. If the carrier source is under control, the carrier frequency control can be used to adjust the spectral offset independent of the sampling rate.

Often, however, the microwave transition analyzer does not control the carrier source, or it is desired that the carrier frequency not be modified. In these cases, the simultaneous requirements on the sample rate are achieved by slight modifications to either the requested time span or the number of trace points. The parameter to be modified is determined by the user. Remembering that the displayed time span is equal

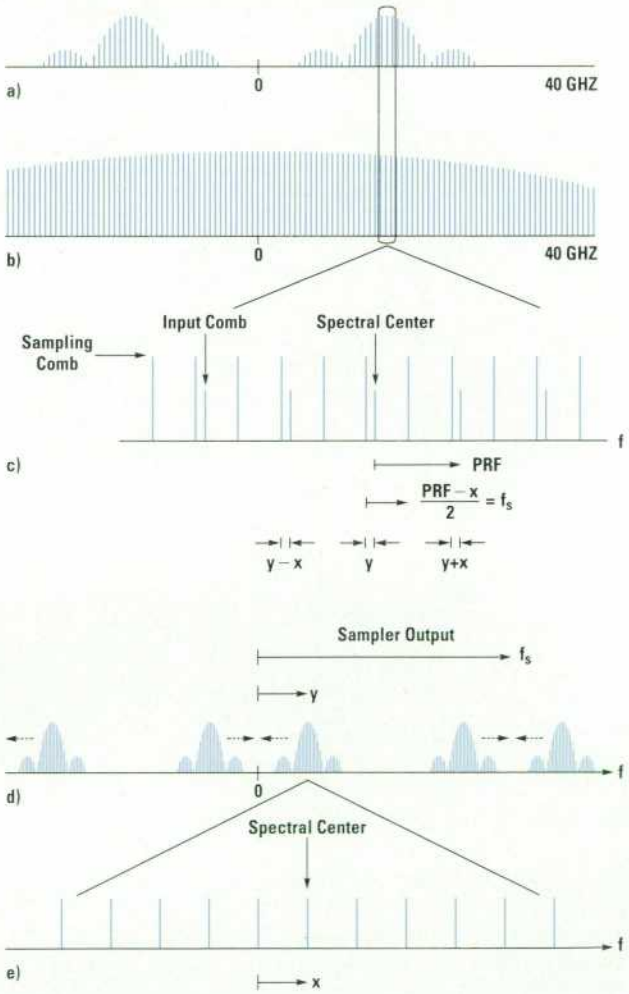


Fig. 11. Sampling used to analyze periodic wideband modulation. (a) Input signal spectrum. (b) Sampling comb. (c) Expanded frequency scale showing the relationship between input and sampling signal components. (d) The sampler output signal is the convolution of the waveforms in (a) and (b). (e) The IF spectrum on an expanded frequency scale, showing the spacing of the signal components.

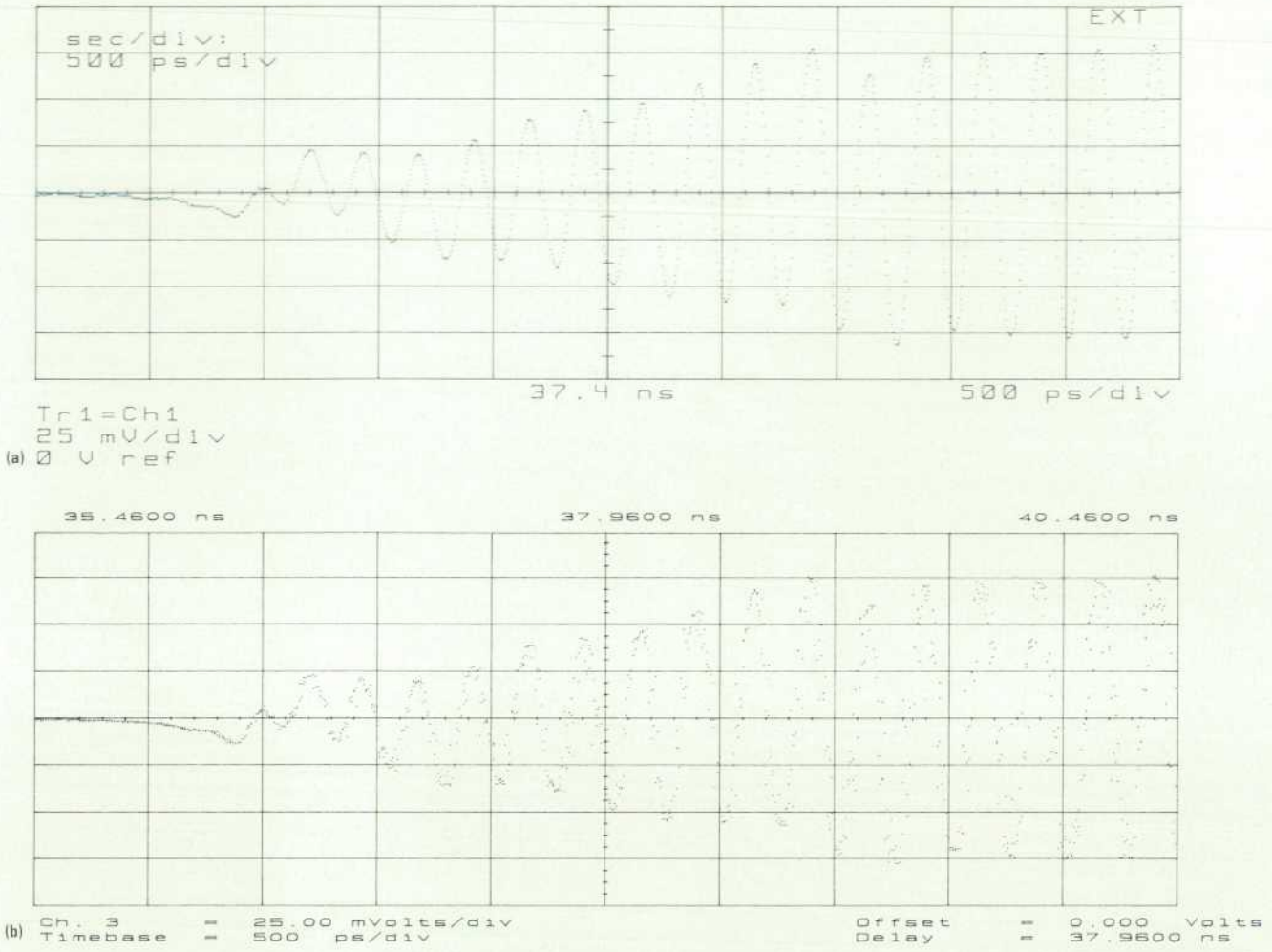


Fig. 12. Most digital signal processing algorithms require sampling the input signal at a uniform interval. (a) In the microwave transition analyzer, the sampler is driven at a synthesized rate, resulting in very precise sample timing. (b) Conventional sampling oscilloscopes rely on high-frequency trigger circuitry for timing accuracy, so the sampling interval can be considerably more uncertain.

to the real time divided by the compression factor, the following equation results:

$$\text{Time Span} = \frac{(\text{Sample Period})\ (\text{Number of Trace Points})}{\text{PRF}/x}.$$

Since the input PRF is a constant and x is a function of the PRF and the sampling rate (see Fig. 11c), the above equation relates the three variables: time span, number of trace points, and sampling period. Fixing either the time span or the number of trace points and slightly adjusting the other quantity results in a small change in the sampling period. A small change in sampling rate causes a much larger shift in the harmonic nearest the input carrier. In this fashion, the centering of the spectrum at the sampler's output is controlled.

Numerical Processing

Discussion to this point has concentrated on how the sampling process can be used to translate and/or compress a high-frequency input signal into a low-frequency signal at the IF suitable for digitization. Equally important for the microwave transition analyzer is the processing done on the signal after it has been digitized. Conventional digital signal processing algorithms, such as digital filtering, demodulation, and FFT analysis, assume that the input signal has been sampled at an exact, uniform rate. In the microwave transition analyzer, the sampling interval is synthesizer-based, resulting in sample-to-sample timing that is precisely uniform. A single trigger event initiates the storage of an entire trace of data. By contrast, conventional sampling oscilloscopes rely on high-frequency trigger circuitry to provide a consistent sampling interval. Since a different trigger event is used for the measurement of each data point, any triggering uncertainty results in sample-to-sample timing variations. Because the triggering accuracy is dependent both on the (trigger) signal characteristics and the amount of trigger delay selected, the resulting sampling interval can become significantly nonuniform under certain conditions, reducing the options for further numerical processing (see Fig. 12).

Analytic Signal Representation. One of the first operations applied to the sampled data is the creation of the quadrature function using the Hilbert transform. This quadrature function is combined with the original data to form a complex-valued representation of the waveform called the *analytic signal*. Just as complex-valued phasor representations simplify the analysis of linear circuits, the analytic signal

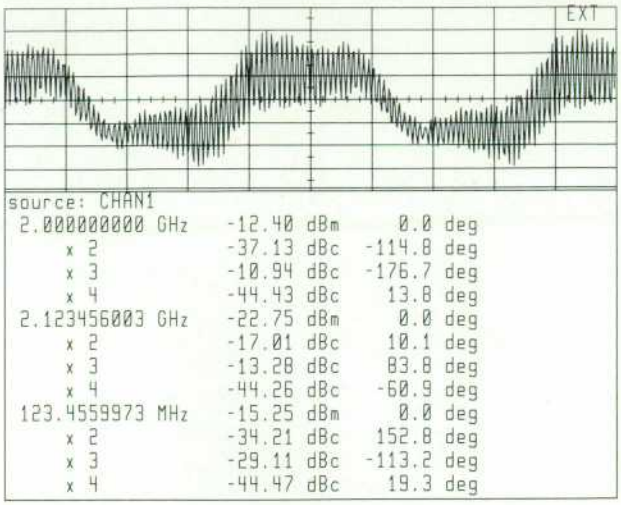
simplifies the manipulation and analysis of modulated waveforms. Use of the analytic signal representation also allows for vector normalization of traces, an operation normally found in network analyzer systems.

RF and User Corrections. One of the more obvious uses for additional processing of the sampled data is to compensate for nonideal conditions in the analog circuitry. As mentioned, the sampler has a frequency response characteristic largely determined by the aperture time of the sampling pulse. The magnitude response of the sampler is measured at the factory and stored inside the analyzer. This data is then used to correct for the sampler roll-off in subsequent measurements. Regardless of whether the sampler is used for translation or compression, an assumed, unique mapping exists between IF frequency and input RF frequency. When the IF signal is digitized, an FFT is used to convert the response into the frequency domain. The IF frequencies are mapped into RF frequencies, the appropriate correction at each frequency is applied, and the result is then transformed back to the time domain for display. The same processing routines are also available for user-definable filtering or corrections. User-defined corrections are useful in compensating for cabling or fixturing losses. Filtering applications might include simulating the magnitude and phase characteristics of a transmission channel to predict what effects the channel will have on specific signals.

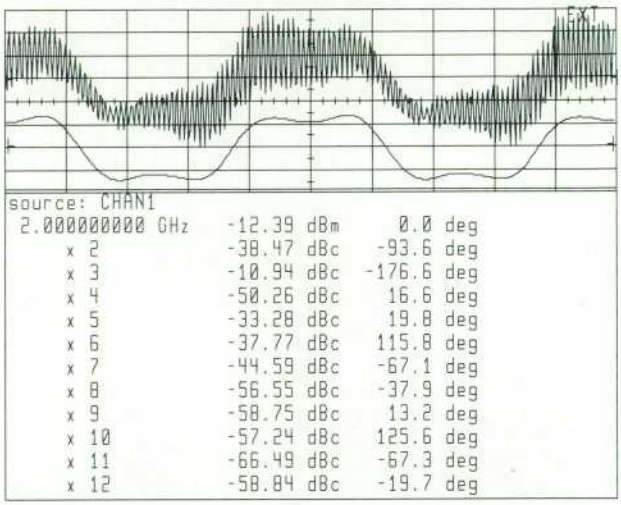
Frequency-Domain Measurements Using the FFT. The ability to execute transformations quickly between the time and frequency domains is important to the operation of the microwave transition analyzer. These tasks are accomplished by a pair of digital signal processing chips (one per channel) tightly coupled to the ADC memory. If the user requests a frequency-domain display, the signal is shown after frequency corrections are applied without the transformation back to the time domain. Separate frequency-domain controls allow the user to zoom in on a narrow portion of the original frequency span. This is accomplished by processing a longer time record with the FFT and displaying only part of the frequency outputs.

As mentioned above, the microwave transition analyzer assumes a unique mapping between the IF and RF frequencies, resulting in a replicated version of the input signal at the IF. However, if a second, unrelated signal is present at the input, it too is converted to the IF and becomes part of the sampled signal. The components of this second signal will fall at seemingly arbitrary IF positions and will not correspond at all to the IF-to-RF mapping that the first signal obeys. Except for the operating mode described below, the microwave transition analyzer is designed, like most oscilloscopes, for the measurement and display of a single signal.

If multiple, nonharmonically related signals are known to be present at the input, the microwave transition analyzer can be instructed to measure these signals independently using the table mode of operation. In the table mode, the fundamental and harmonics of up to five signals are measured, using the FFT to measure each component of each signal individually. The sampling frequency is chosen to avoid converting different spectral components to the same frequency at the IF. The results are displayed in tabular form (see Fig. 13). The table can be updated continuously for only one of the signals or for all of them. If a waveform display is desired,



(a)



(b)

Fig. 13. Microwave transition analyzer table mode displays. (a) The output of a mixer shown in the time-domain display at the top is the combination of several signals. The table displayed at the bottom provides information about the frequencies present. (b) The table is configured to display information for only one of the signals present. The second trace at the top is constructed from this measurement data.

a data trace can be constructed according to the table values and the specified time span. This capability allows the microwave transition analyzer to display time-domain signals in a frequency-selective fashion, combining some of the attributes of both oscilloscopes and spectrum analyzers.

Phase Trigger. Another use of the microwave transition analyzer's FFT resources is in measuring low-level or noisy signals. Trace averaging is used by oscilloscopes to reduce noise on a displayed waveform. However, averaging can work only if the waveform is reliably triggered, which is difficult on low-level or noisy signals. In the microwave transition analyzer, waveform capture is not dependent on reliable triggering, but on knowing the input frequency and sampling at the proper synthesized rate. If at least one period of the signal is collected into memory on every sweep, the trigger point will always be somewhere in this record of data. The microwave transition analyzer introduces a special

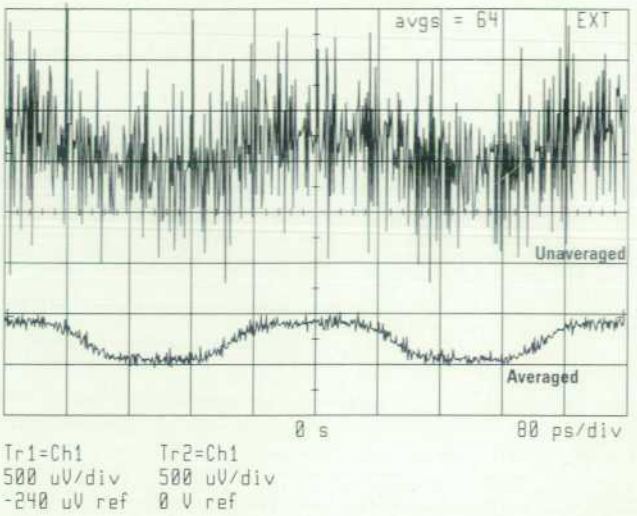


Fig. 14. Very small or noisy signals can be reliably triggered with the microwave transition analyzer's phase trigger. Sweep-to-sweep averaging can then be used to reduce the noise.

trigger mode, called *phase trigger*. The trigger value is specified in terms of the phase of the fundamental component of the signal. An FFT is used to measure the phase of the fundamental at the midpoint of the time record. From this, the index in the record that corresponds to the trigger point is determined, and the correct portion of the record is copied to the display trace. Although the trigger point in the larger memory record may move about from sweep to sweep, the display trace stays triggered. The result is that processing gain (via the FFT) has been applied to extract the trigger information, allowing stable triggering on even very noisy signals (Fig. 14). Trace averaging can then be used to reduce the noise on the displayed waveform.

Component Test Systems

Electronic components are most often tested by measuring their response to a given stimulus. The stimulus source can be anything from an impulse generator to a sweep oscillator or a synthesizer. The measurement instruments include oscilloscopes, spectrum analyzers, and dedicated receivers. Stimulus and measurement functionality are frequently combined to form a stimulus-response system. Network analyzers, spectrum analyzers with tracking sources, and oscilloscopes with built-in step generators are examples of systems in which the stimulus source is controlled directly by the measurement instrument. The microwave transition analyzer, with its built-in pulse generator and ability to control synthesizers over a private HP-IB (IEEE 488, IEC 625), can be configured as a versatile stimulus-response system for component or system test.

An advantage of configuring the microwave transition analyzer as a stimulus-response system for time-domain measurements is that the analyzer is always certain of the signal frequency. Configuring the stimulus under the control of the microwave transition analyzer and sharing a common 10-MHz reference ensures agreement between the assumed and actual input frequencies. Additionally, indirect adjustment of the stimulus via the controls of the microwave transition analyzer allows interesting new time-domain capabilities. One is the ability to hold a fixed number of signal periods on the display regardless of the stimulus frequency. In other words, the time range is automatically updated at any change of stimulus frequency. Using this feature, designers can see changes in a device's response as it operates over a range of frequencies, without the continuous time scale adjustments that would be required with a conventional oscilloscope.

RF and microwave design engineers are familiar with the use of synthesized signal generators for testing their devices. However, for designers requiring a nonsinusoidal stimulus, synthesized pulse generators are not generally available. The repetition interval of most pulse generators is not constant enough for repetitive sampling with the microwave transition analyzer. To ease this problem, the analyzer provides a variable-rate, TTL-level output pulse that is frequency-locked to the sample rate synthesizer. The pulse width and period are adjustable in 100-ns increments. This output can be used directly or as the trigger input to a standard pulse generator, thereby locking the repetition rate to the time base of the microwave transition analyzer. If the pulse is used for modulating a carrier, the analyzer needs to know the carrier frequency to sample the signal at the correct rate. (See the earlier discussion on translation and compression.) For stimulus-response testing under pulsed-RF conditions, this is most easily accomplished by having a configured synthesizer supply the carrier.

The automatic control of a sinusoidal signal generator by the microwave transition analyzer results in an instrument system with the flexibility to measure the response of a device as a function of time, input frequency, or input power. In addition to showing time-domain responses like an oscilloscope at various input frequency and power levels, the analyzer can automatically step the source across a frequency or power range and provide measurement functionality similar to that found in network analyzers. At each point in a frequency or power sweep, the magnitudes and phases of the sinusoids at the two input channels are measured by collecting a set of time samples and applying the FFT. Increasing the number of time samples used in the FFT is equivalent to decreasing the processing bandwidth and results in a more accurate measurement (at the expense of sweep speed). Because of the frequency discrimination provided by the FFT, the measured frequency need not be the same as the stimulus. Conversion loss in devices responding at frequencies that are offset from or harmonic multiples of the stimulus is easily measured with the microwave transition analyzer.

Signal Test Applications

For signal test as opposed to component test applications, the microwave transition analyzer is used as a stand-alone instrument. Repetitive sampling still requires the signal to be periodic, but radar and communication systems are increasingly moving to highly stable, synthesizer-based designs locked to a common reference. Hewlett-Packard's frequency agile signal simulator (FASS) is an example. Many times, the testing of these systems can be accomplished with the system in a periodic operating mode.

In applications where the signal frequency is unknown, the microwave transition analyzer can be used like a counter to determine the signal frequency to high precision. This is accomplished by taking several measurements of the input signal at different sampling rates and comparing the change

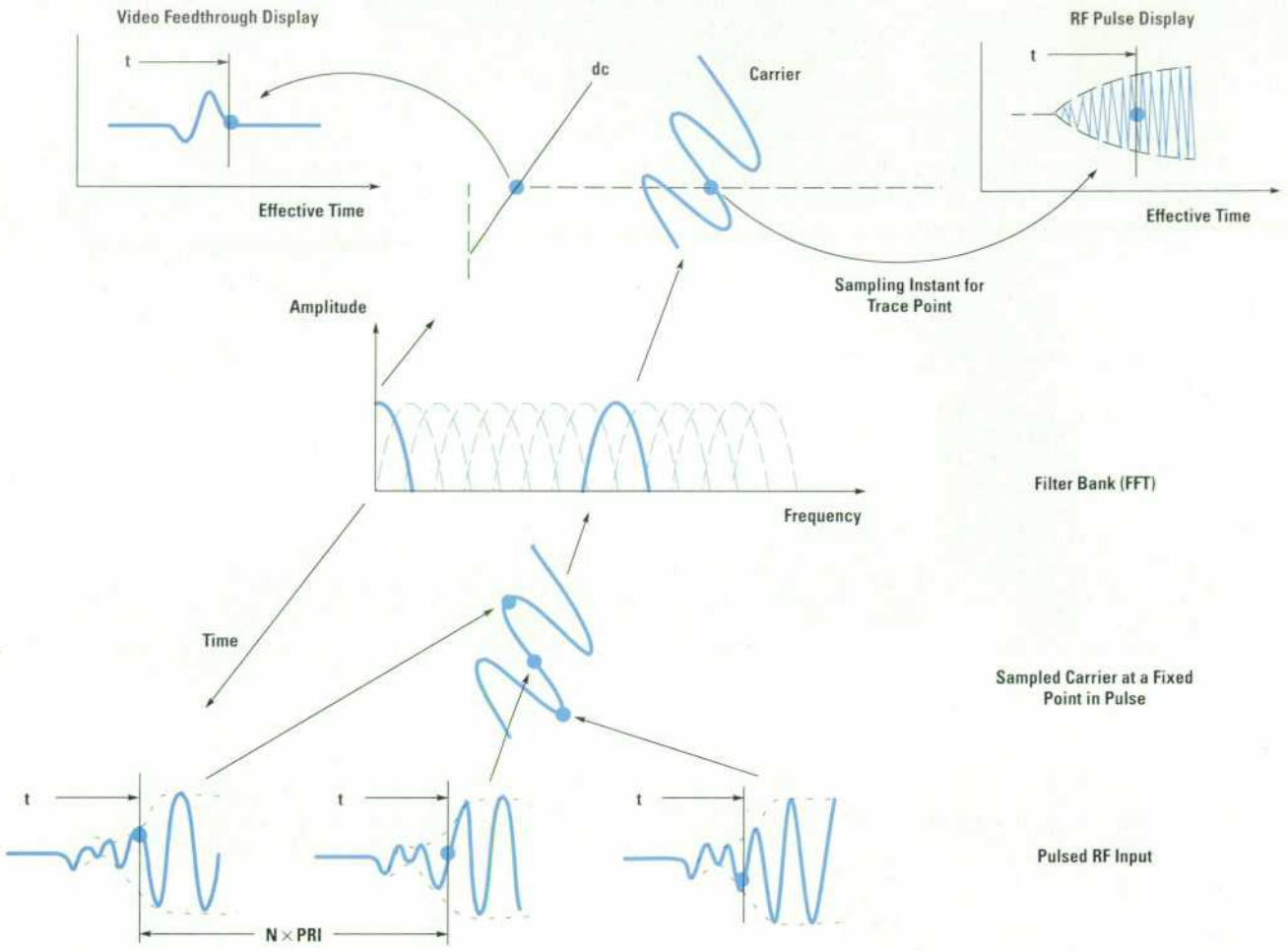


Fig. 15. Processing flow in the stationary sampling mode of operation.

in the IF to the change in sampling rate.[1] For CW signals with at least a 10% duty cycle, the microwave transition analyzer will determine the frequency to an accuracy of 1 part in 10^8. If multiple signals are present at the input, the fundamental frequency for each (up to a maximum of five) will be returned. The analyzer is also able to measure the carrier frequency of pulse modulated signals for pulse widths as narrow as 300 ns. Because the data acquisition for this measurement is single-shot, the pulse repetition interval need not be constant.

Stationary Sampling Mode

A measurement mode known as *stationary sampling* offers significant enhancements to the microwave transition analyzer's pulsed-RF capabilities. Stationary sampling is a technique that substantially reduces the trace noise on time-domain displays, resulting in increased sensitivity and dynamic range. Furthermore, it is through stationary sampling that pulsed network sweeps of frequency and power are achieved. Fig. 15 illustrates the process.

A prerequisite for stationary sampling is that the carrier frequency is not a harmonic multiple of the modulation period. That is, the modulation is not coherent with the carrier. Under this assumption, if the microwave transition analyzer samples the signal at a rate equal to the modulation rate, the sampling instant stays fixed with respect to the modulating envelope, but not with respect to the carrier. This is illustrated at the bottom of the figure. The sampling rate is set to be either equal to or an exact submultiple of the pulse repetition frequency. The resulting set of samples describes the carrier waveform at a particular point in the pulse. The data is then passed through a narrowband filter implemented with an FFT. This filtering acts to suppress noise and separate the carrier fundamental from dc and harmonic components. The complex-valued FFT output bin corresponding to the frequency of the sampled carrier represents one (analytic) time sample of the filtered waveform. This output becomes one data point in the final trace.

The next trace point needs to be taken at a different position with respect to the modulating envelope. To accomplish this, the internal synthesizer controlling the sampling rate is phase-shifted a precise amount. This moves the sampling instant the desired time increment along the modulating envelope. A new set of carrier samples is collected, processed with the FFT, and another complex valued output point is stored to the final trace. The process is repeated for every trace point. The amount of filtering that is applied in creating the output trace is adjustable by the user and is directly related to the number of time samples used in the FFT.

The sampling and filtering process separates the RF component from the dc component at each point in the pulse. Depending on which FFT output bin is recorded, either the dc feedthrough or the RF portion of the input waveform is ultimately displayed (see Fig. 15). This eliminates the requirement for the user to supply external filtering when performing these measurements. Also, measurements of carrier distortion at a particular point in the pulse are possible by setting the time span to zero. The FFT filtering in Fig. 15 is omitted and the sampled carrier is directly displayed. The measurement point is controlled by adjusting the trigger delay. Transforming the carrier waveform into the frequency domain allows easy measurement of the distortion components.

For pulsed network analyzer sweeps, when the sweep axis is frequency or power, the operation is very similar to that described for time sweeps. A triggering process aligns the measurement point with respect to the modulating envelope. Then, instead of phase shifting the sample rate synthesizer between each trace point, the carrier's frequency or power is automatically stepped. The result is a measure of the response of the device as a function of carrier frequency or power at a particular point in the modulating envelope. In conventional pulsed network analyzers, the IF bandwidth sets limitations on edge speed and pulse width. In the microwave transition analyzer, the measurement is performed using repetitive sampling techniques, and the modulation bandwidth is limited only by the sampler's RF response.

User Interface Design

The user-interface design of an instrument as versatile as the microwave transition analyzer required considerable work. Its feature set includes functionality found in a variety of microwave test equipment. The interface must not only provide control for displays of voltage versus time like an oscilloscope, magnitude versus time like a peak power meter, and phase or frequency versus time like a modulation domain analyzer, but must also allow for the automatic control of an external synthesizer, providing CW and pulsed network measurements of magnitude and phase versus frequency or power. Additionally, the FFT can be used for harmonic analysis, showing a display similar to that of a spectrum analyzer, and the automatic signal acquisition routines can be used to provide functionality found in CW and pulse counters and vector voltmeters.

The challenge in any interface design is to balance the requirement of complete functional access with the need for simple, intuitive controls for specific, targeted applications. One approach suggested early in the development cycle was to have the microwave transition analyzer assume different instrument personalities. For example, the user would choose an oscilloscope interface for one measurement, then switch to a network analyzer interface for a second measurement, then to a spectrum analyzer for another, and so on. The appeal of this approach is obvious: users need not learn a new interface. But as the implementation developed, the problems began to outweigh the benefits.

For example, an interface constructed according to this logic would forbid the simultaneous display of a voltage-versus-time waveform and its frequency spectrum, thereby losing a valuable perspective in the analysis of nonlinear operation. One of the major contributions of the microwave transition analyzer is its multidomain capabilities, and the interface needed to emphasize this strength. Less important, but still significant, is the fact that marker operation is substantially different for oscilloscopes, spectrum analyzers, and network analyzers. Any implementation that might impose three different marker systems on the user would be hard to describe as user-friendly, yet a common marker system weakens the implementation of instrument-specific personalities. Most important, measurement features unique to the microwave transition analyzer have no home in such an interface. Realizing this, the designers set out to create a versatile core interface targeted at two application areas: pulsed-RF or switched-RF component test, and time-domain analysis of microwave devices. Later, simplified interfaces for specific applications could be developed by drawing features from this core.

Pulsed-RF Testing. The versatility inherent in the architecture of the microwave transition analyzer allows a good match to the needs of high-speed pulsed-RF characterization. Designers in this area have traditionally required a wide variety of test instrumentation. Measurements of magnitude settling time are possible by combining an oscilloscope, a broadband detector, and a filter to remove the video feedthrough. Measuring phase settling time has been much more difficult, usually requiring the use of modulation-domain analyzers, pulse network analyzers, or custom down-converters, digitizers, and software. The fundamental attributes of the microwave transition analyzer's architecture—a very wide-bandwidth, dual-channel front end, a precisely uniform sampling interval, and powerful digital signal processing—provide the elements for unprecedented measurement flexibility in pulsed-RF component test. This, combined with the analyzer's singular ability to measure magnitude and phase settling times on edges as fast as 25 ps, is the reason for tailoring the interface to pulsed-RF testing.

Making it easy to demodulate a voltage-versus-time display of an RF pulse and show magnitude, log magnitude, or phase versus time was a key goal of the implementation. These sophisticated digital demodulation procedures are accessible simply by choosing a display format for the trace. The phase slope can be removed mathematically at the press of a button, or the phase can be measured with respect to the other channel by defining the trace input as a ratio of the channels. On pulse waveforms with excessive amounts of video feedthrough, the stationary sampling mode can be used to separate the RF and video portions of the waveform with digital filtering and display each portion independently. Invoking the mode is accomplished by simply turning on a filter. A variety of additional processing is available by defining a trace in terms of digital signal processing operations on channels, memories, and other traces (see Fig. 16). The result is a powerful digital signal processing system that is available to the user in a form that is easy to understand and simple to use.

Recognizing the limited availability of synthesized pulse generators, the design team decided to include one in the analyzer. Using this output to control the modulation period and a configured synthesizer to supply the carrier means that all stimulus adjustments are controlled through the interface of the microwave transition analyzer. Since the analyzer needs to know these signal parameters to set the correct sampling rate, a configured setup eliminates the (sometimes

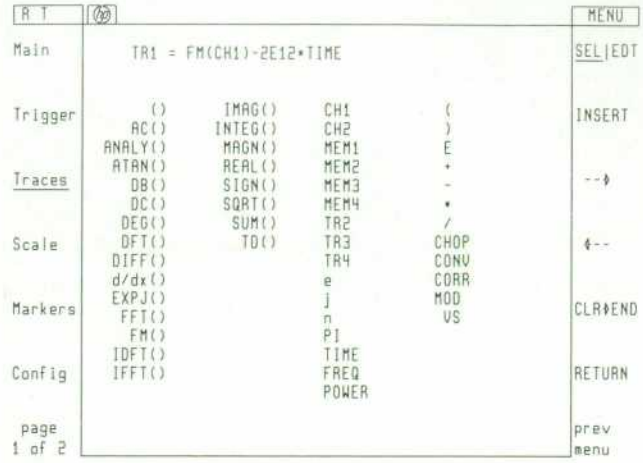


Fig. 16. The microwave transition analyzer includes a powerful trace processing system. The definition shown here for the display trace can be used to measure deviation from linear frequency chirp.

nonobvious) requirement of keeping the microwave transition analyzer abreast of frequency changes made on the signal generators. Furthermore, because the analyzer has complete control of the stimulus, network measurements as a function of carrier frequency or power are also possible. This added flexibility offers the user a multidimensional perspective on the device's operation, along measurement axes of time, carrier frequency, or carrier power (see Fig. 17). One variable is swept while the other two are held fixed.

An Oscilloscope for the Microwave Engineer. The tools of the trade in microwave component design are primarily frequency-domain instruments such as spectrum and network analyzers. Time-domain analysis is not nearly as prevalent at these frequencies as it is at the lower frequencies.

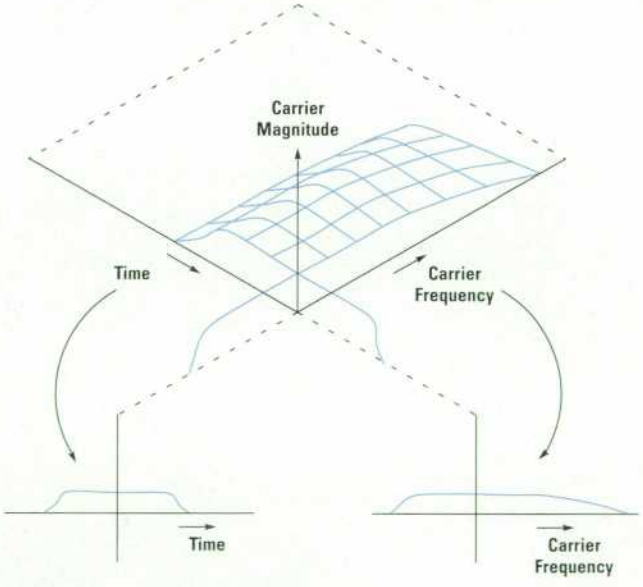


Fig. 17. A multidimensional perspective is sometimes useful in pulsed-RF device characterization. The microwave transition analyzer measures the response of a device as a function of time, carrier frequency, or carrier power. One variable is swept while the other two are held fixed.

The designers felt that by eliminating some of the roadblocks in triggering and sensitivity, the microwave transition analyzer had the potential to bring a time-domain perspective back to microwave design. The user interface was designed accordingly, with the microwave engineer in mind.

One of the simplest and most obvious changes to a standard oscilloscope interface is that display ranges, trigger levels, and marker readouts are entered and annotated in dBm as well as volts. Also, unlike many oscilloscopes, the channel hardware is continuously autoranged and unaffected by the display scaling, which is just a mathematical operation on the acquired data. If desired, this autoranging feature can be disabled. Sensitive internal triggering over the bandwidth of the instrument, combined with new features such as holding a constant number of cycles on the display, filtering away noise instead of averaging, and reliably triggering on even very noisy signals with the phase trigger, all work to simplify the measurement process.

IBASIC Implementation

Despite considerable attention paid to the interface design, some users may still find the controls somewhat intimidating, especially those who work in applications outside the targeted areas. The goals of measurement flexibility and ease of use generally conflict at the design of the user interface. To address this concern, the microwave transition analyzer allows the user to generate custom, application-specific interfaces through the internal execution of HP Instrument BASIC programs. IBASIC eliminates the need for an external controller by bringing the computer inside the analyzer. Programs can be generated and edited by attaching a standard HP-HIL keyboard to the front of the mainframe. Also incorporated into the HP 70004A mainframe is a memory card interface that can be used as a disk drive for the system. External disk drives are also supported over the HP-IB interface. Specialized trace processing, custom interfaces, multistep procedures, programmable control of other instruments—in short, completely customized measurements—are possible using the microwave transition analyzer running an IBASIC program like the one shown in Fig. 18.

Fig. 18. IBASIC programs allow generation of custom user interfaces.

Frequency Translation as Convolution

An ideal mixer multiplies the two signals at its RF and LO ports to produce the signal at the IF port, as shown in Fig. 1.

The frequency-domain representation of the RF and LO signals is shown in Fig. 2.

The convolution of the two frequency functions h(f) and k(f) is the value of the integral:

$$g(f) = \int_{-\infty}^{\infty} h(x)\ k(f - x)\ dx.$$

The function k(x) is first frequency-reversed, that is, folded about the dc axis giving k(−x). Then, at each evaluation frequency f, k(−x) is shifted by f with respect to h(x). The area under the product of the two functions is the convolution output at this frequency. Fig. 3 diagrams the procedure for the output frequency $f = f_2 - f_1$. The product is a single delta function, the area of which is $a_1a_2/4$. This is the convolution result at the frequency $f = f_2 - f_1$.

It is easy to verify that the output will be nonzero only at four values of f: $f_1 + f_2$, $f_1 - f_2$, $-f_1 + f_2$, and $-f_1 - f_2$. At each of these frequencies, the output is $a_1a_2/4$. This result is shown in Fig. 4. This frequency-domain representation is equivalent to the sum of two cosine waves, one at frequency $f_2 - f_1$ and the other at $f_2 + f_1$. The amplitudes are $a_1a_2/2$. Using trigonometric identities, it's easy to verify that this result is equivalent to $a_1a_2\cos(f_1t)\cos(f_2t)$.

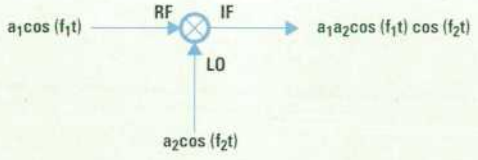


Fig. 1. Ideal mixer operation.

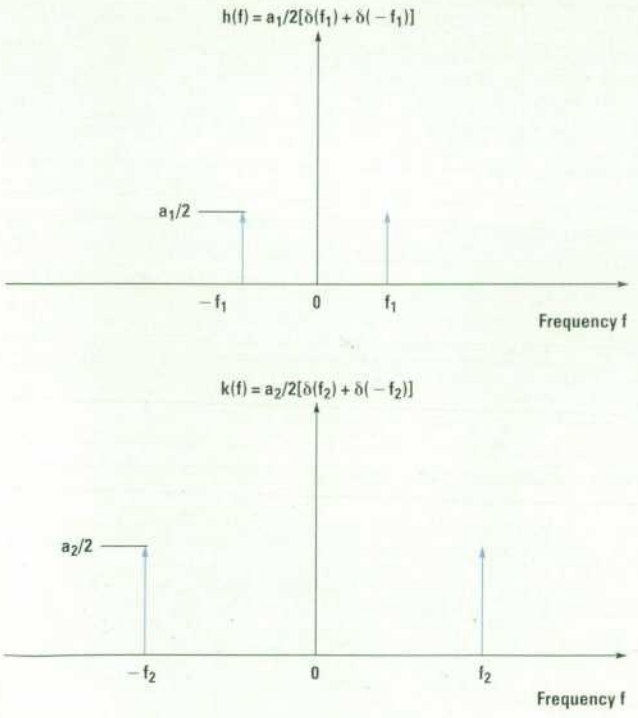


Fig. 2. Frequency-domain representation of the RF and LO signals.

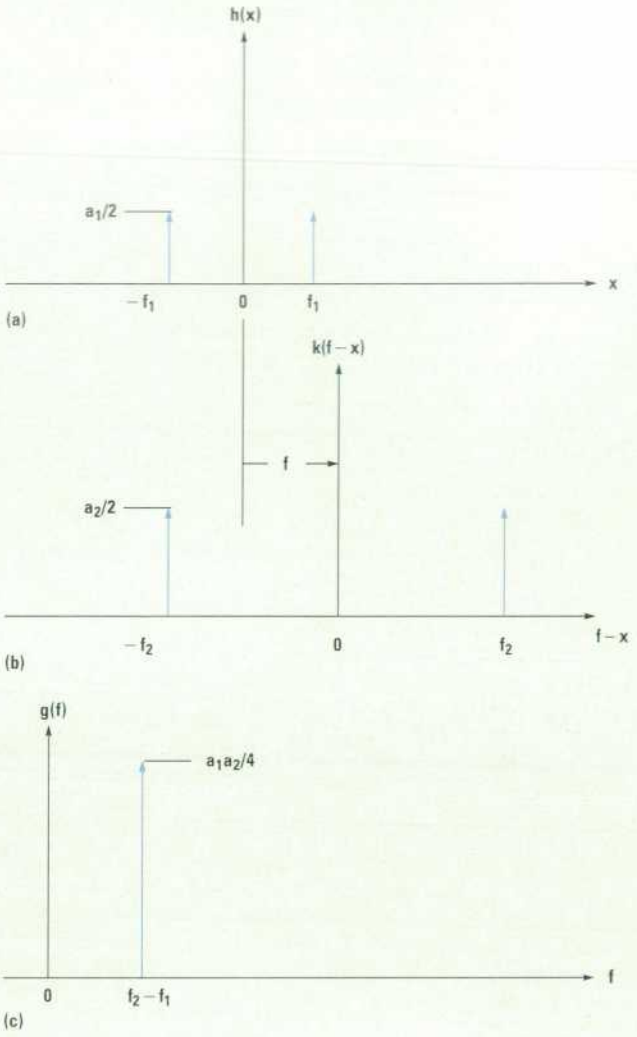


Fig. 3. Convolution result for $f = f_2 - f_1$.

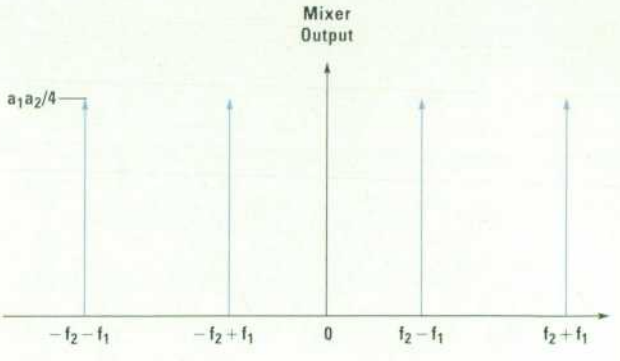


Fig. 4. Mixer output spectrum.

Summary

In addition to bringing the time domain to microwave design, the microwave transition analyzer measures harmonic distortion using the FFT and provides familar vector network analyzer capability when configured with a synthesized signal generator. In this respect, the microwave transition analyzer is a general-purpose, multidomain tool that can be used to link new time-domain measurements with traditional frequency-domain techniques, particularly in the areas of pulsed-RF and nonlinear device characterization. In a single instrument, the microwave transition analyzer integrates a versatile hardware architecture with very flexible means of control. The combination results in an instrument with unprecedented measurement diversity.

Acknowledgments

Special thanks go to the project managers: Mike Marzalek for initiating the project and Jim Coffron for carrying the project through to production. Steve Peterson wrote the bulk of the instrument firmware. John Wilson provided valuable market research and user interface ideas. George Reynard, the production engineer, did a wonderful job making life difficult for the project team, and Dave Sharrit helped steer our way through some very thorny technical issues.

Reference

1. A. Bologlu and V.A. Barber, "Microprocessor-Controlled Harmonic Heterodyne Microwave Counter also Measures Amplitudes," *Hewlett-Packard Journal*, Vol. 29, no. 9, May 1978, pp. 2-16.

Design Considerations in the Microwave Transition Analyzer

Digital signal processing is used extensively to improve the performance of the microwave sampler, the sample-rate synthesizer, and the high-speed analog-to-digital converter, and to extract and display input signal characteristics in both the time domain and the frequency domain.

by Michael Dethlefsen and John A. Wendler

The HP 71500A microwave transition analyzer is an MMS (Modular Measurement System) instrument. As shown in the idealized block diagram, Fig. 1, it consists of the HP 70820A microwave transition analyzer module and the HP 70004A MMS mainframe and color display. For an explanation of the capabilities and applications of the microwave transition analyzer, see the article on page 48.

The block diagram is relatively straightforward. The two input signals are sampled by microwave sample-and-hold circuits with an input bandwidth of 40 GHz. The sample rate is generated by a low-frequency 10-to-20-MHz synthesizer under processor control. The sampled signals are digitized by an analog-to-digital converter (ADC), the digitized outputs are processed by the digital signal processor, and the final results are displayed on the MMS display by the instrument processor.

The implementation was somewhat more complex than it might appear from the block diagram. While microwave samplers with bandwidths up to 40 GHz were generally available, they were not designed to be used as sample-and-hold circuits operating at rates up to 20 MHz. Low-frequency synthesizers, while also commonly available, did not have the desired phase noise performance. The available high-speed ADCs, if used directly on the sampler output, would have been the primary noise floor and dynamic range limitation of the instrument because of their limited resolution. Digital signal processing is relied upon heavily to achieve and improve much of the basic hardware performance and to extract and display the input signals' characteristics in both the time and frequency domains. However, the general-purpose digital signal processors could not do a significant amount of real-time processing at the 20-MHz data rates, so a large buffer memory was required between the ADC and the digital signal processor.

This article attempts to explain some of the design considerations, in both the hardware and the firmware, that went into the development of the microwave transition analyzer block diagram.

Sampler Operation

Microwave samplers have been used in RF and microwave instrumentation for several decades.[1] They traditionally have been the most economical way to obtain the broadest frequency coverage with the smoothest frequency response.

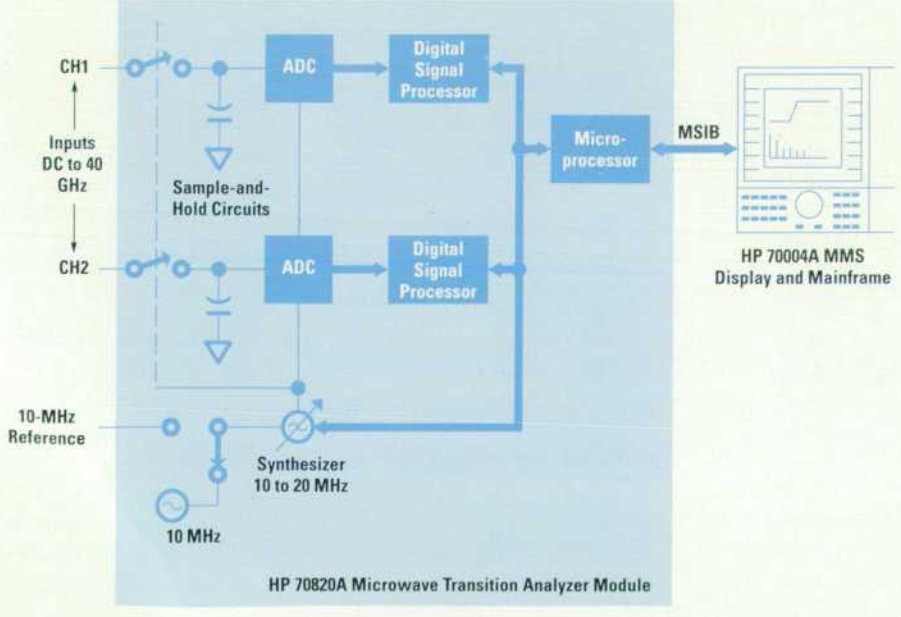


Fig. 1. Idealized block diagram of the HP 71500A microwave transition analyzer.

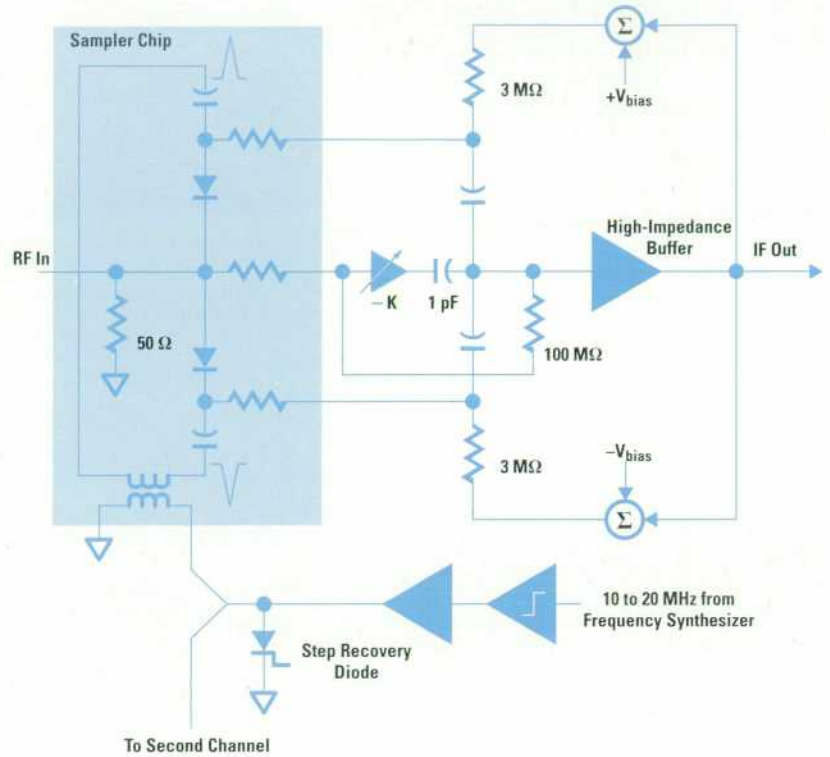


Fig. 2. Simplified diagram of the microwave sampler circuit.

Their noise figure is relatively poor. Their inherent broadband coverage has encouraged their use in frequency acquisition and phase-lock loops as well. However, their ability to capture the time-domain waveform (or the frequency-domain equivalent—simultaneously translating all the harmonics of a repetitive waveform) is what made them so suitable for the 70820A.

The basic concept of a microwave sampler is to generate a very narrow sampling pulse that turns on a series switch between the RF input signal and the IF circuitry, which is mainly a holding capacitor. The amount of time that the switch is on establishes the frequency response of the sampler. If the switch is fully on for 10 ps, the ideal frequency response would be $\text{sinc}(10^{-11}f)$, which has a 3-dB bandwidth of 44 GHz. Although this assumption of a perfectly rectangular switch on-resistance as a function of time is only an ideal, it is a good enough engineering approximation to use here. As shown in Fig. 2, the series switch used in this sampler is an integrated pair of GaAs diodes. The switching waveform is generated by driving a silicon step recovery diode at a variable sample rate between 10 and 20 MHz. The step output of the step recovery diode is split into two signals, one for each channel. The sampler assembly then shapes and differentiates this edge to form a narrow impulse, which briefly turns on the diode switch, allowing some of the RF current to flow into the holding capacitor.

For ideal sample-and-hold circuit operation, the output voltage should only depend on the input voltage during a single sampling instant. Its voltage should not depend on any previous samples or how often the samples are taken. There are two general techniques to achieve this sample-to-sample independence. One is to discharge the holding capacitor fully before each sample and measure the amount of charge or voltage on the hold capacitor after each sample. The other technique is to require that the sample-and-hold circuit capacitor charge to 100 percent of the input voltage during each sample period. The limitations of using the microwave sampler as a high-speed, conventional sample-and-hold circuit now begin to become apparent. At the fast 20-MHz sample rates required, it is not possible to discharge the hold capacitor accurately before each sample. On the other hand, to attain the required microwave input bandwidth, the sampling pulse must be so narrow that it is not possible to charge the hold capacitor fully.

A simplified model of a sample-and-hold circuit and the equations describing its frequency-domain transfer function are shown in Fig. 3. The fraction of the input signal that is stored on the hold capacitor is referred to as the sampler efficiency ϵ, and for this model it can be computed as:

$$\epsilon = 1 - e^{-t_{on}/RC}.$$

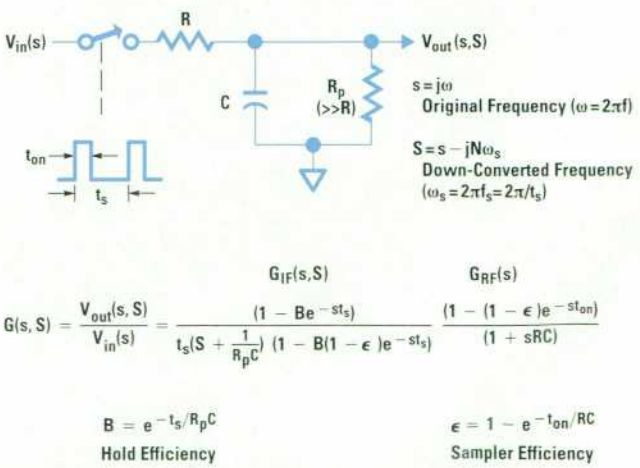


Fig. 3. Sampler model.

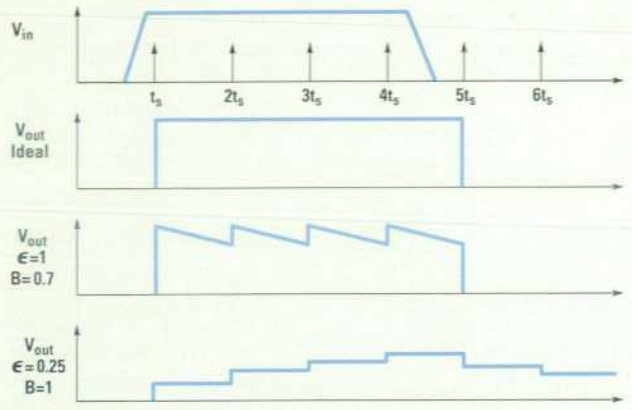


Fig. 4. Sampler time-domain response for different values of sampler efficiency ϵ and hold efficiency B.

100% efficiency, $\epsilon = 1$, would require that the sampler on-time be several RC time constants long, but this would mean an excessively small input bandwidth, as shown by the G_{RF} portion of the sample-and-hold circuit frequency response equation in Fig. 3. Most microwave samplers have relatively low voltage transfer efficiencies, usually significantly less than 10%. With this low sampling efficiency, the resultant voltage on the hold capacitor is a weighted combination of the input voltage from many samples. Sample-to-sample independence is not achieved.

An ideal sample-and-hold circuit also holds its sampled voltage indefinitely until either a reset or the next sample occurs. The voltage droop from one sample to the next is characterized by the hold efficiency B, which can be computed as:

$$B = e^{-t_s/R_pC},$$

where t_s is the sampling period and R_p is the sample-and-hold circuit's load resistance as defined in Fig. 3. 100% hold efficiency means no droop and an infinite load impedance. Fig. 4 shows the time-domain results of sampling a pulse waveform with a sample-and-hold circuit that has ideal characteristics, with reduced sampler efficiency, and with reduced hold efficiency. Fig. 5 plots the G_{IF} portion of the sample-and-hold circuit's frequency response equation for sampler efficiencies of 100%, 10%, and 1%, and for hold efficiencies of 90% and 100%. For low sampler efficiencies like those normally encountered in microwave samplers, the plots in Fig. 5 look very much like a single-pole, low-pass filter. The equations for G_{IF} do indeed simplify and converge, in this case, to a single-pole filter. The efficiency equations become:

$$\epsilon = t_{on}/RC$$

$$B = 1 - t_s/R_pC.$$

The original model then becomes the very commonly used model shown in Fig. 6. The sampler is simply replaced by its time averaged impedance Rt_s/t_{on}.

This characterization of the sampler model points out the main difficulties of using a microwave sampler as a sample-and-hold circuit. The IF output voltage is low-pass filtered and represents an average of many samples of the input voltage. In addition, if the hold efficiency is not very close to 1, even the low-frequency gain will vary with the sampling frequency and the sampler efficiency as shown in Fig. 5. To solve this latter problem, the HP 70820A microwave transition analyzer module uses a very high-impedance buffer on the output of the sampler and provides a positive feedback bootstrap voltage to remove the low-frequency loading effects of the current biasing resistors as shown in Fig. 2. Operating with this high load impedance has the additional benefit of minimizing the sampler compression at high input levels since very little signal current has to flow through the sampler diodes. In addition, since the sampler diodes are effectively current biased instead of voltage biased, their sensitivity to temperature variations is considerably reduced.

The problems created by the sampler low-pass filter effect are more difficult to solve. As the sampling frequency is varied, the current bias is changed by the processor to keep the sampler on-time t_{on} constant. This is required so that the RF frequency response does not vary noticeably with sampling frequency. However, since the sampler time constant is proportional to t_{on}/t_s, the IF bandwidth now varies with sampling frequency. To solve this problem a programmable zero was added following the IF buffer amplifier (see Fig. 7). During the IF calibration process, the sample-and-hold circuit low-pass pole is measured as a function of the sampling frequency. Whenever the sampling frequency is changed, the programmable-zero amplifier is adjusted to cancel the effect of the sampler pole.

Another challenge encountered when using a microwave sampler as a sample-and-hold circuit is its feedthrough capacitance. A capacitance as low as 50 femtofarads between the RF input and IF output will cause significant errors in the expected operation of the microwave sampler. Signals below 10 MHz will directly couple into the IF even when the sampler is supposed to be off. To cancel this effect, the input signal is tapped off before the sampler diodes, inverted, and capacitively summed back into the IF signal.

The IF output of the sampler is ac coupled. When the instrument is dc coupled, the dc component is restored by picking it off before the sampler diodes and summing it back in at the IF buffer stage. The crossover frequency is about 3 Hz.

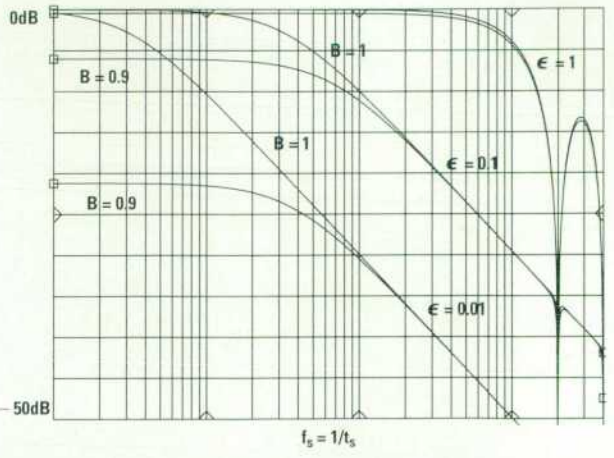


Fig. 5. Sampler IF frequency response for different values of sampler efficiency ϵ and hold efficiency B.

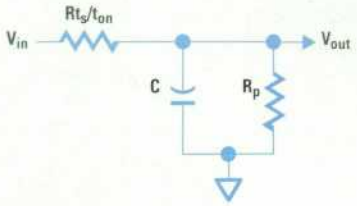


Fig. 6. Simplified sampler model.

IF and ADC Operation

Now that the signal has been sampled and the sampler pole effect has been canceled, the IF signal can be processed and digitized. The IF processing block diagram is shown in Fig. 7. The ADC used in the HP 70820A is a 10-bit device, operating at the same frequency as the input sampler. This 10-bit resolution does not provide enough dynamic range for many of the measurements performed by the microwave transition analyzer. For example, network analysis measurements can be performed over a greater-than-100-dB range and time-domain waveforms of 1 mV full scale can be captured without requiring trace-to-trace averaging. To achieve this dynamic range improvement, step gains are placed in the IF signal path. Up to 60 dB of gain in 6-dB steps can be switched in, either autoranged or manually controlled by the user. This means that even low-level signals can use the full range and accuracy of the ADC. To allow gain to be used even in the presence of a large dc signal, a dc offset DAC is added ahead of the step gains as shown in Fig. 7. This allows up to ±420 mV of offset to be applied to the IF signal before the step gains. The dc offset capability does not affect the allowed input signal range. It must be kept less than 420 mV peak to avoid sampler compression.

The total noise present in the IF may mask the input signal and limit the amount of step gain that can be used without overranging the ADC. This noise is there because the sampler translates the entire 40-GHz bandwidth into the IF frequency range. The programmable-zero amplifier also adds a lot of high-frequency amplification to the sampler and the IF buffer noise floor. All of this noise needs to be minimized. It is also highly desirable to remove any harmonics of the sampling LO signal and signals centered around them. To solve these problems, switchable low-pass IF filters are used. These include a 10-MHz filter for sampling rates between 14 and 20 MHz and a 7-MHz filter for sampling rates less than 14 MHz. In addition, a 100-kHz analog noise filter can be switched in to provide a greater-than-20-dB reduction in total noise. Since this noise filtering is done in real time, it provides faster signal-to-noise ratio improvement than the digital signal processor-based alternatives.

As described in the article on page 48, the IF signal is a time-scaled version of the original RF signal when the instrument is operating in the standard, repetitive sampling mode. Therefore, triggering information can be obtained from the IF signal. Since the signal is at a much lower frequency and is potentially amplified and filtered, the trigger circuitry is cheaper to implement and more accurate than a trigger circuit operating directly on the microwave signals. This allows the microwave transition analyzer to trigger internally on very low-level periodic signals anywhere in its microwave frequency range.

Once the IF signal has been filtered, offset, and amplified, it is ready to be digitized. The ADC is a commercially available, two-pass, 10-bit ADC and the required external sample-and-hold circuit is implemented with a discrete design. The sample-and-hold circuit and ADC are driven at the same frequency as the microwave input sampler. The digitized signal is stored into the 256K-sample ADC memory buffer for further digital signal processing.

IF Corrections

Fig. 8 shows a representation of the spectrum of the analog IF signal for a sample rate of $f_s = 1/t_s$. The ideal sampling operation creates a spectrum that is replicated every f_s so the spectral component at $f_2 = f_s - f_1$ is the complex conjugate of the ideal spectral component at f_1. The IF processing, including the hold operation of the microwave input sampler and the low-pass filters, provides a different amount of attenuation and phase shift at the IF frequency f_2 than at frequency f_1. This is signified by the G(f) transfer function in Fig. 8. When the ADC sample-and-hold circuit resamples the IF signal, the spectral component at f_2 will be aliased or folded onto the same frequency as f_1. It is not possible to build a perfect anti-aliasing filter that will totally eliminate f_2, even at a fixed sample frequency of 20 MHz, and in this application, where the sample rate is continuously variable between 10 and 20 MHz, there will be significant

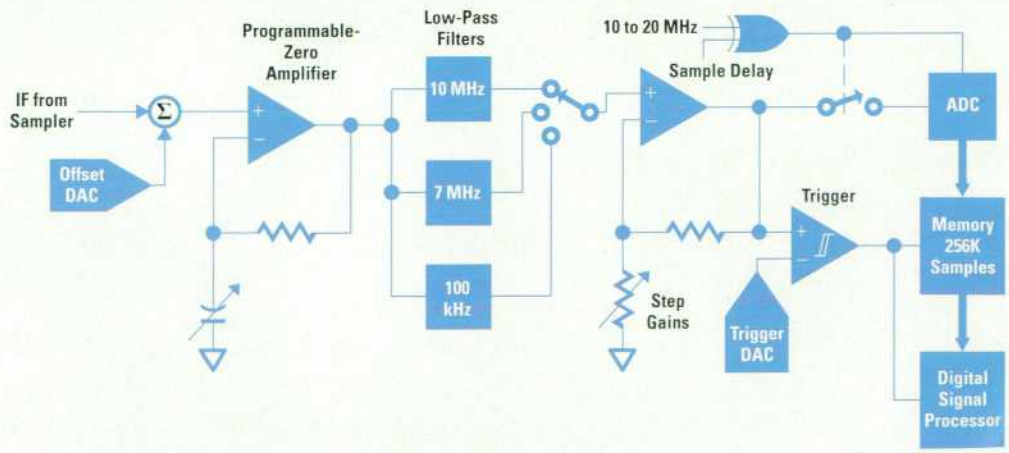
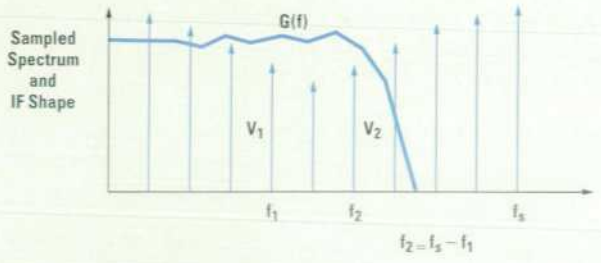


Fig. 7. Microwave transition analyzer IF block diagram.



Because of Front-End Sampling Operation,

$$V_2(f_2) = V_1^*(f_1)$$

Because of IF Shape and LO Delays,

$$V_{IF2}(f_2) = V_2(f_2)\,G(f_2)$$
$$V_{IF1}(f_1) = V_1(f_1)\,G(f_1)$$

Because of Folding in ADC Sample-and-Hold Circuit,

$$V_{ADC}(f_1) = V_{IF1}(f_1) + V_{IF2}^*(f_2)$$

$$V_{ADC}(f_1) = V_1(f_1)\,\underbrace{[G(f_1) + G^*(f_s - f_1)]}_{\text{Folded IF Response}}$$

Fig. 8. Spectrum folding in the microwave transition analyzer.

aliasing. However, since the filtered IF signal was originally a sampled signal, the relationship between the original aliased spectrum and the unaliased spectrum is known:

$$V_2 = V_1^*.$$

Therefore, the original spectrum can be computed if the folded IF response ($G(f_1) + G^*(f_s - f_1)$) can be determined. The folded frequency response varies with f_s, and f_s can be any value between 10 and 20 MHz. Therefore, the IF cannot realistically be calibrated just by measuring the folded IF response, since there are an almost unlimited number of different responses possible. Instead, the unfolded frequency response G(f) must be determined and then the folded response can be computed based on the present value of f_s. Determining the folded IF response is the major requirement for the digital signal processor-based IF corrections in the microwave transition analyzer.

Many things contribute to the overall IF frequency response. In addition to the flatness of the IF buffer amplifier, the programmable-zero amplifier, and any nonideal cancellation of the sampler pole, all possible combinations of analog filters and step gains must be characterized. For example, the relatively high-order filters may have an amplitude response flatness of ±2 dB and some very significant group delay variation which creates considerable ringing in their step response. To measure these, the microwave transition analyzer generates a calibration signal. The calibration signal is a precisely known square wave that is connected by the user to the input port. The frequency and amplitude of the calibration signal are adjusted and varied as required during the IF calibration process. This calibration signal is only used for IF calibration and verification. The rise and fall time requirements on the calibration signal are governed by the requirement that it settle in less than 50 ns, so it is not useful for verifying or calibrating the RF frequency response of the input.

The IF frequency response must be measured in an alias-free fashion and at frequencies higher than 10 MHz. This cannot be done with just a 20-MHz maximum sample rate. The exclusive-OR control shown in Fig. 7, which inverts and delays the ADC clock, helps solve this problem. By first measuring the calibration signal with a normal 20-MHz clock, and then remeasuring the same signal with the inverted clock, which delays the ADC sample by 25 ns, an effective 40-MHz sample rate is achieved after the two measurements are interleaved. In this way, the frequency responses, both magnitude and phase, of the IF path, the step gains, and the switched filters are all determined. In some cases the measured data is used directly in the correction process. In other cases, such as for the step gains, better results are achieved by fitting the measured data to a model and then computing the extrapolated frequency response from the model.

The other critical parameter that must be included in the IF frequency response is the delay between the microwave input sampler and the ADC sample-and-hold circuit. This must include both the IF signal delay and the delay in the LO clock paths. Since this delay is not constant with sample frequency, it must be characterized as a function of the sample frequency. A significant portion of the IF calibration time is spent doing this characterization. This involves measuring the group delay of the harmonics of the calibration signal at different sample frequencies.

Once the unfolded frequency response and the delay have been measured, the folded frequency response can be computed for a given sample frequency. However, since the V_1 and V_2 spectral components can have very similar amplitudes, it may turn out that the folded frequency response has a very deep null in it, depending on the phase relationships. An excessively deep null cannot be properly corrected, for both noise and stability reasons. When this occurs, the firmware in the microwave transition analyzer must change the delay relationship between the IF signal and the ADC clock. This can be done with either the ADC clock invert/delay control or by using a different 20-MHz analog filter in the signal path. The firmware determines which of the possible combinations results in the best possible folded frequency response.

Once the IF calibration process has been completed, the data is stored in battery backed-up RAM. Whenever the sample frequency or IF gain is changed, the microwave transition analyzer firmware recomputes the folded frequency response of the IF. This folded response is then inverted and applied to the digitized input data using either an FFT (fast Fourier transform) operation or a 64-point FIR (finite impulse response) digital filtering operation, depending on the measurement mode. The result is that the IF looks as if it has a flat response all the way to the Nyquist frequency ($f_s/2$). Therefore, the microwave sampler appears to have the ideal impulse and step response expected of a true sample-and-hold circuit.

Sample Rate Synthesizer

The requirements on the sample-rate synthesizer in the microwave transition analyzer are quite stringent. Not only must it have a frequency resolution less than 0.001 Hz over a 10-to-20-MHz frequency range, but it must also be able to phase-lock to a common 10-MHz reference and be capable of shifting the phase of the synthesized output with less than 0.001-degree resolution. Fortunately, this type of source, using fractional-N synthesis techniques,[2] had been used in earlier HP instruments and could be efficiently leveraged.

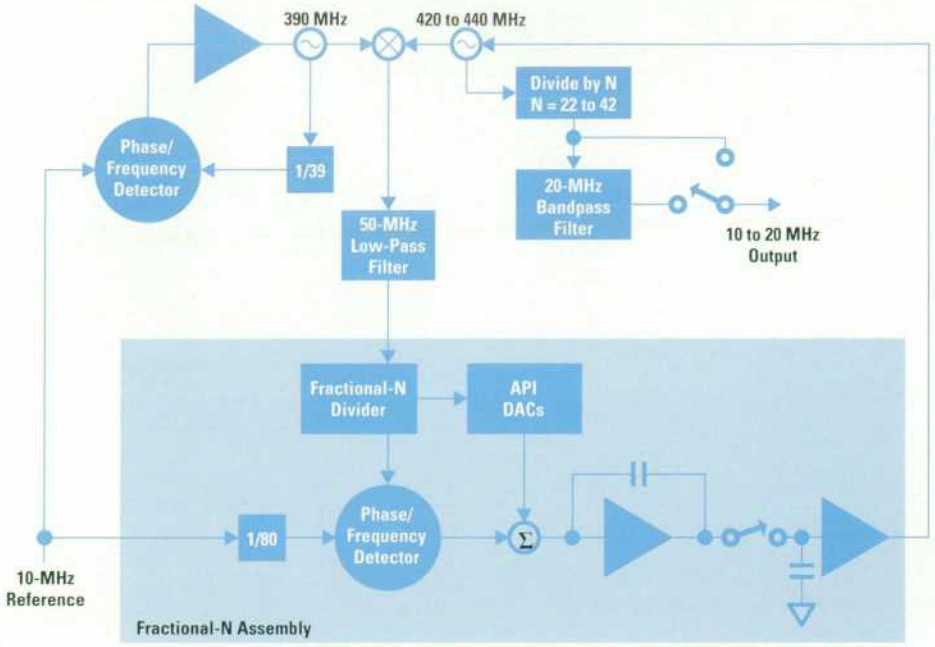


Fig. 9. Block diagram of the 10-to-20-MHz sample rate synthesizer. API stands for analog phase interpolation.

The most stringent requirement for the source was its jitter, or equivalent phase noise, but the available implementations had inadequate performance. Both the close-in and the broadband phase noise of the synthesizer are important. For example, just based on the maximum slope of a full-scale 40-GHz sine wave, jitter of 7 femtoseconds on the sampler LO signal generates additional noise greater than one least-significant bit of the ADC. Since the fundamental of the RF waveforms can be mixed to as low as 100 Hz in the IF, minimizing the close-in noise and spurious components of the sample-rate synthesizer is critical to avoiding low-frequency perturbations and distortion of the digitized signal.

To improve the basic performance of the available fractional-N synthesizers while still leveraging much of the previous engineering effort and available integrated circuits, a translate loop was added to the normal synthesizer block diagram. As seen in Fig. 9, instead of having the loop oscillator operate directly over the normal 30-to-50-MHz band, a 420-to-440-MHz oscillator is mixed with a 390-MHz reference oscillator. The mixer output, 30 to 50 MHz, is the input to the leveraged fractional-N assembly, which does the fractional division, phase detection, and interpolated phase correction, and generates the tuning voltage to lock the 440-MHz oscillator. A second output of the 440-MHz oscillator is fed to a programmable integer divider to generate the 10-to-20-MHz output.

There was no requirement for this synthesizer to sweep continuously over the 10-to-20-MHz range. This translate-and-divide-down block diagram allows the performance of the overall synthesizer to be improved from the original design by a factor equal to the integer divide number, or more than 26 dB. To improve the broadband phase noise further, a 200-kHz-wide bandpass filter is switched in just before the step recovery diode driver whenever the synthesizer is within the 19.8-to-20-MHz frequency range. In the majority of the measurement modes, the synthesizer is set very close to 20 MHz, so this bandpass filter is normally used. While it was not possible to achieve the 7-fs performance number, this combination of improvements reduces the jitter contribution of the synthesizer to less than 1 ps.

RF Filtering

Because the microwave transition analyzer digitizes waveforms with a continuous and extremely precise time axis, it becomes feasible to apply digital filtering functions to these waveforms. These filters can be used to simulate the adding of a hardware filter to the system, to improve the signal-to-noise performance, to remove undesired harmonics and spurious frequency components, and to compensate for non-ideal microwave frequency response effects in the RF circuitry, cabling, probes, and test fixtures, which inevitably degrade the system bandwidth. This ability to correct for RF frequency response roll-off is also used within the instrument to flatten the frequency response of both the samplers and the internal RF cabling to 40 GHz.

Two filters can be defined by the user, one for each of the two input channels. These filters are specified by defining the magnitude and phase response at up to 128 arbitrarily spaced frequency points. The type of interpolation to be used between these frequencies can be specified as flat, linear, or logarithmic. These user-defined filters are combined with the instrument's own RF correction data to generate the composite filter function that is applied to the digitized signal. Regardless of whether the sampler is being used for frequency translation or frequency compression or a combination of the two, there is a unique mapping between the input RF frequency and the IF frequency. This means that the desired RF filtering can indeed be performed by scaling and translating the filters' frequency axis, based on the current time span and carrier frequency, into the IF band and performing the filtering on the digitized IF signal.

There are some modes of sampler operation in which the RF waveform is not replicated in the IF, so there is no unique RF-to-IF frequency mapping and RF filtering cannot be performed. An example of this mode of operation would be when triggering on the clock frequency while sampling a

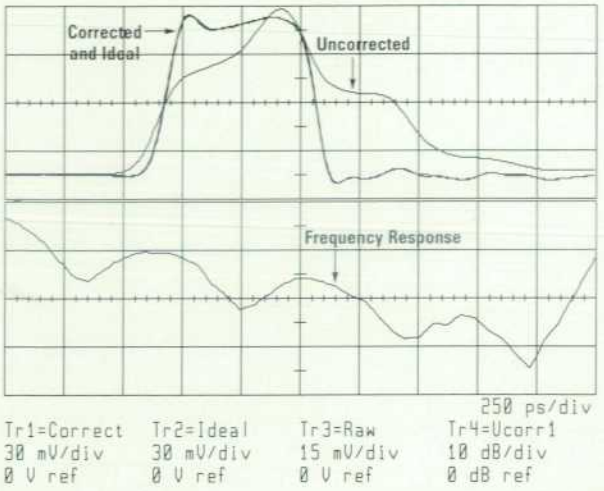


Fig. 10. An example of a measurement with user filter corrections. The frequency response of a cable fixture was measured (bottom trace), stored, and used to correct a measurement of an 800-ps pulse that was made using the fixture. The corrected trace measured with the fixture is almost identical to the ideal trace measured without the fixture. Because the ideal and corrected traces are almost identical, they are indistinguishable in this figure.

random data sequence to create an eye diagram. User filtering, including the internal RF corrections to 40 GHz, is not valid in this mode so the feature must be turned off. If, however, the data sequence is actually a pseudorandom data sequence and the sample rate of the microwave transition analyzer is adjusted to correspond to the pattern repetition rate instead of the clock rate, then the RF waveform is replicated in the IF, and RF filtering and corrections can be performed, even while triggering on the clock to make eye diagram measurements.

The user filter and correction data can be generated in several ways. Direct entry of values is one alternative that is useful for simple, rectangular filters. The frequency-domain data for the RF filter can also be generated from any of the four traces. This means that mathematical filters can be defined using the built-in trace math and stored to the RF filter. More important, it means that measurements can be used to create the RF filter or correction data. For example, the microwave transition analyzer, in conjunction with an appropriate source, can be used to measure a transfer function of a microwave test fixture. This transfer function can be stored to the user correction table, and the inverse of this transfer function can then be used as the RF filter to be applied to the time-domain data. This filtering (deconvolution) removes the frequency response effect of the test fixture from the time-domain waveform of interest. The transfer function measurement can be done in frequency sweep mode with a CW source stepped over the frequency range of interest like a conventional network analyzer, or the transfer function can be measured with a wide-bandwidth step or impulse source.

An example of applying the user correction filter is shown in Fig. 10. First the frequency response transfer function of a cable fixture was determined by measuring a 400-ps pulse both before and after it went through the cable fixture. The FFT of these two pulse trains was computed to determine their frequency spectrums. The ratio of these two spectrums was taken and stored to the user correction filter as the cable fixture's frequency response. This is shown on the bottom trace in Fig. 10 out to a frequency of 8 GHz. A test pulse of of 800 ps was then used. The upper raw trace shows the pulse distortion caused by the cable fixture. After user corrections were turned on, the corrected trace was generated, which lies almost directly on the ideal trace. The ideal trace was established by measuring the pulse directly out of the pulse generator before it was connected to the cable fixture. The time-domain traces are showing a half cycle of a 200-MHz pulse train.

There are some limitations on the ability of the microwave transition analyzer to apply RF filtering and correction to time-domain waveforms. First, the IF waveform must be a valid representation of the RF waveform. If there are nonharmonically related, spurious, or random signals present, they will be mixed into the IF but will not appear at the correct frequencies, so incorrect filter values will be applied to them. Second, the signal must be sampled with fine enough equivalent time resolution to avoid aliasing any significant harmonics or sidebands. This is just the Nyquist criterion, which says that a signal must be sampled at a rate greater than twice its single-sided bandwidth. This applies to both real-time, single-shot sampling and repetitive, equivalent time sampling. Any aliased components will appear at the incorrect frequencies and be improperly filtered.

Third, because limited time records are captured and processed with the FFT, totally arbitrary filter shapes cannot, in general, be precisely accommodated. For example, a filter shape with a 2-μs transient response would require that 2 μs of an arbitrary input be digitized to generate even the first output point. A desired time resolution of 1 ps would require two million data samples and a digital filter with effectively two million coefficient taps. There are two methods available to minimize this limitation. If an integer number of cycles of the input are used in the FFT processing, then arbitrary filter shapes can be precisely handled. The circular convolution performed by the FFT is, in this case, exactly equivalent to the desired linear convolution. To make this more practical to the user, a cycles mode is available in which the time span can be set in terms of cycles of the fundamental instead of seconds per division. This automatically tracks the fundamental frequency as it is changed. The microwave transition analyzer oversweeps the time-domain span up to the next largest power-of-two trace size. For example, if a 0.5-cycle time span is specified with a trace point size of 512, then one full cycle of the waveform is digitized and a 1024-point FFT is used.

The second method of minimizing the effect of limited time records is to make sure that the filter's transient response is shorter than the minimum displayed time span to be used. If the waveform used in the FFT does not contain an integer number of signal periods, there will potentially be edge effects because of the combination of the discontinuity at the edges of the time record and the filter's transient response. Since the instrument oversweeps up to a factor of two, the edge effects will not be part of the displayed waveform as long as the filter has a transient response shorter than the displayed time span.

A fourth limitation on the ability of the microwave transition analyzer to apply RF filtering and correction is that deconvolution cannot be performed over a wide dynamic range. For example, if the test fixture has an upper frequency roll-off or a narrowband notch of 50 dB, correcting for this would require that the microwave transition analyzer effectively apply 50 dB of gain at these frequencies. This much gain on the noise components creates an excessively noisy and unstable waveform trace. In general, 20 to 30 dB is about the largest amount of frequency dependent amplification that can be used while maintaining an acceptable waveform with a reasonable amount of averaging and reasonable stability in the test fixture. When creating the transfer function to be used in the deconvolution, the attenuation should be rolled off once the transfer function has dropped below the acceptable limit. This also effectively low-pass filters the signal at frequencies above which deconvolution can no longer be done. This is seen in the last division of the user correction trace shown in Fig. 10, where the frequency response is being tapered from –35 dB to +20 dB at 10 GHz. Keeping the value of –35 dB to the default 100-GHz maximum frequency would have resulted in excessive noise because of the +35 dB of broadband gain that this would have created.

Analytic Signal

Many of the primary applications for the microwave transition analyzer result from its built-in capability to easily display the instantaneous magnitude and phase envelope of the input signal. This capability is implemented by creating the quadrature function using the Hilbert transform. This quadrature function is combined with the original data to form a complex-valued representation of the waveform called the *analytic signal*. Just as complex-valued phasor representations simplify the analysis of linear circuits, the analytic signal simplifies the manipulation and analysis of modulated waveforms.

Assume a signal can be mathematically expressed as

$$x(t) = a(t)\cos(\omega_c t + \phi(t)),$$

where ω_c is the carrier frequency and $a(t)$ and $\phi(t)$ are the amplitude and phase modulation functions.

Now create the complex function,

$$x_a(t) = x(t) - jx_{hi}(t),$$

where $x_{hi}(t)$ is the Hilbert transform of $x(t)$. The Hilbert transform is defined as a convolution of the the signal with $-1/\pi t$. The frequency-domain relationship is simply:

$$X_{hi}(f) = jX(f)\text{sign}(f),$$

where $\text{sign}(f) = -1$ for $f < 0$, 0 for $f = 0$, and 1 for $f > 0$. Thus,

$$\begin{aligned} X_a(f) &= X(f)(1 + \text{sign}(f)) \\ &= 2X(f) \text{ for } f > 0 \\ &= X(f) \text{ for } f = 0 \\ &= 0 \text{ for } f < 0. \end{aligned}$$

This is simply the positive spectrum of $x(t)$.

If the modulation bandwidth is less than the carrier frequency (i.e., the modulation spectrum doesn't wrap around dc), then

$$x_a(t) = a(t)e^{(j\omega_c t + \phi(t))}$$

and $a(t)$ and $\phi(t)$ can be simply calculated from the magnitude and phase of the analytic signal $x_a(t)$:

$$a(t) = \sqrt{(x(t)^2 + x_{hi}(t)^2)}$$

and

$$\phi(t) = -\tan^{-1}(x_{hi}(t)/x(t)) - (\omega_c t).$$

Another way to view the Hilbert transform is as a phase shifter that shifts the phase of each input frequency component by 90 degrees, leaving the magnitude unaffected. The resulting signal is said to be in phase quadrature with the original. This is sometimes done with analog phase shifters in radar and receiver systems. Very accurate, wideband results become much easier to achieve, however, with digital signal processing, using either the FFT as shown above, or time-domain FIR filters.

The analytic signal representation is particularly useful in the measurement of pulsed RF signals. In these applications it is often the characteristics of the magnitude or phase as a function of time that are of interest. These features of the signal can be obtained by simply converting the analytic representation to polar format and displaying the desired quantity. Traces of phase as a function of time can be numerically differentiated to display frequency as a function of time. A linear slope corresponding to the carrier frequency can be mathematically removed leaving a display of the carrier's phase modulation as a function of time. As described here, these are single-channel measurements where the carrier phase is measured relative to the fixed-frequency sampling pulse. Measuring the phase with respect to a CW reference is also possible, using both input channels.

Two-channel operation suggests another valuable use of the analytic representation: vector normalization of traces. Many times it is desired to measure the magnitude and phase response of a particular device under a time-varying stimulus, such as pulsed RF. In this case, a two-channel measurement is needed in which the time response at the output of the device is divided (in vector fashion) by the response at the input. Also, to support the measurements typically found in conventional network analyzer systems, a complex (vector) representation of trace data is mandatory.

Since the Hilbert transform is a filtering process, it has the same limitations and constraints as previously discussed for RF filtering. In addition, it has the constraint that the signal must correspond to the basic model of a single carrier frequency with modulation sidebands. For example, baseband signals with a fundamental and many harmonics do not meet this criterion, and the analytic operator has little, if any, useful meaning. The user can disable the analytic operator in this mode to achieve some increased performance speed. The Hilbert transform performs best if the carrier is located in the center of the IF bandwidth. This provides the maximum amount of double-sided modulation bandwidth, and keeps the carrier the maximum distance from dc or the Nyquist frequency where inaccuracies in the Hilbert transform are the greatest. The internal implementation of the Hilbert transform generally uses the positive-side FFT technique, with some additional enhancements using windowing and unwindowing operations to minimize the time record edge discontinuities.

Sampling at Slower Rates

As mentioned earlier, the sampling rate synthesizer operates between 10 and 20 MHz. Normally, the sampling rate must be set very close to the input signal's fundamental frequency or pulse repetition frequency or a subharmonic thereof. This means that signal frequencies less than 10 MHz cannot be acquired in the normal manner. If the hardware sampling rate is set to a multiple of the desired rate, then simply keeping one out of every N points would result in the desired sampling rate. Unfortunately, complications arise for triggering and providing IF correction. Because more than one sample is taken during each period of the input signal, the IF signal presented to the ADC is not an accurate replica of the input signal, with or without IF correction. For this reason, triggering on the IF signal is no longer adequate for proper operation. What is required is a digital triggering operation using only the samples corresponding to the final (decimated) sampling rate. To correct the IF response and ensure that the retained sample is an accurate measure of the input signal voltage, the influence of the samples leading up to the desired one needs to be removed. This can only be achieved by processing the original samples obtained at the hardware sampling rate and correcting the desired sample before discarding the remaining N–1 samples.

Many samples may have to be processed before a software trigger is detected. Consequently, the processing involved in applying IF corrections to the retained samples needs to be done in real time, that is, the correction process needs to operate in parallel with the data collection at a rate that is sustainable over an indefinite period. This is accomplished by designing the ADC memory to be a dual-ported circular buffer so that one segment of memory can be worked on while another is being written to by the ADC. As samples are collected at the 10-to-20-MHz rate in another part of memory, a finite impulse response (FIR) IF correction filter of up to 64 taps is applied to the block of samples just collected and a single corrected output sample is produced. The digital signal processor is able to produce corrected samples at a maximum rate of about 10 kHz. This then becomes the maximum sampling rate in the new mode of operation. The minimum sampling rate is 1 Hz.

Input signals with a fundamental repetition frequency less than 10 kHz are sampled once per cycle. The sample rate synthesizer is set somewhere between 10 and 20 MHz, N is determined, and 1 of N hardware samples are corrected and retained. The process is the same for input repetition frequencies between 10 kHz and 10 MHz except that multiple input periods occur between each retained sample, keeping the output rate less than 10 kHz.

Conclusion

By combining and enhancing the basic analog and digital hardware blocks with the flexibility of digital signal processing, the HP 70820A microwave transition analyzer team was able to implement a fundamentally simple microwave data acquisition instrument that is capable of making a wide variety of time-domain and frequency-domain measurements on microwave signals and components. Not only does this provide new versatility previously unavailable in a single instrument on both CW and complex time-varying signals, but it also provides new functionality to the microwave designer—for example, in looking at extremely fast transitions on periodically pulsed signals.

Acknowledgments

The development of the HP 70820A microwave transition analyzer module, given its new approach to making such a wide variety of measurements, required the work, contributions, and advice of a large number of people as its definition and functionality evolved over time. Mike Marzalek and Ron Hogan pioneered many of the initial hardware concepts. Steve Peterson helped develop the digital signal processor processing algorithms, user interface, and firmware functionality. Mechanical product design by Jim Tranchina and major hardware design efforts by Matt Fowler, Dave Ballo, and Jim Jensen brought the product to its final production quality status.

References

1. W.M. Grove, "A dc-to-12.4-GHz Feedthrough Sampler for Oscilloscopes and other RF Systems," *Hewlett-Packard Journal*, Vol. 18, no. 2, October 1966, pp. 12-15.

2. D.D. Danielson and S.E. Froseth, "A Synthesized Signal Source with Function Generator Capabilities," *Hewlett-Packard Journal*, Vol. 30, no. 1, January 1979, pp. 18-26.

A Visual Engineering Environment for Test Software Development

Software development for computer-automated testing is dramatically eased by HP VEE, which allows a problem to be expressed on the computer using the conceptual model most common to the technical user: the block diagram.

by Douglas C. Beethe and William L. Hunt

For many years, the cost of developing computer-automated testing software has grown while the cost of the computer and instrumentation equipment required to run tests has dropped significantly. Today, in many test systems, the hardware costs represent less than 25% of the total cost of the system and software costs consume the other 75%. HP VEE was designed to dramatically reduce test software development costs by allowing the test to be expressed on the computer using the conceptual model most common to the technical user: the block diagram. This article will provide a general overview of the development of HP VEE, its feature set, and how it applies the concept of the executable block diagram. Further details of the architecture of HP VEE can be found in the articles on pages 78 and 84.

There was a time when business and finance people dreaded using a computer because it meant an extended question-and-answer session with a primitive mainframe application being displayed on a dumb terminal. Even after the first personal computers were introduced, very little changed, since the existing applications were simply rewritten to run on them. When the electronic spreadsheet was developed, business users could finally interact with the computer on their own terms, expressing problems in the ledger language they understood.

The technical community was left out of this story, not because the personal computer was incapable of meeting many of their needs, but because their problems could seldom be expressed well in the rows and columns of a ledger. Their only options, therefore, were to continue to work with the question-and-answer style applications of the past, or to write special-purpose programs in a traditional programming language such as Pascal, C, or BASIC.

Technical people often find it difficult to discuss technical issues without drawing block diagrams on white boards, notebooks, lunch napkins, or anything else at hand. This begins at the university where they are taught to model various phenomena by expressing the basic problem in the form of a block diagram. These block diagrams usually consist of objects or processes that interact with other objects or processes in a predictable manner. Sometimes the nature of the interactions is well-known and many times these interactions must be determined experimentally, but in nearly all cases the common language of expression is the block diagram.

Unfortunately, the task of translating the block diagram on the lunch napkin into some unintelligible computer language is so difficult that most technical people simply cannot extract real value from a computer. Staying up on the learning

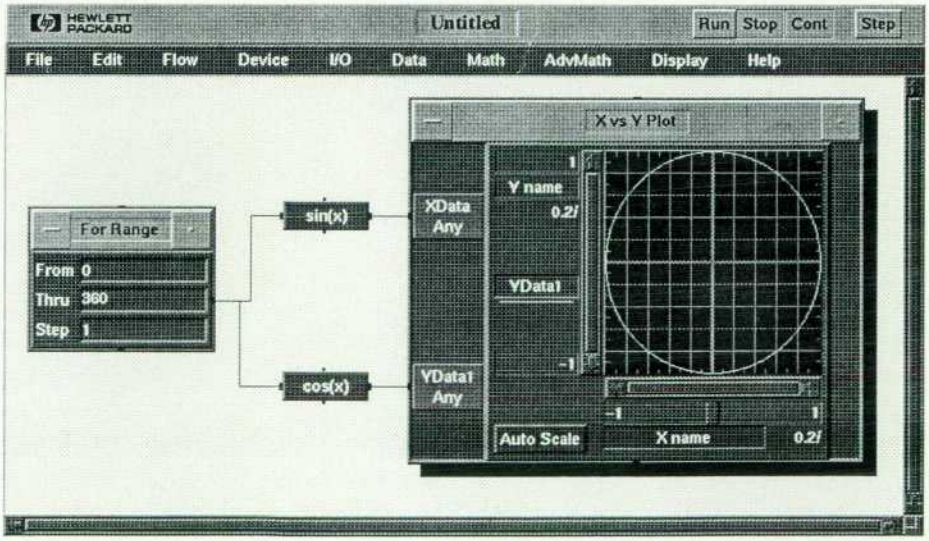


Fig. 1. A simple HP VEE program to draw a circle.

curve of their own problem domain is a sufficient challenge that embracing a whole new learning curve (programming) just to translate problems for the computer's benefit hardly seems worth the effort. While it is true that many wonderful solutions to certain kinds of problems have been generated over the years, most of the potential usefulness of computers has never been realized. In many cases, a good calculator is still the best bet because it makes a manual solution relatively easy to compute.

What is HP VEE?

HP VEE, Hewlett-Packard's visual engineering environment, is a software tool that allows users to create solutions by linking visual objects (icons) into block diagrams, rather than by using traditional textual programming statements. HP VEE provides objects for data collection, analysis, and presentation, in addition to objects and features for data storage, flow, modularity, debugging, documenting, and creating graphical user interfaces. The objects work together in the form of an interconnected network or block diagram constructed by the user to represent the problem at hand. The user selects the necessary objects from the menu, links them in the manner that represents how data flows from one object to another, and then executes the resulting block diagram. No translation to some other language. No other intermediate step.

To understand HP VEE better, consider a simple graphical program to draw a circle. By connecting a loop box, two math boxes (sin and cos), and a graph, this simple program can be built in less than one minute (Fig. 1). Although this is not a difficult task using a traditional language that has support for graphics, it is still likely that it will be quicker to develop it using HP VEE.

HP VEE eases the complexity of data typing by providing objects that can generate and interpret a variety of data types in a number of shapes. For example, the virtual function generator object generates a waveform data type, which is just an array of real numbers plus the time-base information. It can be displayed on a graph simply by connecting its output to the graph object. If its output is connected to a special graph object called a MagSpec (magnitude spectrum) graph, an automatic FFT (fast Fourier transform) is performed on the waveform (Fig. 2). By connecting a noise generator through an add box, random noise can be injected into this virtual signal (Fig. 3). If we had preferred to add a dc offset to this virtual signal, we could have used a constant box instead of the noise generator.

User panels allow HP VEE programs to be built with advanced graphical user interfaces. They also allow the implementation details to be hidden from the end user. To complete our waveform application, we can add the slider and the graph to the user panel (Fig. 4). We can adjust the presentation of this panel by stretching or moving the panel elements as required for our application.

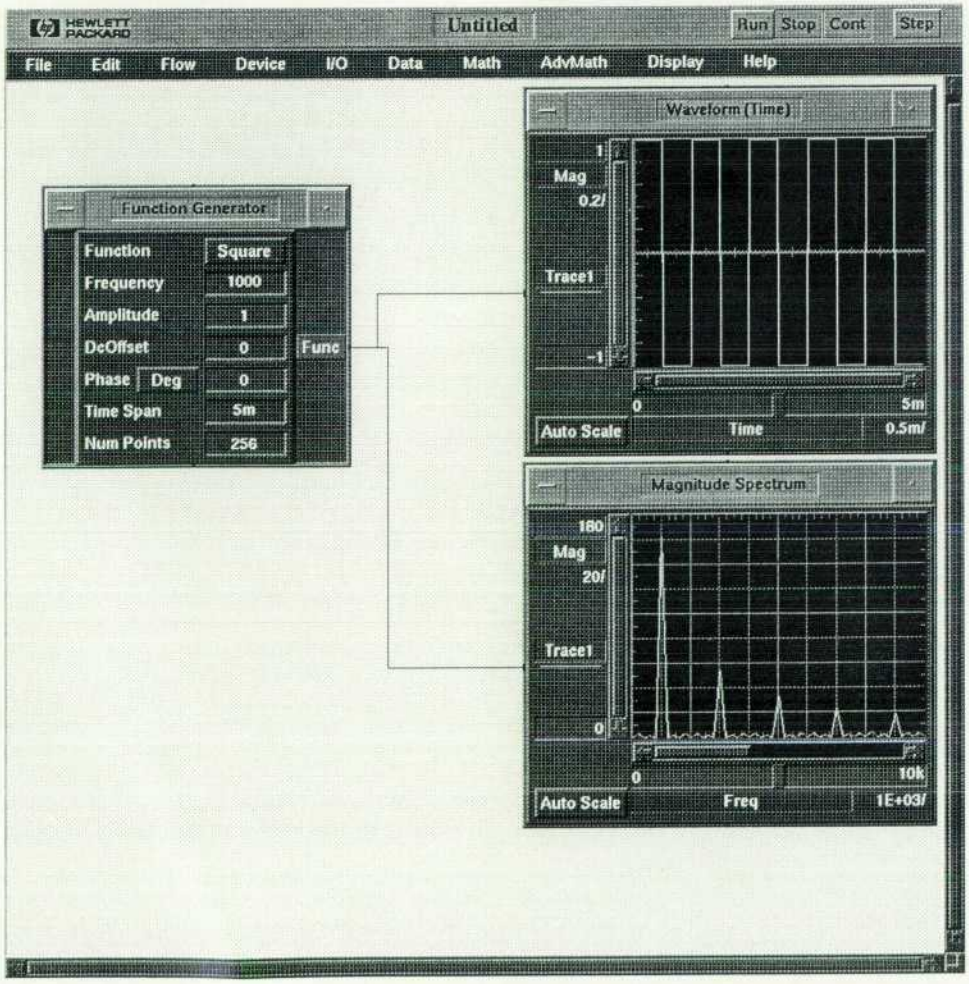


Fig. 2. A waveform displayed in the time and frequency domains.

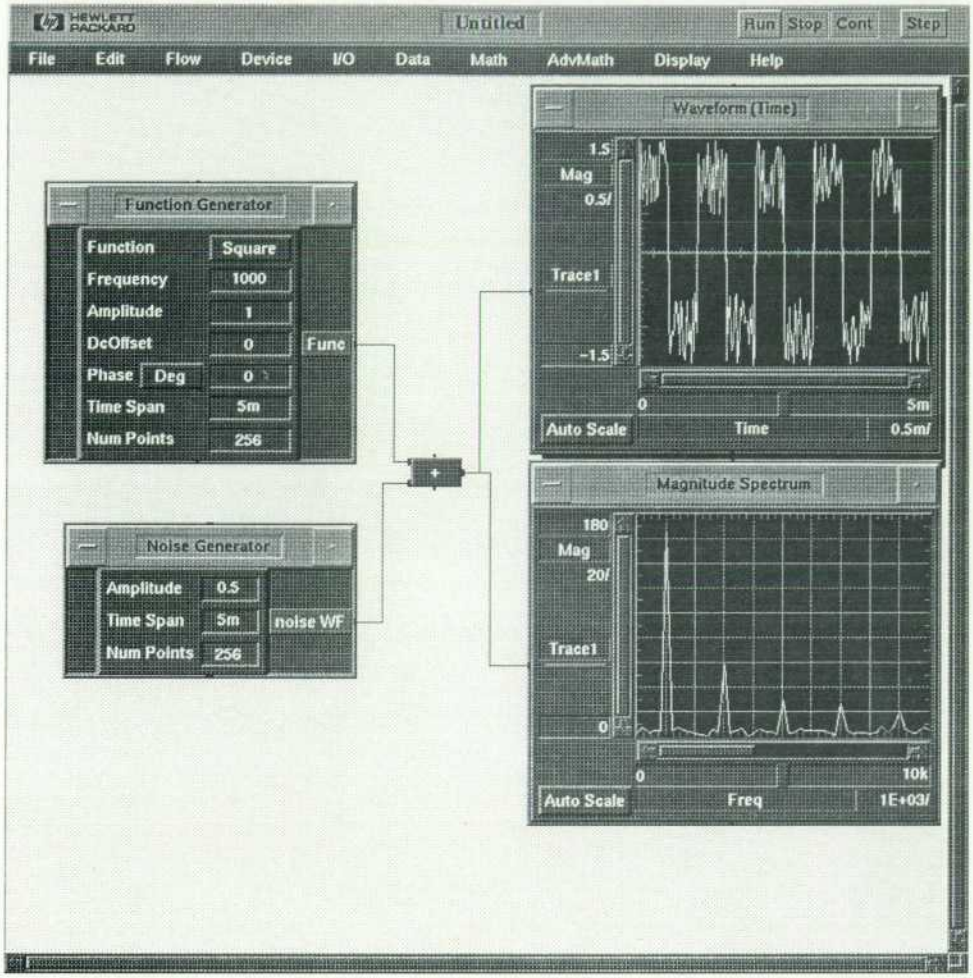


Fig. 3. Noise added to a waveform in the time and frequency domains.

This is just a trivial overview of the basic concept behind HP VEE. Other major features not covered include objects for sending data to and from files, data translation and conversion, advanced math capabilities, and data display functions. HP VEE actually consists of two products. HP VEE-Engine is for the analysis and presentation of data gathered from files or programs or generated mathematically. HP VEE-Test is a superset of HP VEE-Engine and adds objects and capabilities for device I/O and instrument control.

Development Philosophy

The team's goal for HP VEE was a new programming paradigm targeted not only at the casual user, but also at the advanced user solving very complex problems. One simple approach would have been to assign an icon to each statement in a traditional language and present it to the user in a graphical environment. The user would simply create icons (statements) and connect them in a structure similar to a flowchart. However, such a system would be harder to use than a traditional language, since the graphical program would require more display space than the older textual representation and would be more difficult to create, maintain, and modify. This would actually have been a step backward.

We decided that a genuine breakthrough in productivity could only be achieved if we moved to a higher level of abstraction to more closely model the user's problem. We therefore chose to allow users to express their problems as executable block diagrams in which each block contains the functionality of many traditional program statements. Many elements in HP VEE provide functionality that would require entire routines or libraries if the equivalent functionality were implemented using a traditional language. When users can work with larger building blocks, they are freed from worrying about small programming details.

Consider the task of writing data to a file. Most current programming languages require separate statements for opening the file, writing the data, and closing the file. It would have been relatively easy to create a file open object, a file write object, and a file close object in HP VEE. Such an approach would have required at least three objects and their associated connections for even the simplest operation. Instead, we created a single object that handles the open and close steps automatically, and also allows all of the intermediate data operations to be handled in the same box. This single To File box supports the block diagram metaphor because the user's original block diagram would not include open and close steps (unless this user is also a computer programmer), it would only have a box labeled "Append this measurement to the data file." The open and close steps are programming details that are required by traditional programming languages but are not part of the original problem.

Or, consider the task of evaluating mathematical expressions. In some common dataflow solutions, a simple operation such as 2×A+3 would require four objects and their

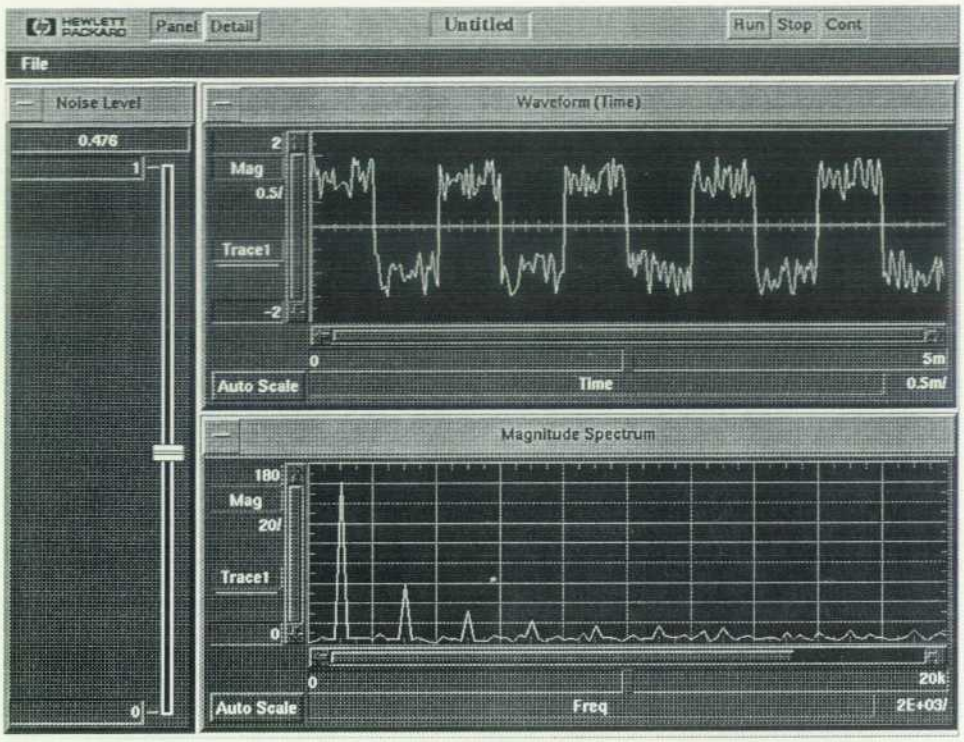


Fig. 4. User panel for waveform plus noise application.

associated connections (two constants, one add operation, and one multiply operation). Using HP VEE's formula box requires only the single expression object to solve this problem. The point of a block diagram is to show an overview of how a complex system operates without regard to implementation details. Had HP VEE been implemented without a higher level of abstraction, the resulting graphical program would have had so many boxes and lines that it would have resembled a maze rather than a block diagram.

Development Process

We followed a fairly informal development lifecycle for HP VEE. It was based on the spiral lifecycle,[1] which divides the development phase into a series of design/build/test cycles with a risk assessment before each. This worked very well for us for several reasons. Probably the most important factor was that the team was small and highly motivated. This made rigorous checkpoints and detailed design documents unnecessary since all of the team members worked very closely together toward the same goals. Another important factor was the use of an object-oriented design approach coupled with very careful design practices. This allowed us to design classes according to their interactions with the rest of the system without spending a great deal of time implementing the internals of the classes. This is important in a spiral lifecycle because during each cycle, an entire class or set of classes may need to be reimplemented. Without an object-oriented approach, this would require an excessive amount of time rewriting other seemingly unrelated parts of the system. Another successful development decision was the early incorporation of a full-time software testing team to help us with the test phases of the lifecycle.

The Search for Primitives

The initial functionality was specified by the team based on our experience as frustrated users of third-generation languages (3GLs) such as Pascal, C, and BASIC. Certain tasks appeared over and over on the "I wish there were some other way to do this ..." list. Experience had already shown that because of limited flexibility, the usual subroutine library approach did not offer the type of productivity increase being sought. However, with our executable block diagram metaphor, we felt that many of these tasks could be implemented as primitives in HP VEE while still providing the necessary flexibility.

Foremost among these tasks were data management, engineering graphics, instrument control, and integration of multiple applications. In each case we were convinced that a higher level of abstraction could be developed that would be flexible yet simple enough to require only minor configuration specification from the user in most situations.

Data Management

To tame the basic data management problem we developed the container architecture. Containers hold data, either arrays or scalars, of a wide variety of data types, and provide a rich set of mathematical intrinsics to operate on that data. Many operations such as type conversion and array processing, formerly left to the user, are incorporated into these object abstractions in a fashion that makes them relatively transparent.

Another aspect of data management involves interfacing with the file system because so much effort must be expended on it when using 3GLs. We developed objects that offer the powerful input/output capabilities of many 3GLs,

Object-Oriented Programming in a Large System

The biggest problem with a large software development effort is that there is just too much complexity for the human mind to manage. The obvious solution is to add more people to the project so that the members are not asked to manage more than their individual abilities permit. Unfortunately, the law of diminishing returns applies, since each additional team member adds a very large communication and training load on the rest of the team. In addition, there are increased opportunities for disagreement and conflict.

In some ways, development of large software systems is like one person trying to dig a canal using only a shovel. Yes, it is possible, but probably not in that person's lifetime. If more people are assigned to the task, it can be done more quickly, but only at an enormous cost. However, if equipped with the right tools (backhoes, earth movers, etc.), one person can accomplish so much that only a small number of people are required to complete the project within a reasonable amount of time.

This is exactly the idea behind object-oriented programming. By reducing the amount of complexity that one software developer must manage, that one person can be responsible for a much larger portion of the system. The result is that much higher productivity is attainable since smaller teams can be used, thereby avoiding the effects of the law of diminishing returns. Features of object-oriented programming such as encapsulation and inheritance allow one person to maintain a much larger portion of a large system than would be possible with a traditional approach.

Encapsulation is probably the strongest reason to use an object-oriented approach for a large system. Object-oriented systems encapsulate functionality by combining data and associated routines into one package (the class) and then disallowing access to the data except through one of the routines. When this is done, code outside of the class is less likely to have dependencies on the structure or meaning of the data in the class since its only access to the data is through the access routines rather than directly to the data. This allows a class to define the externally visible interface separately from the internal implementation. Because of this basic structure, a class or even an entire hierarchy of classes can be completely rewritten without affecting other parts of the system as long as the externally visible interface remains constant.

Inheritance is another reason to use an object-oriented approach in a large system. Inheritance allows a new class to be written simply by specifying additions or changes to an existing class. This means that just a few lines of added code can provide a significant increase in functionality. The other benefit of inheritance is that code reuse of internal routines is increased substantially. For example, there is only one single-line text editor in HP VEE, which is used for all single-line text entry fields. However, since it is easy to add to the behavior of the editor class through inheritance, the numeric fields that allow constant expressions as numeric input are just a very small incremental effort over the original line editor. They simply add to the "accept" mechanism at the end of an editing session and pass the typed string to the parser for evaluation as an expression before attempting to record the numeric result.

However, features such as encapsulation and inheritance do not automatically result in a system that is easier to maintain and build. Very careful design practices must be followed and the team members must be motivated to do high-quality work. Probably the most important design practice is careful partitioning of the system so that complexity in one area is not visible in an unrelated area.

An object-oriented approach coupled with careful design practices will often cause the software development effort to seem harder than with a more traditional approach. For example, in a traditional approach, if a variable in one module needs to be accessed in another module, it is easy to declare that reference directly to the compiler. In an object-oriented approach, it is common for these variables to exist only as instance variables, which are not allocated until the owning class has been instantiated. This means that the compiler cannot reference a value directly because it doesn't exist until run time. Therefore, a more complete solution must be devised to find the required value. This usually means that a message asking for the value must be sent to the object that knows the answer without ever directly accessing the variable. This sounds harder, and it is, but in the long run the resulting code is much more maintainable and extendable.

William L. Hunt
Development Engineer
VXI Systems Division

but present them to the user by means of an interactive dialog box to eliminate the need to remember syntax. Each of these dialog boxes represents a single transaction with the file such as read, write, or rewind, and as many transactions as necessary can be put into a single file I/O object.

Engineering Graphics

For engineering graphics, the task of finding a higher level of abstraction was relatively easy. Unlike data management, engineering graphics is a fundamentally visual operation and as such it is clear that a single element in a block diagram can be used to encapsulate an entire graphical display. Therefore, we just developed the basic framework for each type of graph, and we present these to the user as graph displays that require only minor interactive configuration. In this area we had a rich set of examples to draw from because of the wide variety of highly developed graphs available on HP instruments. In some cases, we were even able to reuse the graphics display code from these instruments.

Instrument Control

Instrument control is a collection of several problems: knowing the commands required to execute specific operations, keeping track of the state of the instrument, and (like file I/O) remembering the elaborate syntax required by 3GLs to format and parse the data sent over the bus. We developed two abstractions to solve these problems: instrument drivers and direct I/O.

Instrument drivers have all of the command syntax for an instrument embedded behind an interactive, onscreen panel. This panel and the driver behind it are developed using a special driver language used by other HP products in addition to HP VEE. With these panels the task of controlling the instrument is reduced to interacting with the onscreen panel in much the same fashion as one interacts with the instrument front panel. This is especially useful for modern card-cage instruments that have no front panel at all. Currently HP provides drivers for more than 200 HP instruments, as well as special applications that can be used to develop panels and drivers for other instruments.

In some situations instrument drivers are not flexible enough or fast enough, or they are simply not available for the required instruments. For these situations, we developed direct I/O. Direct I/O uses transactions similar to the file I/O objects with added capabilities for supporting instrument interface features such as sending HP-IB commands. Direct I/O provides the most flexible way to communicate with instruments because it gives the user control over all of the commands and data being sent across the bus. However, unlike instrument drivers, the user is also required to know the specific commands required to control the instrument.

To simplify the process of reconfiguring an instrument for a different measurement, direct I/O also supports the uploading and downloading of learn strings from and to the instrument. A learn string is the binary image of the current state of an instrument. It can be used to simplify the process of setting up an instrument for a measurement. A typical use of this feature is to configure an instrument for a specific measurement using its front panel and then simply upload that state into HP VEE, where it will be automatically downloaded before making the measurements. Thus, the user is saved from having to learn all of the commands required to initially configure the instrument from a base or reset state before making the measurement. In most cases the user is already familiar with the instrument's front panel.

Multiple Applications

Multiple application integration turned out to be one of the easiest tasks in HP VEE, since the inherent parallelism of multiprocess operations can be expressed directly in a block diagram. Each element of a block diagram must execute only after the elements that provide data for its inputs. However, two elements that do not depend on each other can execute in any order or in parallel. This feature, along with the powerful formatting capabilities provided for interprocess communication, allows the integration and coordination of very disparate applications regardless of whether they exist as several processes on one system or as processes distributed across multiple systems. The only object abstractions required to support these activities are those that act as communication ports to other processes. A pair of objects is available that supports communication with local processes (both child and peer) using formatting capabilities similar to those used by file and instrument I/O.

Finally, we needed to develop objects that would encapsulate several other objects to form some larger user-defined abstraction. This abstraction is available using the user object, which can be used to encapsulate an HP VEE block diagram as a unit. It can have user-defined input and output pins and a user panel, and from the outside it appears to be just like any other primitive object.

Refining the Design

While still in the early cycles of our spiral lifecycle, we sought a limited number of industry partners. This enabled us to incorporate design feedback from target users attempting real problems well before encountering design freezes. Although there were fears that such attempts would slow our development effort because of the additional support time required, we felt that the payback in design refinement for both user interface elements and functional elements was substantial.

One example of such a refinement in the user interface is the automatic line routing feature. Before line routing was added, our early users would often spend half of their time adjusting and readjusting the layouts of their programs. When asked why they spent so much time doing this, they generally were not certain, but felt compelled to do it anyway. We were very concerned about the amount of time being spent because it reduced the potential amount of productivity that could be gained by using HP VEE. Thus we added automatic line routing and a snap grid for easier object alignment so that users would spend less time trying to make their programs look perfect.

An example of a refinement in the functional aspects of the product is the comparator object. Several early users encountered the need to compare some acquired or synthesized waveform against an arbitrary limit or envelope. This task would not have been so difficult except that the boundary values (envelope) rarely contained the same number of points as the test value. Before the comparator was developed, this task required many different objects to perform the interpolation and comparison operations on the waveforms. The comparator was developed to perform all of these operations and generate a simple pass or fail output. In addition, it optionally generates a list of the coordinates of failed points from the test waveform, since many users want to document or display such failures.

Conclusion

Early prototypes of HP VEE were used for a wide variety of technical problems from the control of manufacturing processes to the testing of widely distributed telecommunications networks. Many began exploring it to orchestrate the interaction of other applications, especially where network interconnections were involved.

Current experience suggests that the block diagram form of problem expression and its companion solution by means of dataflow models has wide applicability to problems in many domains: science, engineering, manufacturing, telecommunications, business, education, and many others. Many problems that are difficult to translate to the inline text of third-generation languages such as Pascal or C are easily expressed as block diagrams. Potential users who are experts in their own problem domain, but who have avoided computers in the past, may now be able to extract real value from computers simply because they can express their problems in the more natural language of the block diagram. In addition, large-scale problems that require the expert user to orchestrate many different but related applications involving multiple processes and/or systems can now be handled almost as easily as the simpler problems involving a few variables in a single process.

Acknowledgments

We would like to thank design team members Sue Wolber, Randy Bailey, Ken Colasuonno, and Bill Heinzman, who were responsible for many key features in HP VEE and who patiently reviewed the HP Journal submissions. We would also like to thank Jerry Schneider and John Frieman who pioneered the testing effort and provided many key insights on product features and usability. More than any other we would like to thank David Palermo without whose far-sighted backing through the years we could not have produced this product.

Reference

1. B.W. Boehm, "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, May 1988.

Developing an Advanced User Interface for HP VEE

Simplicity and flexibility were the primary attributes that guided the user interface development. Test programs generated with HP VEE can have the same advanced user interface as HP VEE itself.

by William L. Hunt

HP VEE, Hewlett-Packard's visual engineering environment, was developed as a programming tool for nonprogrammers. In the past, computer users were required to move into the computer's domain. Our goal for HP VEE was to bring the computer into the user's domain. This meant developing a system that operates in the way that our target users expect.

To accomplish this goal, ease of use was of critical importance. However, because most target users of HP VEE are highly educated technical professionals, simple-minded approaches to user interface design were not appropriate. For this audience, the system must be powerful and flexible, but must not become an obstacle because of overprotection.

With these constraints in mind, we decided that the primary attributes of HP VEE should be simplicity and flexibility. Learnability was also considered to be important, but we felt that no one would bother to learn the system unless it were a truly useful and powerful tool. Therefore, we felt that we could compromise some learnability in situations where a great deal of the power of the system would be lost if learnability were our primary goal. Our overall approach, therefore, was to design a system that is as natural to learn and use as possible and powerful enough that our customers would be excited about learning how to use it.

Development Guidelines

In general, simplicity is very important in a user interface because it frees the user from having to worry about unnecessary details or rules. The underlying goal of a good user interface is to increase the communication bandwidth between the computer and the user. However, this does not mean that there should be a myriad of displays and indicators. In fact, quite the opposite is true. The more things there are for the user to comprehend, the greater the chance that something will be missed. The goal, therefore, should be to reduce the amount of data that the user must be aware of and present the essential data in the simplest and most compact way possible. Similarly, any piece of data presented to the user should always be presented in a consistent way because this is known to increase comprehension dramatically.

An example of a simple way to present information to the user is the 3D look used in the OSF/Motif graphical user interface and more recently in other systems such as Microsoft® Windows. When used properly, the 3D borders can be used to communicate information about the state of individual fields without consuming any additional display space. Fig. 1 shows how HP VEE uses the 3D look to identify how fields will respond to user input. Fields that are editable are displayed as recessed or concave. Buttons and other fields that respond to mouse clicks are shown as convex. Fields that are only used as displays and do not respond to input are shown as flat. These states are very simple to comprehend because the three states are unique in the way that they look and operate. Without realizing it, users will naturally learn how to identify which fields are editable, which fields can be activated, and which fields will not respond to input. This 3D display technique allows these states to be displayed without any additional display area.

Fundamentally, HP VEE was designed around the concept of direct manipulation. This means that wherever possible, a setting can be changed by operating directly on the display of that setting. This results in a significant simplification for the user since special operations or commands are not generally required to make changes to settings. For example, the scale of a strip chart is shown near the edges of the graph display (Fig. 2). If the user wants to change the graph scaling, the limit fields themselves can be edited. It is not necessary to make a menu choice to bring up a pop-up dialog box for editing the scale. In many other systems, making any change requires a menu pick. This effectively reduces a system to a command-line interface that happens to use a mouse and menus instead of the keyboard. Such a system is no easier to use than the command line interface systems of the past.

Flexibility is more important for an easy-to-use system than for more traditional systems because there is a perception that power and ease of use cannot be combined in the same system. In the past, powerful systems have generally been

Fig. 1. A view containing a noneditable field, two buttons, and some editable fields.

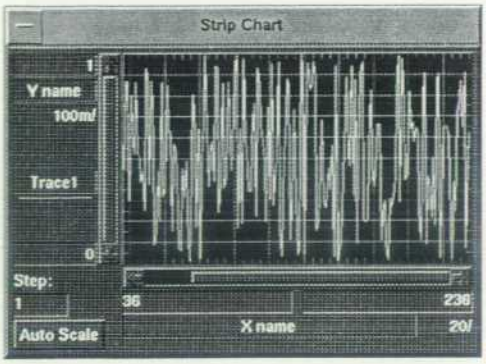


Fig. 2. Direct manipulation is useful for settings such as graph limits.

hard to use, and easy-to-use systems have generally not been very flexible or powerful. To overcome this perception, therefore, an easy-to-use system must be very powerful so that potential customers' fears can be overcome. When designing HP VEE, we were very careful to avoid limiting flexibility wherever possible. It often seemed as if we were faced with a choice between ease of use and flexibility. However, with careful consideration of the alternatives, we usually found that the more flexible approach could be implemented with an easy-to-use interface.

Flexible and powerful systems are naturally very complex because there are so many features and capabilities to remember. In these systems, it is extremely important that each area of the system operate in a way that is consistent with the rest of the system because even the most advanced users are rarely familiar with all aspects of the system. Users must be able to rely on their experience with other parts of the system to help guide them through the unfamiliar areas. For this reason, consistency was an important guideline throughout the development of HP VEE.

High performance for interactive operations is critical because users will become frustrated using a product that is too slow. Very few users will be happy if they must wait an inordinate amount of time before a particular operation is complete. The allowable time for the system to complete a task depends on the nature of the task and what the user is likely to be doing at the time. For example, a key press should be echoed back to the user within about 100 ms or so. If it takes longer, the user may press the key again thinking that the system didn't get the first one. A pop-up dialog box or menu should appear within about 500 ms. Other tasks such as loading a file can take many seconds before the user will become annoyed because of sluggish performance. We created a list of about ten different interactive operations for which we felt that performance was an important goal. On all supported platforms, many of the operations in this list such as the pop-up menus and dialog boxes are completed within the required time. Unfortunately, there are still a few operations that are completed within the specified time limits only on the very fast HP 9000 Series 700 workstations. On the other hand, we have received very few complaints about interactive performance, so our time limits may have been overly stringent.

In some situations, such as saving a file to the disk, performance simply cannot be guaranteed. In these cases, continuous feedback indicating progress being made is important. Otherwise, it isn't easy to tell whether something is happening or not. In HP VEE, all user-invoked operations that could take more than about 200 ms will result in a change to the mouse cursor. Some of these cursors represent specific activities such as reading from or writing to the disk. For other situations, a general hourglass cursor is used. Any action that is expected to take longer than one or two seconds is also accompanied by a pop-up message box that indicates that the operation is in progress.

Reducing the total number of mouse clicks, menu choices, and various other adjustments required of the user was a major challenge. Each adjustment required of the user, no matter how easy, will reduce the user's overall effectiveness. For this reason, HP VEE is designed to do as much as possible with default settings while allowing adjustments if more control is desired. Other systems often require that the user fill out a form each time a new object is selected from the menu. In most cases, HP VEE will insert default values for all settings and then allow the user to change them later if it becomes necessary.

System messages for errors and other reasons are an especially important source of difficulty or frustration for users. Most software developers seem to take the attitude of a hostile enemy when presenting the user with an error message. However, errors are seldom true user mistakes, but instead are usually triggered by failings in the system either because it misled the user or because it did not adequately protect the user from making the mistake in the first place. In many cases in HP VEE, we were able to avoid generating errors simply by restricting available choices to those that would not result in an error. For example, if a certain combination of selections will cause an error, we show them as mutually exclusive choices. In cases where such restrictions could not be used to avoid the potential for an error, we worked to simplify the interface so that users would be less likely to make mistakes in the first place. In cases where errors were unavoidable, we kept the attitude that error messages should help the user complete a task. We tried to remember that the user generally has a valid reason for performing the operation that resulted in an error.

Two kinds of messages that are common in many systems are not present in HP VEE. The first is the message "Please wait." It is irritating to users because they don't want to wait and they are being instructed to do so. The message is also unnecessary since more descriptive messages can be used instead. Messages such as "Reading from file program2" are much more informative and are not nearly so annoying. The other common message is a confirmation box that asks "Are you sure?" This is also very annoying because the user seldom initiates any operation without being pretty sure about wanting to perform that operation. There are really two reasons for asking "Are you sure?" One is to caution the user about data loss and the other is to protect against accidental requests.

In HP VEE, we solve the first situation by asking a question such as "Save changes before clearing workspace?" This has the same result as "Are you sure?", but does not sound as if the user's choice (or sanity) is being questioned.

In the second situation, accidental requests are better solved by making the input mechanisms easier to operate without error or by making corrections easy to perform. For example,

instead of asking "Are you sure?" to find out if the user really wants to delete an object, HP VEE puts the deleted object into the cut buffer so that if the user decides that a mistake was made, the paste operation can be used to restore the deleted object.

Attention to detail is very important to a user. Most systems available today lack the small details that make a system more convenient and easier to use. In HP VEE, we have attempted to pay attention to as many of these small details as possible. For example, when connecting a line to a box, an outline is displayed around the pin that would be connected if the line were released at that point. Another example of a very small detail is that HP VEE allows objects to be resized as they are being placed on the workspace for the first time. These seemingly minor details help reduce the amount of frustration that users often feel.

Program Visualization Features

In a traditional programming environment, the programmer must spend a large fraction of the development time thinking about details of the programming process including the language syntax, debuggers, and so on. Since HP VEE allows the user to think almost exclusively in terms of the problem domain, most of the development time and effort is spent on solving the problem instead of the programming details. This is the primary source of the productivity gains that many users of HP VEE have experienced. However, even though HP VEE allows programs to be expressed directly in terms of the problem, not all user-written programs will run correctly the first time. Although the so-called accidental complexities[1] of program development such as language syntax and semantics have been reduced or even eliminated, the fundamental complexities of the problem itself still remain. Therefore, once an HP VEE program is developed, it is likely that some aspect of it will not quite work as expected. For this reason, we developed several tools that can be used to visualize the execution of a program to help identify the source of any problems.

Show Execution Flow animates the execution of the program by outlining each object as it begins to execute and then erasing that outline when execution is complete. Besides helping to visualize how the program executes, this is useful when trying to understand performance issues, since an object in the program that consumes a lot of time will be highlighted for more time than other objects. HP VEE also has a timer object, which allows a more objective way to measure performance.

Show Data Flow animates the movement of data as it travels between objects in the program by displaying an icon moving rapidly along each line as data flows across it. This helps the user visualize the relationships between the data and the execution of the objects of a dataflow program. Both Show Execution Flow and Show Data Flow slow the execution of HP VEE programs so much that they are designed to be turned on and off separately.

All data in HP VEE has additional information such as size and shape associated with it. This information is maintained so that one operation can work with a variety of different data types and shapes. For example, math functions such as sin( ) can operate on either an individual number or an array of numbers with any number of elements. This is possible because the size and number of dimensions are packaged with the data. Other information such as the name of the data and mappings (the implied domain of the data) can also be associated with the data. The line probe feature allows the user to examine the data and this associated information at any time.

The execution of a program can be halted when execution reaches a particular object simply by setting that object's breakpoint flag. Breakpoints can be set on any number of objects at any time. When execution reaches an object with its breakpoint flag set, the program will pause and an arrow pointing to that object will appear. At that point the step button can be used to single-step the program one object at a time or the line probe can be used to examine data.

If an error occurs during execution of the program and no error recovery mechanism has been attached, a message will be displayed and an outline will highlight the source of the error visually. This allows the user to locate the source of the error more quickly.

User Interface for HP VEE Programs

Since a user of HP VEE should be able to generate programs with the same advanced user interface as HP VEE itself, several important capabilities have been incorporated into HP VEE to make the task of building impressive-looking programs simple.

For example, data can be entered using a variety of data entry objects. The simplest of these is a text field that accepts a single line of textual data. Numeric fields of various types such as integer, real, complex, or polar complex accept the appropriate numeric data. In addition, these numeric fields can accept constant expressions such as "SQRT(45)" or system-defined constants such as "PI." When typed, these constant expressions are immediately evaluated and the result is converted to the expected type by the input field. Since all editable fields in HP VEE share the same editing code internally, any numeric field in the system that requires a numeric entry can also accept a constant expression.

There are other more advanced mechanisms for entering data or specifying selections to an HP VEE program. Integer or real numeric input can be generated within a predefined range by using the mouse to drag the handle of a slider object. Selections from a list of acceptable values can be made using an enumerated list box, which can be displayed as radio buttons, as a single button that cycles through the list of values, or as a button that accesses a pop-up list box of choices. An HP VEE program can be designed to pause until the user is ready to continue by using the Confirm button.

Data can be displayed in a variety of ways. In addition to textual displays, real or integer numbers can be displayed on a meter object, which can show visually where a number falls within a range. Graphical displays such as XY graphs and polar plots show two-dimensional plots of data and can be interactively scrolled or zoomed. Stripcharts graph a continuous scrolling history of the input data.

All of these input and output types would have limited value if they could only be displayed when the rest of the HP VEE program with all of its lines and boxes is also visible. For this reason, HP VEE is designed with a feature called user panels, which allows an advanced user interface to be attached to a user-written HP VEE program. The user panel is built using an approach similar to many of the available user interface builders. Elements to be placed on the user panel are selected from the HP VEE program and added to the panel. The user can then move and resize these elements as appropriate for the design of the panel. Other layout options such as whether a title bar appears can also be adjusted. Since the elements on the user panel are selected from the user's program, no external code is required and the finished program is easier to build than with most user interface builder tools.

Another important aspect of an advanced human interface is the ability to hide data until the user has asked to examine it. HP VEE is designed with a feature called Show On Execute which allows HP VEE programs to use pop-up windows to hide data until a user request is received. This works by associating a user panel with a user object (similar to a subroutine in traditional programming languages) and enabling the Show On Execute feature. When the user object begins executing, the associated user panel is automatically displayed. When execution of the object is complete, the user panel is erased. Messages such as "Writing test results to file" can be displayed using this mechanism simply by putting the appropriate message on the associated user panel. If it is desirable to pause the program until the user has finished examining the displayed panel, a confirm object can be used.

Programs developed in HP VEE are highly malleable; they can be changed and adjusted as much as desired. However, in many situations it is desirable to protect the program from being changed. The secure feature in HP VEE accomplishes this by displaying only the user panel and making it impossible to alter the program or even look at it after the program has been secured.

Fig. 3. Simplified class hierarchy of HP VEE.

Using all of these features allows users to generate complete application programs with professional appearances without having to work outside of the simple dataflow environment.

Internal Architecture

Fig. 3 shows a simplified class hierarchy for HP VEE showing some of the key classes in the system and how they relate to each other in the inheritance hierarchy. The Object class is at the root of this hierarchy and implements the fundamental protocol for all objects in the system. This includes creating, freeing, and copying objects. The key subclasses of Object include View, which maintains a rectangle on the display, Container, which holds a unit of data, and Device, which represents the inner workings of an element in an HP VEE block diagram.

The fundamental visible element in HP VEE is implemented with the class called View. It maintains a single rectangular region on the display and can be transparent or a composite of other views. The View3d class adds a solid background color and a 3D border to View.

Views are organized into a hierarchy tree based on the display stacking order. The root of this tree is called DispDriver. It is always mapped to overlay the system window allocated to HP VEE. It performs all low-level screen display operations such as drawing lines and filling regions. It also handles the window system interface functions such as repaint requests and dispatching of input events. Fig. 4 shows a composite of views in a view hierarchy with some of the views labeled with the name of their associated class. Fig. 5 shows the complete hierarchy tree of the views in Fig. 4.

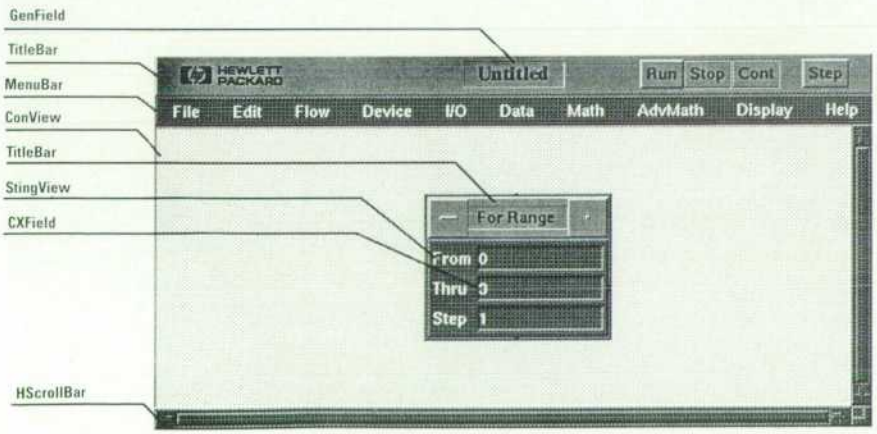


Fig. 4. A composite view with some of the component views labeled.

Fig. 5. Display hierarchy tree.

Subviews are views that are attached to another view called the superview in the display hierarchy tree. Subviews are clipped at the edges of their superview. In this way, each level of the view hierarchy tree limits the visual boundaries of all views below it. This view hierarchy indirectly describes the view stacking order, which ultimately controls which views appear to be on top and which ones are hidden.

Each view maintains a description of the region on which it is allowed to display itself. This clip region is calculated by taking its own bounds, subtracting any region that falls outside the bounds of any view in its superview hierarchy, and then subtracting any views that partially or completely cover it or any view in its superview hierarchy.

Repainting

When repainting an area that it is maintaining, a view may either use the clip region to limit the areas it actually changes on the display, or it may paint any area that it owns and then paint every view that appears closer to the user in the view stack. The full view stack repaint method has nothing to calculate or check before it begins painting itself completely and then painting anything that might be on top of it. If nothing is on top of it, then the complete stack repaint is very efficient because it is so simple. However, if there are many other views covering the view to be repainted, the full stack repaint will be very slow because of all of the unnecessary repainting that must be done. The clip region repaint method is much more efficient in this situation since only those areas that are directly visible to the user will be repainted. This means that no unnecessary repainting must be done.

Unfortunately, the clip region is at best an approximation since views are allowed to display images of arbitrary complexity (such as text) and be transparent in other areas. This gives the user the illusion that views can have any shape without incurring the performance penalties associated with nonrectangular views. It would be very costly to calculate these regions accurately, so only the approximation based on the rectangular view bounds is actually calculated. This means that repaints using the clip region method will not correctly update regions behind transparent areas of other views. Therefore, the clip region repaint method is used in only a few special cases.

Input events such as mouse clicks are dispatched to individual views in the system using a two-phase search mechanism. In the first phase, working from back to front, each view in the view stack where the event occurred asks the views in front of it to process the event. When there are no more views in front of the current view, the second phase begins with an attempt to consume the event. Working from front to back, each view in the view stack (as determined during the first phase) is given an opportunity to consume or ignore the event. If the view takes no special action, the event is passed to the next view down in the view stack. If the the view intends to consume the event, it does so by performing an action associated with the event such as indicating that a button has been pressed and then marking the event as consumed. This process continues until the event is consumed, or until the DispDriver class is given the event (this class consumes all events).

Other Classes

The visible part of each object in an HP VEE program is maintained by the DevCarrier class. DevCarrier's job is to maintain the visual appearance of all objects in the system by showing the terminal pins, maintaining the various highlights and outlines required by HP VEE, and allowing various editing operations on the user's program such as connecting lines and adjusting the sizes or positions of objects. DevCarrier does not display any object-specific information. That information is displayed by object-specific view classes, which are attached to DevCarrier as subviews.

User objects are specialized objects that are built using a subclass of DevCarrier called SubProg. SubProg maintains a visual subprogram which acts just like a normal object when viewed from the outside, but contains an internal dataflow network of its own that is just like the main program. All of the dataflow networks in HP VEE are maintained by a class called ConView (context view). It is the gray background area behind the lines and boxes in a dataflow network.

The top-level workspace environment class—IPEditor (iconic program editor)—is just a special case of SubProg and is therefore built as a subclass of SubProg. It is attached as the only subview of DispDriver and maintains the top-level editing environment. It is the same as SubProg, except that it must maintain the menu bar, run/stop buttons, and other features specific to the top level.

The context view class (ConView) maintains a list of all objects in the context and the lines connecting them. When the context view is asked to repaint itself, it first paints its background color (gray, by default), and then paints all lines in the line list. Then each HP VEE object in the context is painted according to the stacking order. If an HP VEE object falls partially or completely outside the context view's bounds, then according to the clipping rules, that view will be only partially painted or not painted at all.

The clipping and repaint algorithms allow an HP VEE program to be visually much larger than the screen space allotted to it. By adding navigation controls such as the background scroll capability, a very large dataflow network can be supported even on a comparatively small screen.

Model-View Architecture

HP VEE is organized around a model-view architecture. This is similar to the model-view-controller architecture used in other object-oriented systems except that we chose to merge the functionality of the controller into the view. The fundamental assumption in the model-view architecture is that the internal data and program elements (the models) can operate without any knowledge of or dependence on their visual representations (the views). By separating the system at this natural boundary, both the views and the models can be written more simply without any unnecessary dependencies. One feature of this architecture is that one model can be attached to any of several different views without any special support in the model. For example, a model that contains a real number can be attached to a text field or to a meter. Since the properties of the number do not change based on how it is displayed, no changes are required of the class that holds the number. However, since there are few similarities between a meter view and a text view, they need not be built with one view class.

User panels are one area that really benefit from the split between models and views. When the user selects an HP VEE object such as a slider and asks that it be added to the user panel, several things happen internally to make that happen. First, if a user panel has not been created for this program or user object, one is created. The user panel class is similar in concept to the context view class, but without support for interconnections required for dataflow networks. Next, an instance of the PanelCarrier class is created to hold a copy of the object-specific part of the slider view. This copy is created from the original and attached to the new panel carrier and to the original slider model (which is not copied). The panel carrier is then attached to the user panel view.

One of the most significant architectural impacts of the implementation of user panels is the fact that there can be many independent views attached to the same underlying model at the same time. Because of this architecture, it is easy for panels from user objects to be added as a unit to higher-level panels. This allows the creation of complex panels consisting of grouped controls and displays.

The DispDriver class is designed to be the only place where calls to the underlying window system (such as the X Window System) occur. This allows the display driver to be replaced if appropriate when porting to a new platform. During development, for example, we used a driver written to talk directly to the display card of an HP 9000 Series 300 computer because it ran so much faster than the window systems. Now that very high-performance workstations are available, this is no longer necessary.

Printing is handled simply by replacing DispDriver with the printer driver class, which knows how to perform graphics operations on a printer. The information intended for the printer is just "displayed" on the printer and no special printer support must be developed aside from the printer driver itself. This also allows the print output to match the screen display very nicely.

Acknowledgments

Building an advanced user interface is really not difficult, but it takes a great deal of thought and perseverance. It also requires support from management. We were lucky on the HP VEE team because we had managers who understood the value of a good user interface. They encouraged the team to produce the best product that we were capable of even if the schedule would be put at risk. Of course, the team members themselves were very highly motivated to produce an exciting product. John Bidwell, the HP VEE project manager, provided the leadership and management support required for our success. He was able to resist the temptation to ship the product before it was ready, and kept all of the various team members focused on the goal of a truly easy-to-use product. Sue Wolber, Randy Bailey, and Ken Colasuanno each contributed to the overall usability of the system in each of their respective areas. Jon Pennington performed usability testing and provided most of the usability feedback during development.

Reference

1. P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*, September 1987, pp. 43-57.

HP VEE: A Dataflow Architecture

HP VEE is an object-oriented implementation. Its architecture strictly separates views from the underlying models. There are two types of models: data models and device models. Special devices allow users to construct composite devices.

by Douglas C. Beethe

The HP VEE dataflow programming environment was developed with the specific objective of providing an interface that would allow users to express a problem in block diagram form on the screen and then execute it directly. Dataflow programming was chosen because of its direct correlation to the block diagram models we wished to emulate.

Previous efforts in industry and academia related to dataflow programming had yielded some interesting results, but general applicability had not yet been established. Thus our early research efforts were directed primarily at the question of whether we could solve some of the problems that had plagued earlier attempts and prove general applicability.

The design and construction of HP VEE used object-oriented technology from the beginning. We had enough experience with procedural coding technology to realize that a project like HP VEE would be too complex to achieve with procedural technology. The architecture that evolved from this development is the subject of this article. The design of various elements of the underlying HP VEE architecture will be discussed as will the manner in which they work together to produce the executable block diagram as a dataflow model.

The Model-View Paradigm

One of the characteristics of the HP VEE architecture that is common to most object-oriented implementations is the strict separation between models and views. Most users are familiar with the world of views, and indeed often make no distinction between the view of an object and its underlying model.

From a functional point of view the model is the essence of an object, incorporating both the data (state variables) that gives the object its uniqueness, and the algorithms that operate on that data. In HP VEE, by definition, the model operates independently of the view, and does not even know of the existence of any graphical renderings of itself, except as anonymous dependents that are alerted when the state of the model changes (see Fig. 1).

There are many examples of applications that have views possessing no underlying functional models. Consider the various draw and paint programs, which allow the user to create very sophisticated views that, once created, are incapable of performing any function other than displaying themselves. Likewise, there are numerous examples of applications that support very sophisticated functional models but lack any ability to display a view of those models, be it for passive display of state or for active control.

Most of the scientific visualization software appearing today is used to create views of the data output of functional models that have little or no display capability. Most of the views that are seen by the HP VEE user are actually graphical renderings of the states of underlying models. In the interactive mode, access to the models is by means of these views, which communicate with their respective models to change their state, initiate execution, and so forth. For example, the view of the ForCount iterator has a field in which the user can enter the number of times the iterator should fire at run time. Upon entry, this value is sent to the underlying device model, where it is kept as a state variable. When this state variable is changed, the model sends out a signal to anyone registered as a dependent (e.g., the view) that its state has changed, and the view then queries the model to determine the appropriate state information and display it accordingly (see Fig. 2).

This strict separation between model and view might seem excessive at first, but it results in a partitioning that makes the task of generating the two different kinds of code (very different kinds of code!) much easier from the standpoint of initial development, portability, and long-term code maintenance. It also eases the job of dealing with noninteractive operations in which HP VEE is running without any views at all, either by itself or as the slave of another application. And finally, this separation eases the task of developing applications that must operate in a distributed environment where the models live in one process while the views are

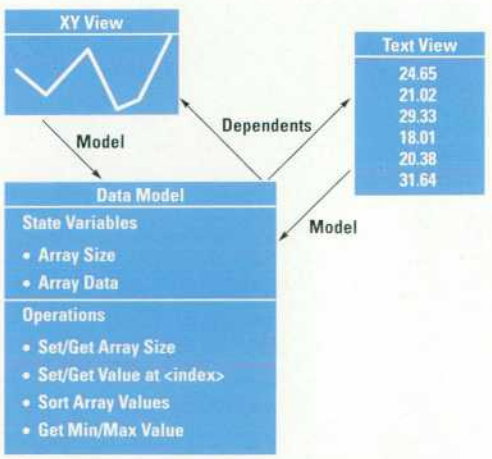


Fig. 1. Two different views of the same underlying model.

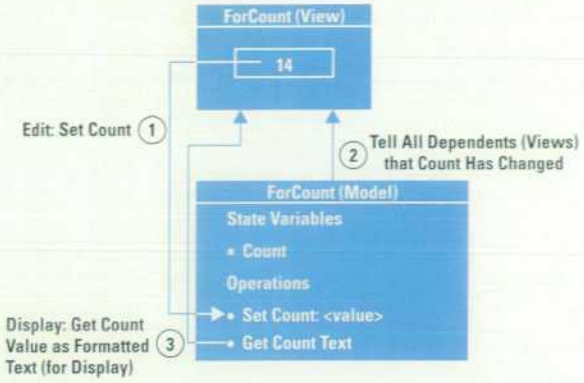


Fig. 2. Interaction of a view and the underlying model at edit time.

displayed by another process, possibly on an entirely different system. This last aspect is becoming more and more important in an application world that is taking increasing advantage of networked systems.

HP VEE itself is composed of two kinds of models. The first is the device model, which acts like a black box having inputs, outputs, and some operational characteristic that transforms the data at the inputs to the result at the outputs. The second is the data model (container), which handles the transport of information along the data lines, which interconnect devices. The data model also provides mathematical functions, which can be invoked to operate on the data, and formatting and deformatting functions, which change the representation of the data when required for display or for communication with some other application that requires the data in a different form. Both types of models will be discussed in some detail.

Data Models

The fundamental abstraction for information in HP VEE is the container object (Fig. 3). Containers can hold data for any of the supported data types: text, enumerated, integer, real, complex, polar complex, coordinate, waveform, spectrum, and record. Both scalars (zero dimensions) and arrays from one to ten dimensions are supported. In addition, the dimensions of array containers can be mapped in either linear or logarithmic fashion from a minimum value at the first cell of a dimension to a maximum value at the last cell of that dimension. This allows an array of values to have some physical or logical relationship associated with the data. For example, a one-dimensional array of eleven measurements can be mapped from 0 to 32 cm to indicate the physical relationship of the values in each position in the array to some real-world phenomenon. The first value would be at 0 cm, the next at 3.2 cm, the next at 6.4 cm, and so on.

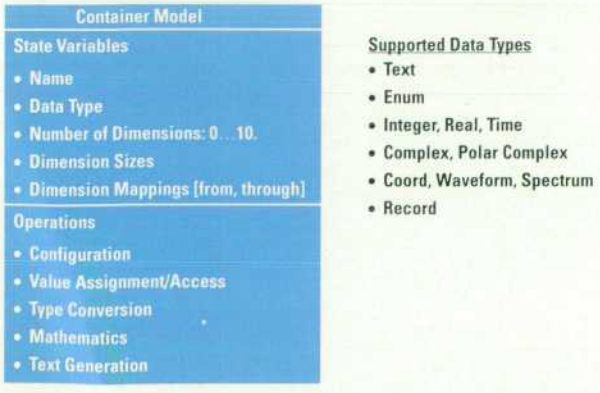


Fig. 3. Container model attributes.

One of the properties of containers that is used extensively in HP VEE is the knowledge of how to transform to another type on demand. The automatic form of this transform is allowed only for transforms that incur no loss of information. This has the effect of allowing most promotions, but disallows any automatic demotions. For example, integer can be promoted to real, and real to complex or polar complex, but complex cannot be demoted automatically to real. To do so would likely cause the loss of information that would not reappear in the promotion of that real value back to complex. An interesting special case of this is the reversible transformation between waveform and spectrum (time and frequency domains). While these data types seem to have the same irreversible relationship to each other as the real and complex types just discussed, it is a well-known fact that a reversible transformation exists between these two special types by means of the Fourier transform. For example, a 256-point waveform is transformed to a 129-point spectrum (ignoring the symmetrical values with negative frequency), and the same spectrum regenerates the original 256-point waveform by means of the inverse Fourier transformation (Fig. 4).

Another powerful property of containers is their inherent knowledge of data structure as it applies to mathematical operations. At first glance, operations such as addition and subtraction seem relatively simple, but only from the standpoint of two scalar operands. For other structural combinations (scalar + array, array + scalar, or array + array) the task requires some form of iteration in typical third-generation languages (3GLs) like C that has always been the responsibility of the user-programmer. Containers encapsulate these well-understood rules so that the user deals with, say, A and B simply as variables independent of structure. When any of the nontrivial combinations is encountered, the containers decide among themselves if there is an appropriate structural match (scalar with any array, or array with conformal array) and execute the appropriate operations to generate the result.

Other more complicated operations with more robust constraints (e.g., matrix multiplication) are handled just as easily since the appropriate structural rules are well-understood and easily encapsulated in the containers. These properties aid the user in two ways. First, the user can express powerful mathematical relationships either in fields that accept

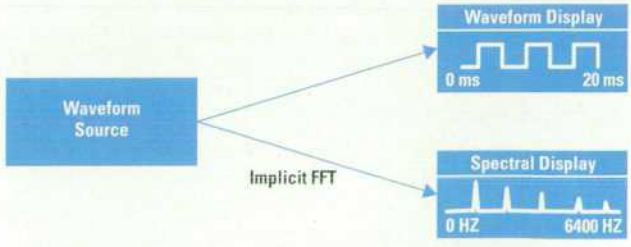


Fig. 4. Automatic transformation of a time-domain waveform (e.g., 256 real values, 0 to 20 ms) to a frequency-domain spectrum (129 complex values, 0 to 6400 Hz).

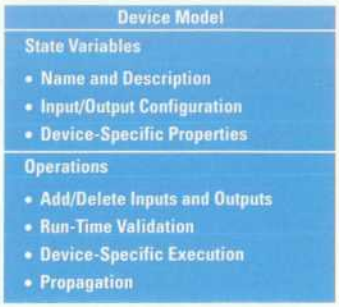


Fig. 5. Attributes of a simple device model.

constant expressions or in any of several delayed-evaluation fields (e.g., Formula, If/Then, ...) without having to deal with the cumbersome iteration syntax of 3GL programming. This by itself has the pleasant side effect of eliminating much if not most of the iteration in many applications, compared to their 3GL equivalents. Second, the interconnection of the various objects that make up a model in HP VEE is much simpler when any of the inputs is constrained to a specific data type. Since the containers know how to respond to most requests for type change, the user is freed from the cumbersome task of explicitly changing (casting) the original type to the required type. For example, the inputs to a spectral display that requires a spectrum input will not disallow connection to a waveform (time-series data) because the output supplying the waveform will transform it to a spectrum on demand at run time. This same capability is used during the evaluation of any mathematical expression, thus allowing the user to intermix types of operands without explicit type casting.

Device Models

Fig. 5 shows the attributes of a simple device model. Each device can have its own inputs and outputs. Many have user-controllable parameters that are accessed as constants through the panel view of the device or as optionally added inputs. In general, the device will execute only when each of the data inputs has been given new data (including nil data). Thus the data inputs to any given device define a system of constraints that control when that device can execute. This turns out to be quite natural for most users since the data relationships that are depicted by the data lines that interconnect devices generally map directly from the block diagram of the system in question, and often are the only form of constraint required for the successful execution of a model.

There are numerous cases, however, where an execution sequence must be specified when no such data dependencies exist. Such cases typically fall into two categories: those where there is some external side effect to consider (communications with the real world outside my process) and those that deal explicitly with real time. To deal with this situation we developed the sequence input and output for each device (on the top and bottom of the device, respectively), as shown in Fig. 6. The sequence output behaves like any other data output by firing after successful execution of the device except that the signal that is propagated to the next device is a always a nil signal. Likewise, the sequence input behaves like any other data input with one exception. When connected it must be updated (any data will do, even nil) along with any other data inputs before the

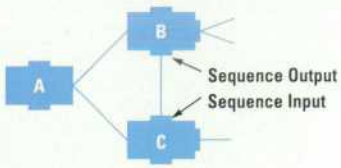


Fig. 6. While B and C both need the data from A, the sequence connection between B and C will cause C to execute after B.

device will be allowed to execute, but unlike other data inputs, connection is not required. Thus any time it is required that A must execute before B where no other data dependencies exist between the two devices, it is sufficient to connect the sequence output of A to the sequence input of B.

For users who have already been introduced to programming in third-generation languages such as Pascal, C, or BASIC this can require a paradigm shift. Experience with such users has shown that they are often preoccupied with sequencing (since 3GLs almost universally use control-flow paradigms) and have a difficult time at first believing that the data constraints represented by the lines that interconnect the devices are sufficient to define a robust sequence of execution. It is only after using the system for a time that they are weaned away from this need to sequence each and every device explicitly and begin to feel comfortable with the dataflow paradigm.

Contexts

Several types of devices are supplied as primitives with HP VEE, including those used for flow control, data entry and display, general data management, mathematical expressions, device, file, and interprocess I/O, virtual signal sources, and others. There is also a mechanism that allows users to construct special devices with their own panels and a specific functional capability. This device is known as a UserObject and is essentially a graphical subprogram.

UserObjects (Fig. 7) encapsulate networks of other devices (including other UserObjects) and have their own input/output pins and custom panel displays. Viewed as a single collective object with its own panel, each UserObject operates under the same rules as any primitive device: all data inputs must be updated before the UserObject will execute its internal subnet. Each UserObject will contain one or more threads, which execute in parallel at run time. In addition, threads in subcontexts (hierarchically nested contexts) may well be

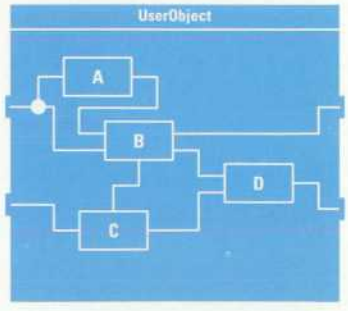


Fig. 7. A UserObject encapsulates a subnetwork of other objects into a single larger object with its own inputs and outputs.

running in parallel with their host threads in their parent contexts.

UserObjects can be secured such that the user of the device can access only the panel and not the internals. In this form the UserObject is almost indistinguishable from any primitive device. This capability allows developers to create arbitrary devices that can be archived in a library for later access by users, who can treat these devices as true primitives in their application.

Threads

Devices that are connected to each other within the same context form a single thread of execution. One of the inherent advantages of dataflow programming is the ability to support multiple independent threads of execution with relative ease (see Fig. 8). This becomes particularly useful when interacting with the rest of the world, since independent monitoring operations ("Has that message arrived yet?") can proceed in parallel with related operations. In typical 3GLs such operations require elaborate schemes for enabling interrupts and related interrupt service routines. Most who have dealt with such code as inline text can attest to the difficulty of maintaining that code because of the difficulty of easily recreating the relationship between parallel operations once the code has been written.

Several devices were developed especially for thread-related activities. One of these is the Exit Thread device, which terminates all execution for devices on that same thread when encountered. Another is the Exit UserObject device, which terminates all execution on all threads within the context in which it is encountered.

Certain devices have the ability to elevate a thread's priority above the base level to guarantee that thread all execution cycles until completion. One such device is the Wait For SRQ device (SRQ = service request), which watches a specified hardware I/O bus in anticipation of a service request. If and when such a request is detected, this device automatically elevates the priority of the subthread attached to its output so that all devices connected to that subthread will execute before devices on any other thread (within this context or any other context) until that subthread completes.

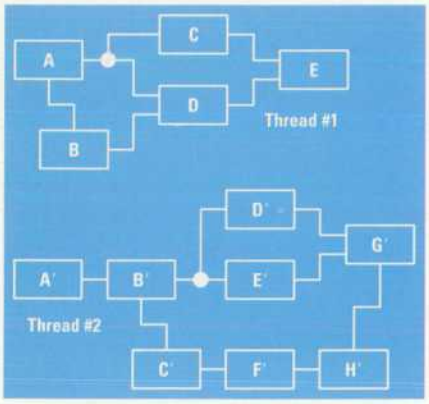


Fig. 8. Any context (e.g., a UserObject) can contain one or more threads, each of which executes independently of all others within that context.

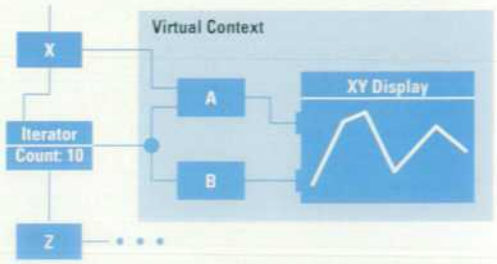


Fig. 9. Objects A and B and the XY display will execute 10 times each at run time as the iterator fires its only data output (right side) 10 times before firing its sequence output (bottom). The data from the output of X is reused for the last 9 of the 10 executions of A (active data rule).

Although it is not specifically thread related, a similar capability exists for exception service. At the time an exception is raised (e.g., an error occurs), all other devices on all other threads are suspended until an exception handler is found (discussed later).

Propagation: Flow of Execution

From an external point of view, the determination of which devices can execute is a simple problem of finding out which devices have had all of their inputs updated. From an internal point of view, the problem is a bit more difficult. To prevent infinite feedback the general rule for dataflow programs is that each device can execute only once per activation of the context in which the device resides. On the other hand, it was felt from our earliest prototypes that having iteration occur within some subgroup of devices in a context was superior to dropping down into a subcontext multiple times to accomplish the same thing, especially for nested iteration.

Thus we were faced with the problem of allowing groups of devices to execute multiple times within a single activation of a context. Identification of these devices could only occur at run time as they appeared on the subthread hosted by the primary output of an iterator. To deal with this we developed the virtual context, which is defined not by the user but by the system (see Fig. 9). At run time, the devices that are executed on the subthread hosted by an iterator are remembered. Then, just before the next firing of the iterator (since an iterator generally fires its output more than once for each execution of that iterator), the devices in this virtual context are selectively deactivated separately from the other devices in the context. This allows them to be re-executed when the iterator fires again by the normal rules of propagation.

One other side effect of such iteration is that any data being supplied to a device within the virtual context by a device that is outside that virtual context is going to be delivered only once to the device within the virtual context. Thus new data is supplied to the inputs as required on the first iteration, but on all subsequent iterations no new data arrives. One could solve this by using a special intermediary Sample&Hold device, but a simple extension to the rules of propagation turned out to be much easier. The extension,

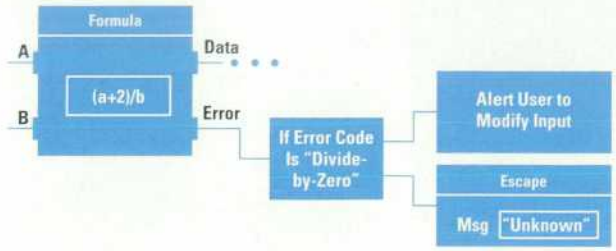


Fig. 10. The special error output will fire in lieu of the data output if any error is encountered while evaluating the formula. The value posted at the error output is the error code number. This allows the user to decide how to handle the situation.

known as the "active data rule," says that data from any active output of a device that is currently active (executed, but not yet deactivated) can be reused. This has essentially the same effect as the Sample&Hold but is much less error-prone.

The goal in all of this is to create a scheme of execution that does not require the user to specify a sequence of execution with explicit device-by-device triggering as is common in the world of digital design. In addition, we wanted execution to proceed as if the entire network were running on a multiprocessor architecture with true parallelism. On a typical uniprocessor machine only one primitive device is actually drawing cycles from the processor at any one instant, but the overall effect is as if all devices both within the same context level and across other levels of the network hierarchy are running in parallel.

Asynchronous Operations

For some devices we found a need to invoke certain operations programmatically that were peripheral to the general operation of the device, such as AutoScale or Clear for an XY graph. While the primary function of the graph is to construct a graph from the data present at the synchronous data inputs, operations such as AutoScale could happen at any time. A different class of inputs that were not incorporated into the general scheme of propagation was needed to initiate these asynchronous operations. Thus we developed the control input, which when updated at run time will perform its assigned function within the associated device regardless of the state of any other input on the device.

Exception Management

Exception (error) management could have been approached from a number of different points of view, but it proved most effective to implement a strategy based on an optional output that fires if and only if an untrapped exception is raised from within the scope of that device (Fig. 10). For primitive devices this allows the user to trap common errors such as division by zero and deal with possibly errant input data accordingly. In each case a number (an error code) is fired from the error pin and can be used by the ensuing devices to determine just which error has occurred. If the decision is not to handle the error locally, the error can be propagated upward with the Escape device, either as the same error that could not be handled locally or as a new user-defined code and message text, which may be more informative to the handler that eventually owns the exception.

Hierarchical exception handling is possible because an error pin can be added to any context object (UserObject) to trap errors that have occurred within its scope and that have not been serviced by any other interior handler. If the exception pops all the way to the root context without being serviced, it generates a dialog box informing the user of the condition and stops execution of the model. To enable the user to locate the exception source, the entire chain of nested devices is highlighted with a red outline from the root context down to the primitive device that last raised the exception.

Acknowledgments

Much of the conceptual framework for HP VEE in the early stages came from lengthy discussions with John Uebbing at HP Labs in Palo Alto. His insights and questions contributed significantly to many elements of the underlying structure which eventually matured into the HP VEE product. John's vision and imagination were invaluable. I would also like to thank several members of the design and test teams whose continued feedback concerning the functional aspects of the product proved equally invaluable: Sue Wolber, Randy Bailey, Ken Colasuonno, Bill Heinzman, John Friemen, and Jerry Schneider. Finally, I would like to thank David Palermo who in his position as lab manager provided the resources and direction to see this project make it from the first conceptual sketches to the real world. No project of this nature can succeed without such a sponsor.

A Performance Monitoring System for Digital Telecommunications Networks

This system collects CCITT G.821 performance statistics on CEPT 2, 8, 34, and 140-Mbit/s data streams and alarm data on network elements. A demux capability permits monitoring of tributary streams within a data stream. Data is collected nonintrusively by peripheral units, which are modular VXIbus systems.

by Giovanni Nieddu, Fernando M. Secco, and Alberto Vallerini

The HP Model E3560 digital performance monitoring and remote test system is designed for surveillance of the quality of a digital telecommunications network and for collecting alarms from network elements, following the guidelines of CCITT Recommendation G.821. The HP E3560 provides the customer with well-defined performance parameters that tell how the network is doing on a statistical basis, and whether a failure has occurred in a network element.

The actual network monitoring is performed by devices called peripheral units, which continuously monitor the telecom links nonintrusively. The peripheral units scan the PCM streams at the four main bit rates in the European (CEPT) hierarchy (2, 8, 34, and 140 Mbits/s), looking for alarms and binary errors, and computing the G.821 performance parameters.

Data produced by the peripheral units is collected by a first-level processor, an HP 9000 Series 400 workstation, which stores the data in a relational database. The first-level processor also provides for configuration of the peripheral devices and presents the retrieved data and alarms to the user.

Digital Network Quality

Digital networks have had and are still having spectacular growth, constantly adding newer and more sophisticated services to customers. In many European countries, it is now possible for a customer to lease 2-Mbit/s digital lines to build a private network. It is very common to lease 64-kbit/s permanent or packet circuits. In the most industrialized countries, practically every large company has its own private network.

Network customers demand and pay for a specified quality of service. The CCITT in its Recommendation G.821 starts with the definition of network quality parameters (see definitions below) and gives end-to-end quality objectives (see Table I) for a 27,500-km, 64-kbit/s circuit called the Hypothetical Reference Connection (HRX). Fig.1 shows the functional representation of the HRX.

The following quality parameters are defined in G.821:

- Errored second (ES): a second with at least one error
- Severely errored second (SES): a second with a bit error rate (BER) worse than 10^{-3}

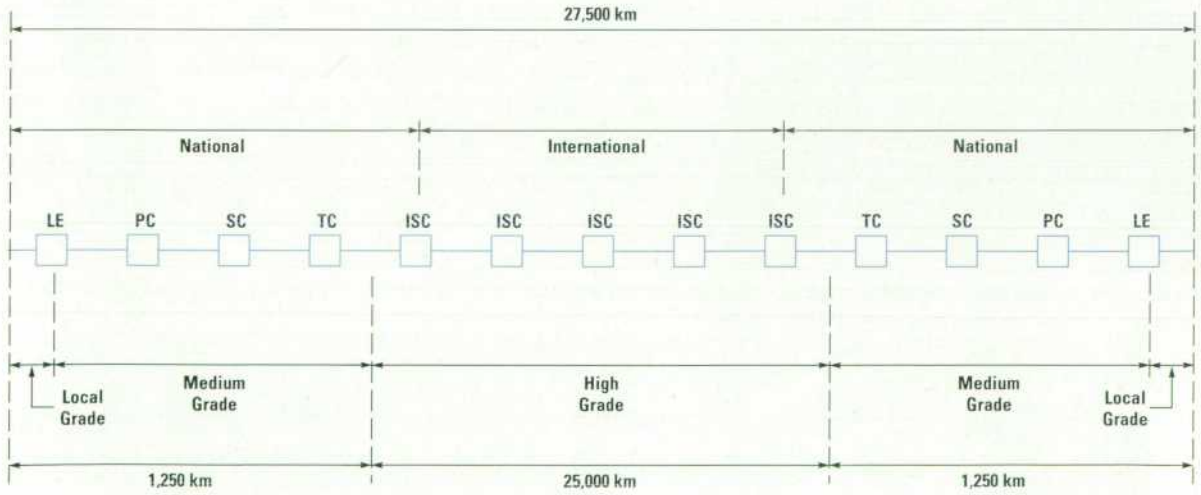


Fig. 1. Functional representation of the Hypothetical Reference Connection (HRX) defined in CCITT Recommendation G.821. LE = local exchange. PC = primary center. SC = secondary center. TC = tertiary center. ISC = international switching center.

- Degraded minute (DM): a collection of 60 non-SES with a BER worse than 10^{-6}
- Unavailable seconds (UAS): a period that starts when at least 10 consecutive SES are counted and ends when 10 consecutive non-SES are seen. UAS includes the first 10 SES and excludes the last 10 non-SES.

CCITT Recommendation M.550 (M-series recommendations are addressed to service management) tells service providers how the objectives of Table I are to be allocated inside the transmission network. The end-to-end objectives of Table I are partitioned according to the quality classification of the circuit (high, medium, or local grade). Table II gives the percentage of the objectives that must be allocated for each circuit classification. For the local-grade and medium-grade circuit classifications, the allocated percentage of the objectives is independent of the circuit length, while for high-grade circuits, the allocated percentage (40% in Table II) must be scaled according to the length of the circuit. For example, for a high-grade circuit 2,500 km long, the allocated percentage of the objectives will be $40 \times (2{,}500/25{,}000) = 4\%$. Annex D of G.821 suggests a method for computing all parameters originally defined at 64 kbits/s for higher bit rates, by measuring errors at the higher rates.

Table I
End-to-End Quality Objectives

Quality Parameters	Objective (Maximum Percentage of Time)
Degraded Minutes (DM)	10
Severely Errored Seconds (SES)	0.2
Errored Seconds (ES)	8

Table II
Allocation of Objectives

HRX Circuit Quality Classification	Percentage of Objective
Local (each end)	15
Medium (each end)	15
High	40

As an example of the allocation of objectives, suppose that the path whose quality parameters are to be measured starts at a local exchange, ends at a secondary center, and passes through a high-grade circuit 1,500 km long. This means that the sum of a local-grade circuit, two medium-grade circuits, and a high-grade circuit must be allocated. According to Table II, the allocated percentage is $15 + 2 \times 15 + 40 \times (1{,}500/25{,}000) = 47.4\%$. This leads to the following path objectives (RPO stands for reference performance objective):

$$\text{RPO(DM)} = 10\% \times 47.4\% = 4.740\%$$
$$\text{RPO(ES)} = 8\% \times 47.4\% = 3.792\%$$
$$\text{RPO(SES)} = 0.2\% \times 47.4\% = 0.095\%.$$

TMN Architecture

The architecture of the HP E3560 follows as closely as possible the architecture proposed in Recommendation M.30 of the CCITT Blue Book Series. M.30, which is better known as TMN (Telecommunications Management Network), establishes the building blocks and data links that should be employed in the design of a network whose aim is the management of the telecomm network.* In Recommendation M.30, four blocks are identified (see Fig. 2):

- Network elements (NE) represent the devices that make up the telecom network. It is assumed that an NE is "intelligent" enough to have the possibility of generating and transmitting some kind of information useful for network management. All NEs produce for external use some sort of internal alarms, both urgent and nonurgent. These are representative of internal faults. Urgent alarms indicate a need for immediate maintenance. Alarms can be displayed in a centralized operation and maintenance center to help network personnel understand where faults have occurred and to minimize the need for manned offices.
- Operations systems (OS) are the blocks where the network management takes place. They can be thought of as computers that receive a large amount of data from the network and provide for its elaboration and for the generation of data useful for management purposes.
- Mediation devices (MD) provide the links between the NEs and the OSs. Their main functions are protocol conversion, information conversion and storage, data buffering, and filtering. These blocks can be absent if the NEs are powerful enough to manage the data link with the OSs.

* Recommendation M.30 has recently been renamed M.3010.

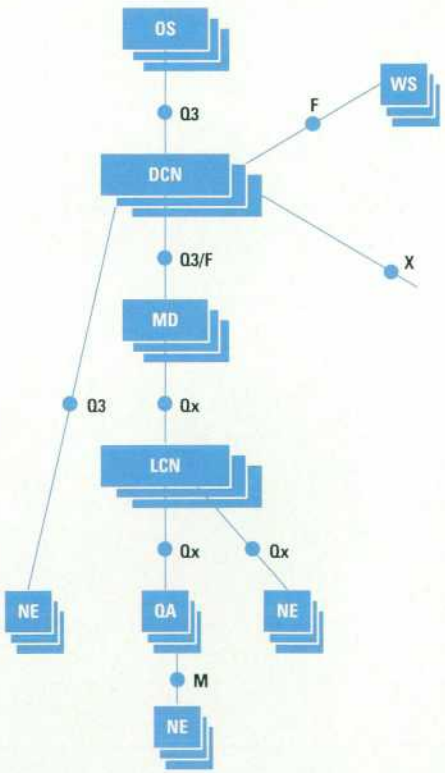


Fig. 2. Simplified physical architecture of the Telecommunications Management Network (TMN) specified in CCITT Recommendation M.30 (now M.3010). NE = network element. OS = operations systems. MD = mediation devices. WS = workstations. DCN = digital communications network. LCN = local communications network. QA = Q adapter, a protocol converter. Qx and Q3 are types of data link protocol stacks. F, X, and M are different types of interfaces.

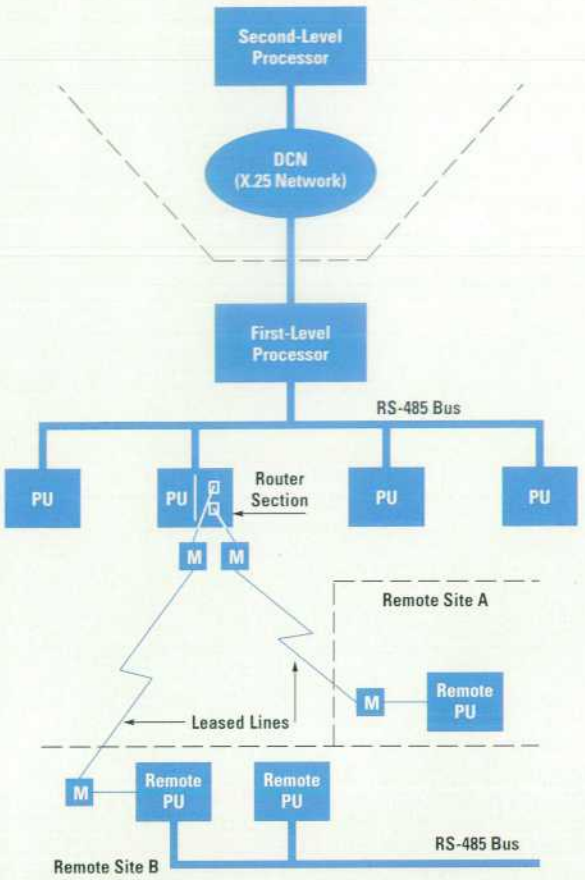


Fig. 3. HP E3560 digital performance monitoring system architecture. PU = peripheral unit. M = modem. DCN = digital communications network. The HP E3560 architecture is modeled on the TMN architecture shown in Fig. 2. The first-level processor is an HP 9000 Series 400 workstation.

- Workstations (WS) display data elaborated by the OSs in a form understandable by humans.

These blocks and the functions they perform must not necessarily be thought of as separate entities. An NE can have functions typical of an MD, and an MD can have functions typical of an OS or a WS.

HP E3560 Architecture

In the HP E3560 architecture, which is shown in Fig. 3, the peripheral units play the role of NEs. They are not actually part of the telecomm network, since they don't provide for the transfer of voice or data, but nonetheless they are a source of management information.

The first-level processor acts as a mediation device (MD) with OS functions. It collects data from the peripheral units and stores it, and it provides basic processing aimed at the generation of performance alarms following CCITT Recommendation M.550. Finally, it is an entry point into the system through the window-based human interface.

The links between the various blocks of the architecture are particular OSI stacks named Qx and Q3. Qx is the link used to transfer data between the peripheral units and the first-level processor. Q3 is a complete seven-layer ISO (International Standards Organization) OSI (Open Systems Interconnection) stack, based on X.25 or Ethernet protocol in its three lower layers. Q3 is defined by CCITT Recommendations Q.961 and Q.962.

Qx, defined in Recommendation G.773, is also called the "short stack," since not all of the OSI layers are present. Two profiles or stack configurations, called A1 and A2, have been proposed by the CCITT. Both are missing OSI layers 4, 5, and 6, which are replaced by some mapping functions that act as a sort of "short circuit" between layer 7 service requests and layer 3 services. Both stacks have the same layer 7 components definition: CMISE (ISO 9595/9596, CCITT X.710/711), ROSE (CCITT X.219-X.229), and ACSE (CCITT X.217-X.227). The mapping functions provide some basic layer 6 services, such as the encoding and decoding functions, according to the basic encoding rules of CCITT X.209.

The two profiles differ in layers 1, 2, and 3, as shown in Fig. 4. Profile A1, which is used in the HP E3560, uses RS-485 as the physical layer, HDLC-NRM as layer 2, and ISO 8473 as layer 3 (of the possible three subsets of ISO 8473, the HP E3560 implements the so-called "NULL IP"). Profile A2 is based on an Ethernet link.

In the HP E3560, Qx/A1 constitutes the data link between the first-level processor and the peripheral units, or in TMN terminology, Qx/A1 is the LCN (local communication network). Since the network topology is point-to-multipoint, the relationship between the first-level processor and the peripheral units is of a master-slave type. The peripheral units are continuously polled by the first-level processor, which acts as a primary station. Only when a peripheral unit receives the polling request (or the RR frame in HDLC terminology) is it allowed to send one packet of data to the first-level processor. Packet length is limited to 256 bytes (one octet in the HDLC frame) and packet segmentation is not allowed.

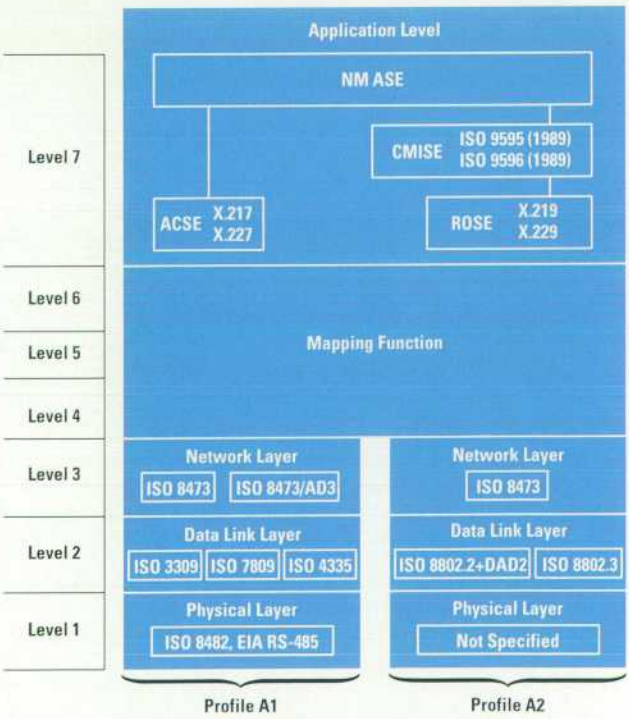


Fig. 4. The Qx data link protocol stack as defined in CCITT Recommendation G.773. Two profiles, A1 and A2, are permitted. Profile A1, which is used in the HP E3560, uses RS-485 as the physical layer.

The number of peripheral units that can be handled by a first-level processor is limited by the addressing capability of the HDLC protocol, which, as stated in G.773, is 254. One byte is used to address a secondary station, and address values 00h and FFh are reserved for the "no station" and "all stations (broadcast)" addresses. On the other hand, the number of peripheral units that can be physically connected to a first-level processor is at most 30 because of charge limitations of the RS-485 bus.

Remote Links

In developing the HP E3560, another protocol limitation had to be overcome: as implied by its name, the LCN cannot span more than a few hundred meters, so the LCN is useless for connecting remote peripherals to the first-level processor. For cost reasons, it is unacceptable to place a first-level processor at each site where a peripheral unit is located, so a solution involving modems and leased lines had to be found.

The HP E3560 solution for remote links takes the form of two additional peripheral unit boards: the communication board and the communication controller board. The communication controller can drive up to eight communication boards, each of which has two RS-232 ports capable of driving an external modem. One of the communication boards is connected to the RS-485 bus. Installed in a local peripheral unit, a set of these boards acts as a router between the main bus and the remote peripheral units, as shown in Fig. 3. One router section can drive up to 16 remote peripheral units or up to 16 remote addresses. The distribution of the addresses among the RS-232 ports is customer-defined, ranging from 16 addresses driven by a single port (using only one communication board) to 16 addresses, each driven by a single port (using eight communication boards).

At the remote site, the peripheral unit physically connected to the modem (called the remote master peripheral unit) acts as a repeater, locally regenerating the RS-485 bus. Throughout the remote links, the protocol from layer 2 up is still Qx, so what we have obtained is the extension of the protocol at the expense of redefining the physical layer for only part of the transmission path.

First-Level Processor Interface

On the first-level processor side of the RS-485 bus, an HP RTI (real-time interface) card interfaces the processor to the peripheral units. This board interfaces to the I/O bus of the HP 9000 Series 400 workstation and accepts inputs from SBX boards purchased from HP or other vendors.† No output interface is provided. From a protocol point of view, the first-level processor's Qx stack is split into two parts: layers 1 and 2 are implemented in the RTI card, while the remaining part runs in the HP-UX* environment. In this way, the stack section most affected by typical real-time problems runs on a dedicated processor with a real-time multitasking kernel (the HP RTI card uses the pSOS+™ operating system). Communication between the two parts is handled by the HP-UX device driver mechanism.

† SBX (Standard Bus eXtension) is an industry-standard bus. The SBX boards used in the HP E3560 fit into the SBX connector on the HP RTI card and have serial ports for RS-485 communication. The HP 94185A 2-channel serial SBX card is used in the HP E3560.

First-Level Processor

The first-level processor's main task is the collection of data produced by the monitoring activities of the peripheral units. This data, divided into the two classes of performance data and alarm data, is processed and stored in a relational SQL database for further analysis and historical tracing. Alarms are displayed on the screen to alert maintenance personnel. To help the operator in problem solving, other software is provided for reporting and fault localization exploiting the demux capabilities available in the peripheral units. The first-level processor can have a Q3 connection to a second-level processor or an existing OS.

The other important first-level processor function is peripheral unit management. Through a simple-to-use human interface based on X Windows, it is possible to set up the boards in the peripheral units and selectively start and stop the monitoring operations.

In addition to the normal software environment provided by HP 9000 Series 400 workstations (HP-UX and X Windows), the first-level processor's software is based on the HP OpenView platform.[1] The services offered by HP OpenView are exploited both from the programmer's side (easy and well-defined communication between tasks, object-oriented approach, etc.) and from the user's side (object management through the use of maps).

Peripheral Units

The HP E3560 peripheral unit can be considered a network element (NE) whose main purpose is to collect status and network quality parameters from other NEs. Alarms are collected directly and indirectly from the NEs and sent to the first-level processor to be processed. Quality parameters are collected indirectly from the NEs, processed according to CCITT G.821, and sent to the first-level processor.

The peripheral unit is designed to be inserted both functionally and structurally into the telecomm environment, specifically in the digital transmission area. The digital transmission area is the part of a telecomm system that deals with digital information transport by means of equipment such as multiplexers, line terminals, regenerators, add/drop multiplexers, cross connections, digital radio relays, and so on. This area and the digital switching area are the building blocks of a digital network. It is reasonable to say that most of today's telecomm equipment is digital and much of it uses fiber optic media to transport signals all over the world.

Peripheral Unit Description

The peripheral unit is built to solve the problem of alarm collection and analysis for a large variety of alarm types. Different physical interfaces are available, including current loop, voltage sensing, and open or closed contact sensing.

Data streams from 2 Mbits/s up to 140 Mbits/s can be analyzed both intrusively and nonintrusively. This is achieved by means of high-impedance probes connected to the data streams at protected monitoring points according to CCITT G.772, or by taking the signal directly from standard monitoring points that are sometimes already present in the network central office. Typical network alarms collected include loss of signal, AIS (alarm indication signal), loss of frame alignment, and so on. It is also possible to count events coming, for example, from radio relay equipment that flags

block parity errors or forward error correction (FEC) code interventions. These events are also processed and used to compute the G.821 parameters.

The peripheral unit is modular for flexible configuration. Many different types of boards can be mixed in the peripheral unit cardcage according to the needs of the application. The boards and backplane conform to the VXIbus standard,[2] and each board is a complete instrument. Thus, expanding existing measurement capabilities is simply a matter of duplicating existing boards, while adding new functionalities is a matter of installing new boards and updating the software. Up to ten application boards and up to three communication controller boards can be installed in each peripheral unit. The peripheral unit can easily be installed in a central office thanks to its standard 19-inch width and relatively small depth (11.8 in). There are two power supply types, one for dc-powered central offices (–48V, –60V) and one for ac-powered offices (100Vac, 240Vac).

The peripheral units are interconnected by an RS-485 bus, which is the physical layer of the TMN Qx protocol. Up to 30 peripheral units can be physically inserted in the same RS-485 bus. This bus is shared by the first-level processor, which is the primary station and polls the peripheral units (secondary stations). It is possible, using the dedicated communication boards described earlier, to accommodate more than 30 peripheral units. The same boards can also be used to connect remote peripheral units through modems on leased lines or service channels.

Fig. 5 gives an overview of the cardcage. The cardcage can house up to 15 B-size VXIbus or VMEbus boards. Thirteen slots have all of the VXIbus lines and form a VXIbus subsystem, while the last two slots have only the VMEbus lines. According to the VXIbus standard, the first slot is for the slot 0 board, which together with the processor board is the resource manager of the cardcage. Slot 0 has four peripheral unit communication ports called J1, J2, J3, and J4. J1 is RS-485, J3 is either RS-232 or RS-485, J4 is RS-232, and J2 is a passive connector that can be paralleled with J1 or J3 and acts as a tee connector for the communication bus. A local maintenance terminal can be connected to J4 for local management and maintenance.

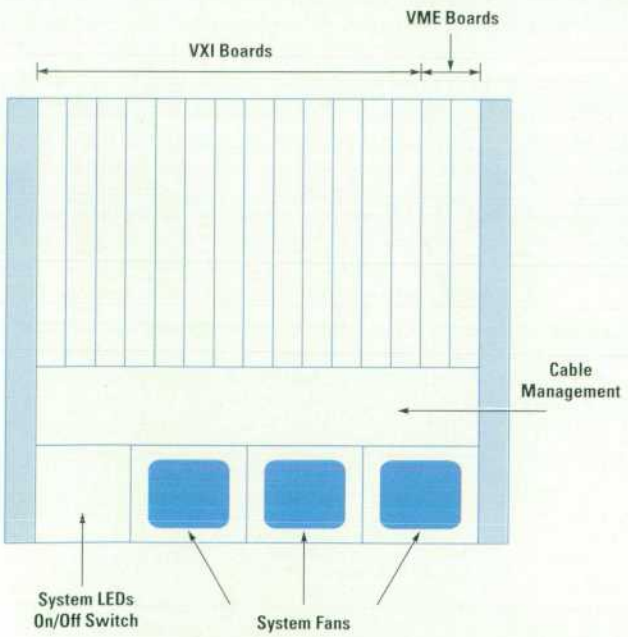


Fig. 5. HP E3560 peripheral unit cardcage organization.

A requirement for the cardcage is that not only the boards, but also the power supply and the fans, must be easily replaced. This is considered important for a telecomm NE because it is common for all of the parts of a telecomm system to be easy replaceable. Another important feature of the cardcage is cable management. In some configurations more than 80 coaxial cables must be managed in the cardcage.

The power supply provides the following resources: +5V at 20A, +12V at 3A, –12V at 3A, –5.2V at 10A, and –2V at 6A. It is also responsible for generating the VXIbus reset and powerfail signals, and it can maintain its specified output capacity for up to 20 ms of power line failure, permitting the system to work without interruptions.

The peripheral unit meets the requirements of IEC 750 regarding safety, CISPR 22 Class B for radiated emissions, and HP environmental specifications (ETM C1 "Office").

Peripheral Unit Boards

The system is organized to house VXIbus or VMEbus boards. All of the application boards are VXIbus register-based, B-size boards. They use a common bus interface unit implemented in an ASIC (application-specific integrated circuit). Every board can run a self-test to determine its status according to VXIbus rules. A useful feature is self-configuration, which is implemented using the VXIbus MODID lines and the standard registers provided in the VXIbus A16 address space. Every board has its own address and a model code that represents its functionality. This allows the processor that controls the peripheral unit to determine the cardcage configuration and any board's status automatically.

The system boards are the slot 0 board and the processor board. The slot 0 board is required by the VXIbus standard to provide common resources to VXIbus subsystem slots 1 through 12. Slot 0 also provides the system trigger, which is used to synchronize the measurements, and basic system resources such as the system clock and the bus arbiter. The processor board is a VMEbus B-size board and is responsible for raw data collection from the application boards. Alarms are stored in local memory waiting to be polled by the first-level processor, while the raw data is processed to obtain the G.821 parameters which are then stored until collected by the first-level processor. Up to 80 digital streams (this is the case for ten 2-Mbit/s boards), or up to 160 alarm points (this is the case for ten alarm boards) can be processed in a peripheral unit. In the case of a loss of communications between the first-level processor and the peripheral units, all of the G.821 records can be stored for up to 12 hours; each monitoring point is allocated a buffer for 50 records, a record consisting of type of alarm, start time, and stop time. The processor software can be updated using a DOS-based personal computer connected to the J4 RS-232 connector.

The communication controller and communication boards are used to extend the bus to remote sites. Any communication board can drive two modems at a maximum speed of

9600 bits/s. The communication controller is basically a processor board with specialized communication software and can control up to 16 remote peripheral units.

HP E3560 application-specific boards include a 140-Mbit/s monitor board, a 34-Mbit/s monitor board, an 8-Mbit/s monitor board, a 2-Mbit/s monitor board, a counter board, and an alarm board. The monitor boards can recover a CCITT G.703 signal. They can analyze the standard CCITT G.702 hierarchy starting with 2 Mbits/s (2048 kbits/s) or a pseudorandom (PRBS) signal according to CCITT O.151. The 2-Mbit/s monitor board analyzes up to eight independent data streams. Measurements can be made using code violations, frame alignment errors, CRC-4 errors, or PRBS errors. The 8-Mbit/s, 34-Mbit/s, and 140-Mbit/s monitor boards can analyze a single data stream. They measure code violation errors, frame alignment errors, or PRBS errors. The counter board can count events at a maximum speed of 1 MHz. Its eight independent inputs can accept signals from –5V to +5V and its maximum sensitivity is 100 mV. The alarm board has 16 independent inputs. When an input is set for high impedance, it can collect events from –60V to +60V, and maximum sensitivity is 1V. It can also be set to measure open or closed contacts or current loops.

Any of the monitor boards can demultiplex a tributary data stream inside the data stream being processed and send it to the VXIbus local bus lines. Any board in the system can sink and/or bypass these lines to the next board. This means that a group of boards, say one 34-Mbit/s, one 8-Mbit/s, and one 2-Mbit/s, can act as shared demultiplexers for the other monitoring boards, which can send these boards their signals to be demultiplexed.

Two special application boards are also available. These are basically the standard 8-Mbit/s and 34-Mbit/s monitor boards without the G.703 interface. They perform the demux function while analyzing the streams coming from the local bus. This can be economically convenient when a demux feature is shared among many monitored streams.

Another common resource is the scanner board, which contains two 4:1 analog multiplexers. The high bandwidth of this board allows the multiplexing of signals up to 140 Mbits/s. Up to three multiplexers can be cascaded. A scanner board can be used to scan a group of digital data streams, connecting one at a time to a monitor board. This can lower the cost per data stream but has the disadvantage that no data stream is monitored continuously.

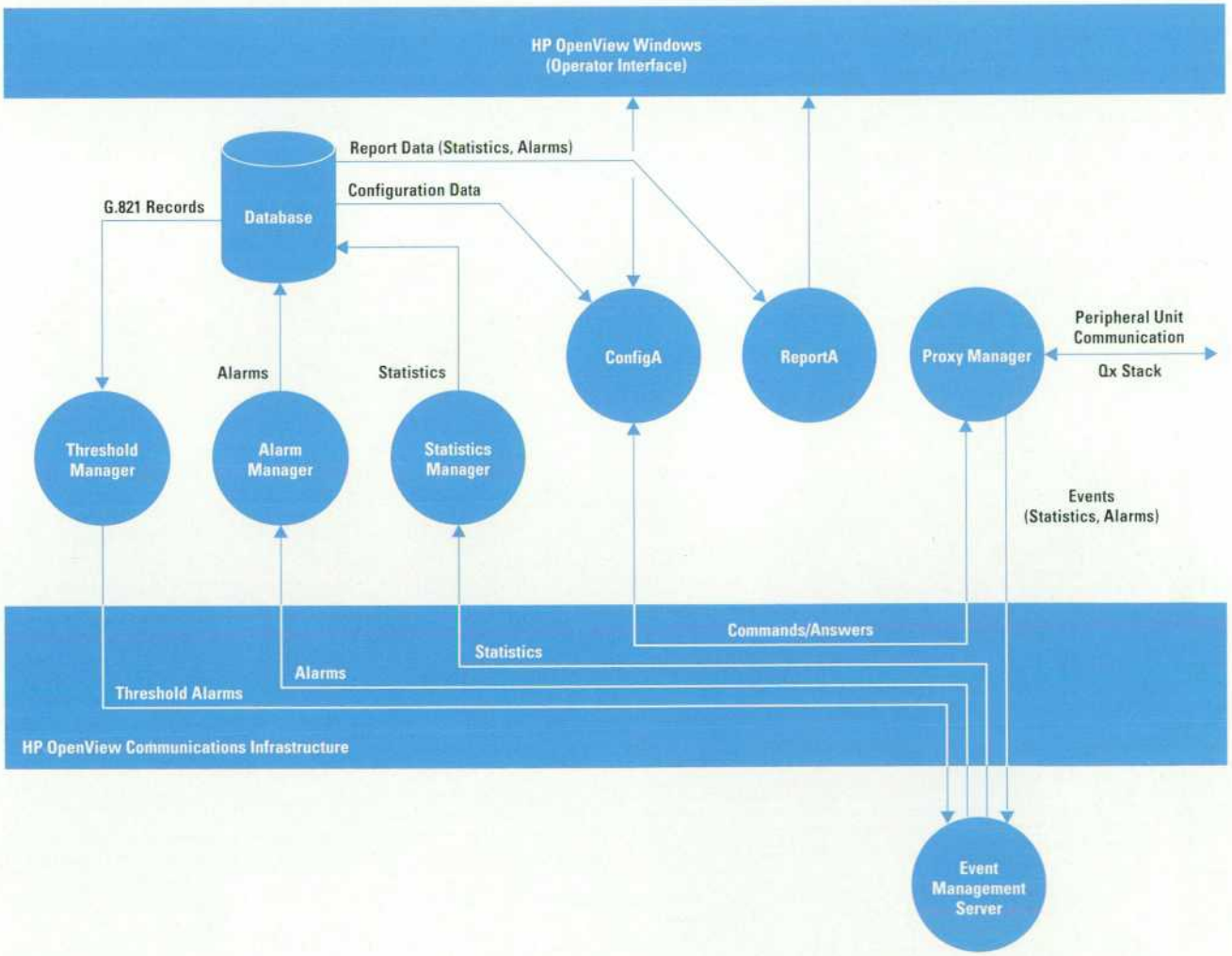


Fig. 6. HP E3560 software architecture. The software is based on the HP OpenView network management software.

HP E3560 Software

The HP E3560 software is based on HP OpenView,[1] as shown in Fig. 6. The applications and managers that form the software environment are divided into four packages:

- Base software
- Communication software
- Presentation software
- Threshold manager software.

Base and Communication Software

The base software lets the user manage the peripheral units by acting on the manager map (Fig. 7). This is actually a set of maps, each exposing a particular level of detail in the system architecture, from a high-level view (peripheral unit level) down to the board level and the individual monitoring channels inside the boards. Another set of maps called the user map is available for surveillance (see "Presentation Software," page 97).

By selecting an object with the mouse and using the appropriate menus, the system manager can set up the peripheral units and start and stop the monitoring activities. This feature is available both on the peripheral unit level (a sort of big switch that turns on and off the monitoring capabilities of an entire peripheral unit) and on the single-channel level. Thus it is possible to enable or disable the monitoring of a single data stream or alarm input.

The part of the user interface not directly handled by HP OpenView is managed by the ConfigA application, which translates user requests into the appropriate primitives and posts them to the HP OpenView communication infrastructure to be sent outside the workstation using the Qx protocols. Since this is not one of HP OpenView's native stacks, the task is performed by a proxy manager (which is part of the communication software). The proxy manager takes care of the translation between the HP OpenView primitives and the Qx primitives. Furthermore, the proxy manager manages all associations with the peripheral units and ensures the correct addressing of each outgoing request. The proxy manager returns incoming responses to the application or manager waiting for them, and sends event report indications to the event management server (see below).

The proxy manager also manages data link faults. The OSI layer 2 protocol continuously polls the peripheral units and can detect any disconnection resulting from line breakage, peripheral unit failure, or some other cause. In this case, it issues a DL-DISCONNECT request towards the OSI stack's upper layers. The layer 7 service element responsible for associations forwards this request to the proxy manager as a PROVIDER-ABORT indication. The proxy manager translates this into a format understandable by HP OpenView by issuing a particular event report indication to the fault manager, a part of the base software not shown in Fig. 6, which signals the fault by changing the object's color on the network map.

The fault manager is one of three managers that handle incoming events. The other two are the alarm manager and the statistics manager, which receive the alarms and the G.821 data, respectively, from the peripheral units and store it in the database. These managers use the services of the event management server. Each manager creates a filter which is used by the event management server to route the various events to the managers that are waiting for them.

Since data handled by these managers requires a large amount of storage (customers typically ask for 1 to 2 years' storage), it was deemed better not to use the database embedded in HP OpenView, but to provide instead an SQL database, which is also useful for report generation. Configuration information is also stored in the SQL database.

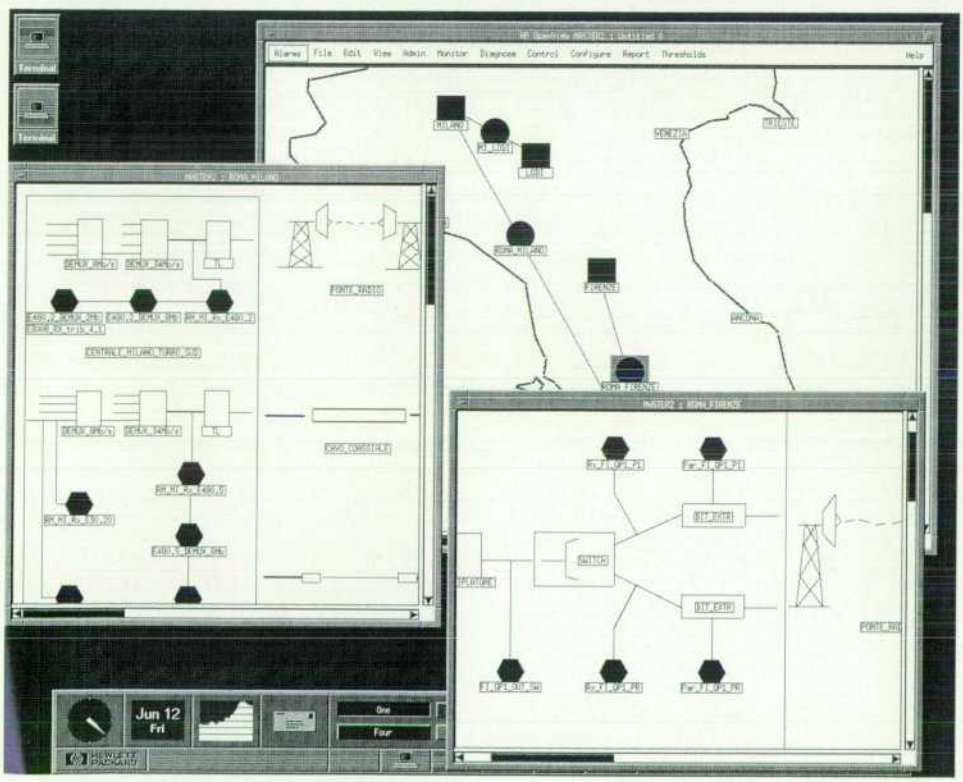


Fig. 7. HP E3560 manager map (top level).

It was a design choice that all information pertaining to the peripheral unit is kept in the peripheral unit itself. The first-level processor stores only logical information related to the peripheral unit (e.g., the data streams' names and creation dates) that doesn't correspond to any physical attribute of the device.

The alarm manager also changes the status and therefore the map color of the affected object. The HP E3560 uses the four colors allowed by HP OpenView. Each color is associated with a particular status of the affected object. The four basic status conditions are:

- Unknown. This status means that the object has not been created yet. It is known by the first-level processor, but not by the peripheral unit that contains it.
- Normal. The object gets this status after a successful creation, that is, after a create request has been issued to the peripheral unit and has been acknowledged by it.
- Warning. The object gets this status when an alarm indication has been received. When the alarm is turned off, the object's status changes back to normal.
- Critical. The object gets this status when a data link fault occurs, that is, when it is no longer possible for the first-level processor to act upon the object.

Database Management

Database management is part of the base software. It is handled by the ConfigA application and the alarm and statistics managers. The database structures consist of three main tables:

- Configuration tables store data regarding peripheral units and monitored data streams. As mentioned above, only logical parameters are stored in the first-level processor, such as the data stream name, its creation and disconnection dates, and the quality parameters of the stream (see "Threshold Manager Software" below).
- Alarm tables store alarm source, type, begin time, and end time.
- Performance tables store performance data coming from the peripheral unit. Each row of the table stores a record containing the error counts (ES, SES, DM, UAS) and three bit error rate indicators computed by the peripheral unit in slightly different ways.

The way in which performance data is managed is critical to the operation of the system. Elementary data coming from the peripheral units occupies a lot of disk space. With some hundred streams being monitored, disk space can be filled in a few months. As a compromise between data storage and disk space, an aggregation technique was developed to maintain data for a longer period at the price of reduced data granularity.

The elementary records are kept in the database for a period of time T1 (expressed in days), which can be defined by the user during the installation of the system. Each day, a background process combines the elementary records into a single daily record. For G.821 parameters (ES, SES, etc.) this is done by simply summing all of the stored values. For BER parameters, it is done by averaging all of the stored values. After T1 days, the older elementary records are removed from the database. Storing the daily records before time T1 speeds up daily report generation at the expense of a small amount of extra storage.

The advantage of this operation to the user is a reduction of the disk space needed for the database. The disadvantage lies in the loss of resolution resulting from the aggregation process: the older data cannot be viewed with 15-minute resolution, but only with 1-day resolution.

The daily records are kept in the database for a period of time T2. At the end of each month, a second aggregation

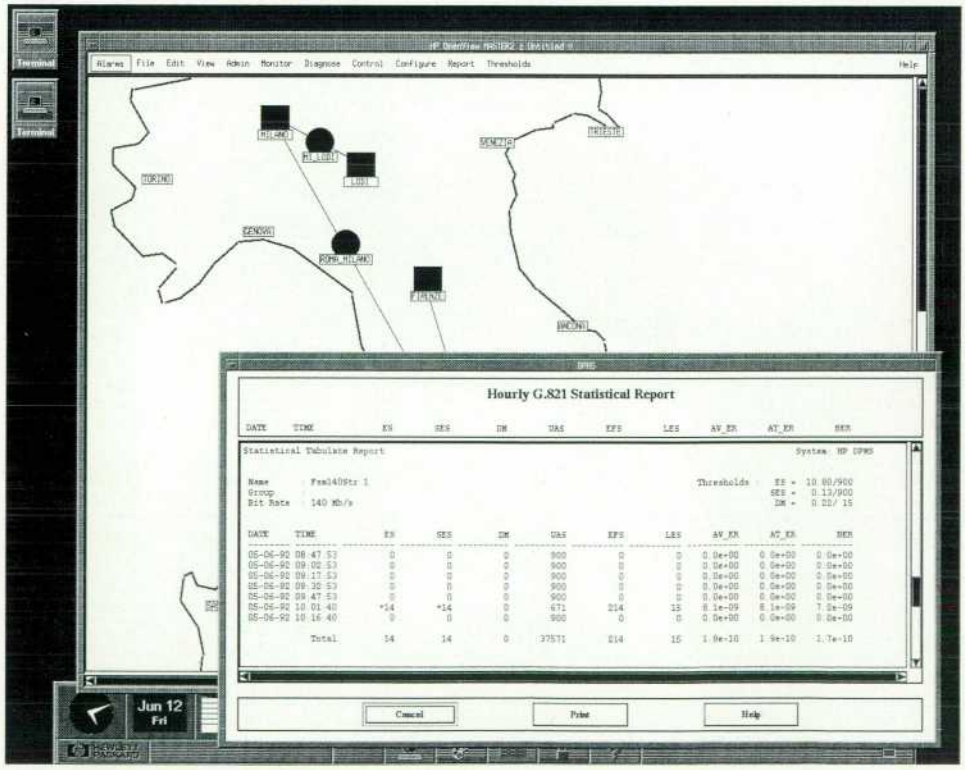


Fig. 8. Hourly G.821 statistical report. ES, SES, DM, UAS, and BER are defined on page 89. EFS = error-free second. LES = local errored second. AV_ER and AT_ER are bit error rates (BER) computed in different ways.

takes place, collapsing the daily records of the oldest month into a single monthly record. After T2 months, the oldest daily records are removed from the database.

Finally, after a period of time T3, the oldest records are deleted from the database.

T1, T2, and T3 can be defined during system installation, but it must be realized that their values have different impacts on disk space. Elementary records have the highest storage requirements, while daily and monthly records play only a secondary role. Therefore, a higher value of T1 means reports with higher resolution for a longer time, but it also means more disk space and cost.

Presentation Software

The presentation software is implemented using the ReportA application. Reports are divided into two families: statistical and alarm reports, which show performance and alarm data, respectively. Performance data can be displayed in three formats: hourly (Figs. 8 and 9), daily (Fig. 10), and monthly. The hourly display shows data as it is stored in the database (elementary data). The daily and monthly displays show aggregated data. Various reporting options let the user make reports on a single data stream or on a group of data streams, showing absolute or percentage values. The user can define thresholds for ES, SES, and DM and have the report flag all records having one or more values over a threshold. Optionally, the user can ask for a report displaying only values that exceed system or user-defined thresholds.

The ReportA application user interface makes extensive use of X Window panels and HP OpenView maps. The user's selections are translated into SQL queries and the results are formatted in a file and displayed. The file can also be printed. The HP E3560 design philosophy allows system users, who perform reporting activities, to make use of user maps.

These contain only objects such as data streams or alarm inputs and are not cluttered with configuration objects (peripheral units, boards, and so on), which are not pertinent to the surveillance task.

Fig. 11 shows a typical alarm report.

Threshold Manager Software

The threshold manager software has the purpose of long-term surveillance of the monitored data streams according to CCITT Recommendation M.550.

Each data stream can be assigned quality parameters by the operator. These consist of a quality classification (high, medium, or low grade) and the type classification of the link (path, section, or line section). These characteristics are processed to produce a set of thresholds that mark the data stream performance limits: the higher the declared quality of the data stream, the lower the limits. The calculated thresholds also depend on another variable, the operational status of the data stream, which can be declared as in service, out of service, or repaired. The software automatically sets the thresholds according to the operator's declarations.

Whenever a data stream is placed under threshold manager control, its performance data (ES, SES, and DM values) is periodically read from the database. The period, called the step, can be defined by the user, from 15 minutes to 1 day. The performance data is accumulated in the threshold manager's private registers. This process continues for a user-defined period of time called the reset period, ranging from a minimum period, which is equal to the step, to a maximum of one month. If during the reset period any one of the accumulated values crosses the calculated threshold, the threshold manager generates an alarm, which is displayed in the same manner as the alarms coming from the peripheral units.

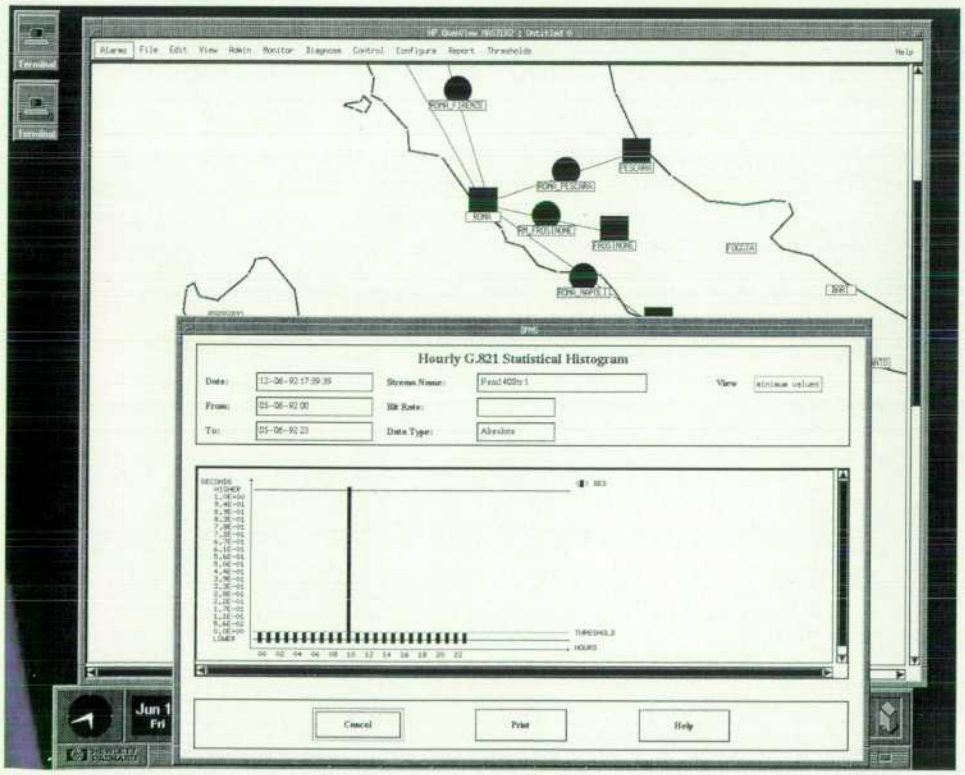


Fig. 9. Hourly G.821 statistical histogram.

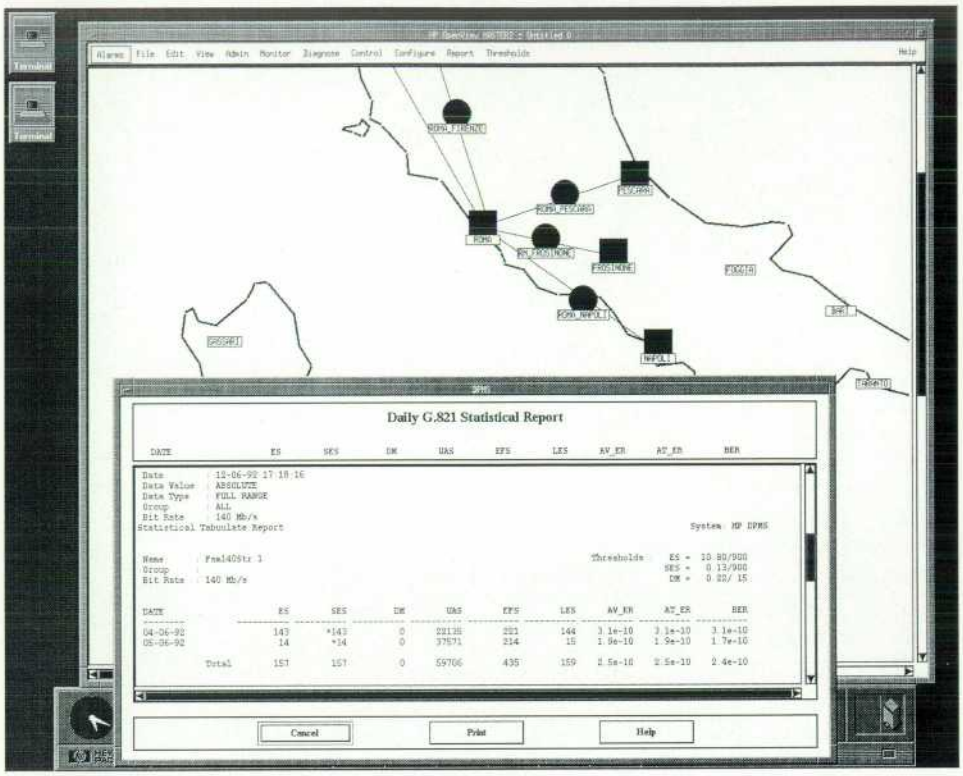


Fig. 10. Daily G.821 statistical report.

Recommendation M.550 classifies circuit performance as normal, degraded, or unacceptable. This classification takes the reference performance objective (RPO, defined earlier) as its reference point and scales it by a factor depending on the circuit type to determine the thresholds of degraded and unacceptable performance. For a digital path like the one used as an example at the beginning of this article, the scaling factor for the degraded performance threshold is 0.75.

Thus, the performance of such a path is defined as degraded (D) when one or more of the quality parameters ES, SES, or DM crosses the corresponding threshold:

$$\text{D ES} = 0.75 \times \text{RPO(ES)}$$
$$\text{D SES} = 0.75 \times \text{RPO(SES)}$$
$$\text{D DM} = 0.75 \times \text{RPO(DM)}$$

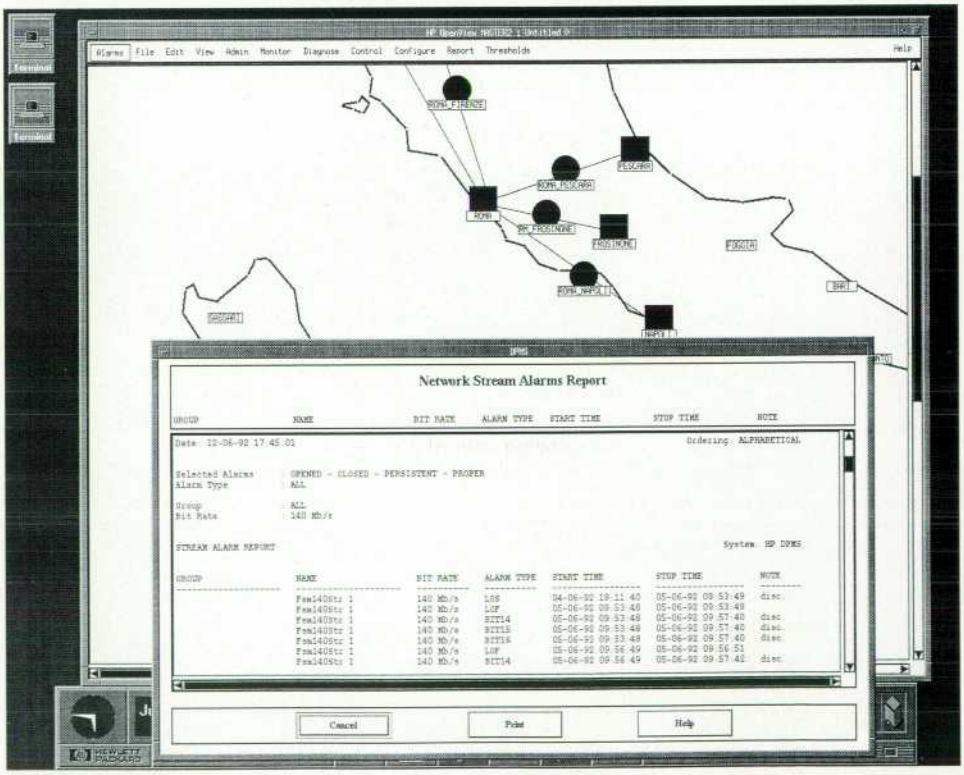


Fig. 11. Network stream alarms report.

Demux Software

One of the most powerful capabilities of the HP E3560 peripheral unit is the possibility of taking a data stream connected to the peripheral unit's front panel and extracting one of its tributaries. The extracted data stream can be fed to another monitoring board, which in turn can monitor and recursively demultiplex the tributary. Thus, starting from a 140-Mbit/s data stream, one can demux down to the 2-Mbit/s level.

This capability is exploited to give three different demux modes: slow, medium, and fast demux (these features are included in the base software).

Slow demux can be used to monitor a selected tributary continuously. This is nothing more than what is normally done with the monitoring boards, except that the controlled data stream doesn't come from the transmission equipment or the DDF (digital distribution frame), but is extracted from a hierarchically higher data stream. All the operator has to do is select a demux source, choose the Start Slow Demux option in the Demux menu, and then, helped by a dialog box, ask for the pattern that leads to the desired tributary. An M-ACTION primitive is then sent to the peripheral unit, which locates the required resources (mainly free monitoring channels) and then communicates to the first-level processor the start of the demux action, sending back the physical addresses of the selected channels. These channels appear to the operator as symbols on the map, which the operator is asked to name to identify the selected tributaries during the demux operations. It is also possible to assign a group name to the whole demux chain (the set of streams that form the patterns) which makes it possible to extract the G.821 report with a single query.

Fast and medium demux are fault localization tools: their philosophy is the same as slow demux, but they are implemented slightly differently. The idea behind these operations is to explore the "tributary tree" contained in a data stream to find possible problems.

Fast demux starts from a selected data stream and scans the tree down to the 2-Mbit/s level, reporting the status of each tributary. If the status is not OK, the most severe alarm detected during the scanning is reported. This operation is very quick and is automatically performed by the peripheral unit. After about 10 seconds, the report is ready on the screen.

Medium demux operates in the same way, but the time dedicated to each tributary can be chosen by the operator (from 1 to 60 seconds). Since this results in a longer operation, the result is more accurate. The report also gives the BER estimated during the observation of the tributary.

Acknowledgments

The HP E3560 is the result of the common efforts of many engineers and managers who put together ideas, experiences, efforts, and enthusiasm. The authors wish to thank all the people who, at different stages of the project, made contributions. These include M. Verde and M. Ferrari for product management, and G. Cattin and M. Mason for project management. Special thanks go to the R&D team, in particular to: M. La Manna, R. Borghetto, T. Santi, F. Vendramin, P. D'Emilio, P. Bottaro, S. Mingardo, A. Canella, M. Dal Sasso, S. Tosato, G. Santi, A Pedron, C. Carrara, L. Vanuzzo, F. Mallamo, R. Cappon, F. Brasola, M. Ongaro, P. Fasolato, R. Ronzani, M. Ballotta, R. Silvestrini, G. Rosina, Mark Smaczny, and John Spink.

References

1. *Hewlett-Packard Journal*, Vol. 41, no. 2, April 1990, pp. 51-91.
2. L.A. DesJardin, "VXIbus: A Standard for Test and Measurement System Architecture," *Hewlett-Packard Journal*, Vol. 43, no. 2, April 1992, pp. 6-14.

Authors

October 1992

6 HP 4980 Network Advisor

Edmund G. Moore

A graduate of the University of California at Berkeley in 1979 with a BS degree in electrical engineering and computer science, Ed Moore joined HP's Colorado Telecom Division that same year. He was a program and section manager for the HP 4980 Network Advisor project and is now the quality manager for the division. In the past he has worked as either a development engineer or a project manager for other HP 4900 Series protocol analyzers. He is a member of the IEEE and his professional interests are computer networking and network protocols. Ed was born in Augusta, Georgia, is married, and has three children. He enjoys rafting and racquetball in his spare time.

11 Embedding Artificial Intelligence

Scott Godlew

Scott Godlew was the technical lead for the Fault Finder component of the HP 4980 Network Advisor. He is now a member of the OpenView development team at HP's Colorado Networks Division. He received his BS degree in computer science from Michigan Technological University in 1984. He joined HP's Colorado Telecom Division (CTD) that same year. At CTD he worked on WAN and LAN protocol analyzer development. He recently received his MS degree in intelligent systems from National Technological University. He grew up in Michigan, but now claims Colorado as his home where he enjoys skiing, camping, and exploring the colorful history of the mineral-rich Colorado mountains.

Rod Unverrich

Software engineer Rod Unverrich joined HP's Telecom Division in 1989. He received a BS degree in electrical engineering and computer science from the University of Wisconsin at Madison in 1986, and an MS degree in computer science from National Technological University in 1991. For the HP 4980 Network Advisor project he worked as the technical lead engineer for the token ring Fault Finder. He also worked as an application engineer for the Network Advisor. Before joining HP he was a software engineer with IBM working on real-time software development for a data switching product. He is listed as an inventor for three pending patents resulting from his work on the Network Advisor. He is active in a local school program that instructs teachers and students about basic electronics. Born in Waukesha, Wisconsin, he is married and awaiting the arrival of his first child. He enjoys running, weightlifting, and hiking in his spare time.

Stephen Witt

Project manager Steve Witt joined HP's General Systems Division (GSD) in 1979 after receiving his BSEE degree from Michigan Technological University. He joined HP's Colorado Telecom Division (CTD) in 1980. AT GSD he worked on the I/O subsystem for the HP 300 business computer and at CTD he has worked on the HP 4953A and HP 4955A WAN protocol analyzers and X.21 state simulators. He was the project manager for the token ring Network Advisor and the token ring and Ethernet versions of the Fault Finder. He is a member of the IEEE and his professional interests include computer networks and local area networks. Born in Ottawa, Canada, he is married and has two children. He is active in his church and enjoys reading, swimming, hiking, photography, and astronomy. In his spare time he is building his own telescope.

22 HP 4980 User Interface

Thomas A. Doumas

Tom Doumas is a project manager at HP's Colorado Telecom Division. He received his BSEE degree in 1981 and his MSEE degree in 1984 from the University of Wisconsin at Madison. He joined HP in 1984. He was the project manager for the Network Advisor's system software. Before the Network Advisor he worked as a hardware, software, and production engineer for the HP 4952A protocol analyzer. Before joining HP he worked as a software designer at the TRACE center for augmentative communication at the University of Wisconsin at Madison. He was born in Memphis, Tennessee, is married, and has one son. He likes to bicycle and ski during his leisure time.

29 Analysis and Real-Time Environment

Sunil Bhat

Software design engineer Sunil Bhat joined HP's Colorado Telecom Division in 1989 after receiving an MS degree in computer science from Iowa State University at Ames. He received a Bachelor of Technology degree in computer science and engineering from the Indian Institute of Technology in Bombay, India in 1987. For the HP 4980 Network Advisor project, he was responsible for the design, development, and testing of many parts of the analysis and real-time (ART) environment. His professional interests include software systems design and performance optimization. He was born in Mangalore, Karnataka, India. He is married and his leisure activities include backpacking, skiing, and music.

34 Protocol Analysis

Rona J. Prufer

Rona Prufer is a member of the technical staff at HP's Colorado Telecom Division. She started with HP as a summer intern in 1980 and joined as a full-time employee at HP's Spokane Division in 1981. She has a BSEE degree (1981) from Washington State University and an MS degree (1989) in computer science from National Technological University. Besides working on the HP 4980 Network Advisor as a software engineer, she has worked as a production engineer for the HP 4937A transmission impairment measuring set, the HP 4938A network circuit access test set, and the HP 8600 Series of synthesized signal generators. She was named Distinguished New Engineer in 1992 by the Society of Women Engineers and received a technical achievement award at the 1991 Technical Women's Conference. She is a member of the Society of Women Engineers and is active in HP's K-12 educational program in Colorado Springs. She was born in Deer Park, Washington and enjoys soccer, scuba diving, reading, mountain biking, camping, hiking, climbing, and traveling in her spare time.

41 HP 4980 Mechanical Design

Kenneth R. Krebs

Product designer Ken Krebs worked on the package design for HP 4980A Network Advisor. Ken has worked as a product designer at HP's Colorado Telecom Division since joining HP in 1980. He has also worked on the package designs for the HP 4953A and HP 4952A protocol analyzers. Before joining HP he worked on the design of blood processing systems at Haemonetics Corp. He received a BA degree in economics and a BS degree in mechanical engineering from Stanford University in 1976, and an MS degree in mechanical engineering from Stanford University in 1980. Born in San Mateo, California, he is married and has three daughters. His leisure activities include soccer, woodworking, bicycling, and acting in a church drama group.

48 Microwave Transition Analyzer

David J. Ballo

David Ballo joined HP's Santa Rosa Systems Division as an R&D engineer in 1980, when it was the Signal Analysis Division. He contributed to the design of the HP 70902/3A IF modules and the HP 70700A digitizer module, and worked on the synthesized LO, modulation source, calibrator, and ADC of the HP 70820A microwave transition analyzer module. After leaving the R&D lab in 1991, he served as a manufacturing engineer for a year, and is now a product marketing engineer. A native of Bellevue, Washington, he received his BSEE degree in 1980 from the University of Washington. David is married. Away from the job, he plays bass in a klezmer band (eastern European folk music), has played guitar in rock bands, and together with coauthor John Wendler has brewed award-winning beer. He also enjoys sailing, cycling, backpacking, cross-country skiing, and photography.

John A. Wendler

R&D engineer John Wendler joined the HP Santa Rosa Systems Division (then called the Signal Analysis Division) in 1981 after receiving a pair of BS degrees in electrical engineering and mathematics from the California Polytechnic State University at San Luis Obispo. He has worked on the HP 70700A digitizer module and on digital signal processing (DSP) algorithms, firmware design, and digital design of the ADC memory and DSP circuitry for the HP 70820A microwave transition analyzer module. He is named as a co-inventor in a patent on reconstruction of sampled signals. In 1987 he received an MSEE degree from the University of Illinois at Urbana-Champaign. John was born in Washington, D.C. He is married and enjoys foreign travel, backpacking, and brewing his own beer.

63 Transition Analyzer Design

Michael Dethlefsen

Mike Dethlefsen received his BSEE degree from the University of California at Berkeley in 1983 and joined HP's Signal Analysis Division (now the Santa Rosa Systems Division) the same year. He designed the third converter for the HP 70907A external mixer interface and the 321.4-MHz calibration source for the HP 70907B, served as a production development engineer responsible for the microwave phase-locked loops of the HP 70900A local oscillator module, and worked on millimeter-wave mixers and receivers. For the HP 71500A microwave transition analyzer, he has done application development and support, particularly for pulsed-RF measurements. In 1992 he received an MSEE degree from the University of California at Davis. Mike was born in San Luis Obispo, California. He is married and is a pilot with an interest in aerobatics. He also enjoys backpacking and duck hunting.

John A. Wendler

Author's biography appears elsewhere in this section.

72 HP VEE

Douglas C. Beethe

A senior system architect at HP's VXI Systems Division, Doug Beethe was the principal architect for the HP VEE software. Several pending patents have resulted from that work. Doug received his BSME degree from Kansas State University and joined the HP Desktop Computer Division in 1979. Before HP VEE, he worked in thin-film and NMOS IC process management. He is a member of the IEEE Computer Society, and his professional interests include modeling, simulation, and the control of physical systems. Born in Hays, Kansas, Doug is married, has four children, and lives on a 200-acre farm. He volunteers for search and rescue activities, participates in local music and drama groups, and enjoys outdoor recreation, flying, travel, and building restoration projects.

William L. Hunt

Bill Hunt received his BS degree in computer engineering from Iowa State University in 1980 and his MS degree in computer science from Colorado State University in 1989. He joined HP's Corvallis Division in 1980 and is now a software engineer at the VXI Systems Division. He was responsible for software development for HP VEE. On previous projects, he developed an MS-DOS I/O library, firmware for the HP 110 computer, and I/O cards for HP Series 80 computers. He has a special interest in object-oriented programming and has published papers on that subject. Born in Summit, New Jersey, Bill is married and enjoys bicycling and skiing.

78 HP VEE User Interface

William L. Hunt

Author's biography appears elsewhere in this section.

84 Dataflow Architecture

Douglas C. Beethe

Author's biography appears elsewhere in this section.

89 Digital Network Monitor

Giovanni Nieddu

Gianni Nieddu is a system engineer at HP's Necsy Telecommunications Operation. His professional interests include protocols and real-time embedded systems. He studied electronic engineering at the University of Padova (Padua), graduating in 1984. He joined Necsy in 1987 and was responsible for the system architecture and software of the HP E3560 digital performance monitoring and remote test system. Born in Sacile, Italy, he served in the Italian Alpine troops for a year in 1985. He is married and enjoys mountain climbing and trekking, skiing, and games in general.

Fernando M. Secco

A system engineer with HP's Necsy Telecommunications Operation, Fernando Secco studied electronic engineering at the University of Padova (Padua), graduating in 1982. He served a year in the Italian infantry in 1983 and joined Necsy in 1989 after several years with Telettra SpA. He was responsible for the design of the peripheral units for the HP E3560 digital performance monitoring and remote test system. Fernando comes from Piazzola sul Brenta, Padova. He is married and has one child.

Alberto Vallerini

Alberto Vallerini is hardware system manager for the HP E3560 digital performance monitoring and remote test system. Previously, he was hardware system manager for the HP 3788A error performance analyzer. He joined HP's Necsy Telecommunications Operation in 1988 and has a degree in electronic engineering from the University of Padova. Alberto was born in Lendinara, Rovigo. He did his military service as an official in the transmission service. He is married and enjoys reading, music, and theater.

103 G-Link Chipset

Chu-Sun Yen

Chu Yen was project manager for the G-link chipset at HP Laboratories. With HP Laboratories since 1961, he previously managed high-speed analog ICs, Ethernet transceiver and cable simulation, and infrared network projects, and was a design engineer on projects dealing with the HP 8405A vector voltmeter and the HP 35 calculator. A member of the IEEE, he has authored 20 professional papers on circuits, phase-locked loops, instruments, and communications links, and is named as an inventor in three patents on a sampling phase-locked loop, the G-link phase-locked loop, and a dc-to-dc converter. He received his BSEE degree in 1955 from Taiwan University, his MSEE degree in 1958 from the University of Florida, and his PhD degree in 1961 from Stanford University. He is married and has three children.

Richard C. Walker

Rick Walker is a principal project engineer at HP Laboratories, specializing in phase-locked loop theory and high-speed circuit design. He joined HP Laboratories in 1981 and has contributed to broadband cable modem design, solid-state laser characterization, and the gigabit-link project. Born in San Rafael, California, he received his BS degree in engineering and applied science from the California Institute of Technology in 1982 and his MS degree in computer science from California State University at Chico in 1992. He has authored 16 professional papers and his work has resulted in six patents and two pending patents, all in the areas of high-speed links and circuit design. He's a member of the IEEE. He's also an advanced class amateur radio operator (WB6GVI) and a private pilot, plays bass guitar and five-string bluegrass banjo, and cultivates several dozen kinds of carnivorous plants in a backyard greenhouse.

Patrick T. Petruno

Now R&D section manager for link products at HP's Communications Components Division, Pat Petruno was project manager for the G-link chipset. A native of Allentown, Pennsylvania, he attended Pennsylvania State University, receiving his BSEE degree in 1976 and his MSEE degree in 1978. After joining HP in 1978, he designed bipolar transistors, Darlington amplifiers, and digital circuits, and served as project manager for wideband amplifiers, AGCs, decision circuits, counters, and multiplexers. He has coauthored three papers on high-speed silicon circuits and participates in Serial-HIPPI and ATM (asynchronous transfer mode) standards activities. Pat is married, has two children, and serves as a Cub Scout den leader. His interests include astronomy, astrophotography, swimming, and softball.

Cheryl Stout

Cheryl Stout has been a member of the technical staff of HP Laboratories since 1983. She has done research and design for gallium arsenide and silicon high-speed multiplexers, optical receivers, and the gigabit-link chipset. She is a member of the IEEE and has authored conference papers on high-speed multiplexers and the G-link chipset. Born in San Jose, California, she received her BSEE degree from California State University at San Jose in 1979 and her MSEE degree from the University of California at Berkeley in 1983. Before coming to HP, she developed optical communication products at Plantronics, Inc. Her interests include mountaineering and natural history.

Benny W.H. Lai

Engineer/scientist Benny Lai joined the HP Microwave Semiconductor Division (now the Communications Components Division) in 1981. He designed the HP HDMP-2003/4 decision circuits and the HDMP-2501 clock recovery data retiming circuit, and did circuit design and layout and high-speed testing for the G-link chipset. He has authored four papers on his designs and his work has resulted in one retiming circuit patent and two pending patents on CIMT coding and a unity-gain positive-feedback integrator. A graduate of the University of California at Berkeley, he received his BSEE degree in 1982 and his MSEE degree in 1983. He was born in Hong Kong, is married, and enjoys woodworking, gardening, and skiing.

William J. McFarland

HP Laboratories principal project engineer Bill McFarland received his BSEE degree from Stanford University in 1983 and his MSEE degree from the University of California at Berkeley in 1985. With HP since 1985, he has designed high-speed ICs for digital test instruments in silicon bipolar, gallium arsenide, and heterojunction transistor technologies, and has done research on bit error rate testers and pulse generators. As a member of the G-link project, he served as technical editor of the Serial-HIPPI specification. He is the author or coauthor of a dozen technical papers and is named as an inventor in two patents related to bit error rate testers. Bill was born in Milwaukee, Wisconsin. His interests include bicycling, playing guitar, and home brewing.

G-Link: A Chipset for Gigabit-Rate Data Communication

Two easy-to-use IC chips convert parallel data for transmission over high-speed serial links. A special encoding algorithm ensures dc balance in the transmitted data stream. A binary-quantized phase-locked loop is used for clock recovery. An on-chip state machine manages link startup automatically.

by Chu-Sun Yen, Richard C. Walker, Patrick T. Petruno, Cheryl Stout, Benny W.H. Lai, and William J. McFarland

The last decade has seen a tremendous increase in computing power with only modest advances in the bandwidth of the data links used to interconnect these computers. Between 1982 and 1992, the speed of a high-performance engineering workstation has increased from 0.5 MIPS (million instructions per second) to 100 MIPS, an increase of over two orders of magnitude. In that same period of time, computer network bandwidths have gone from Ethernet at 10 Mbits/s to FDDI at 100 Mbits/s, an increase of only one order of magnitude. In addition to faster computers, other factors, such as the widespread use of multimedia applications, will put pressure on network bandwidths, threatening to create an I/O bottleneck for modern computing systems.

Unlike computer systems, serial links cannot exploit parallelism and must run at proportionally higher rates for each increment in performance. At clock rates below about 100 MHz, traditional printed circuit board design techniques can be used to implement link circuitry with collections of packaged parts. But as link speeds approach the gigabit-per-second range, interchip timing skews make it impractical to build low-cost gigabit links in this way. Although long-haul telephone networks have used gigabit-rate data links for many years, these links use nonintegrable components and require adjustment and maintenance. Such systems are easily justified when the cost is amortized over millions of users but are too costly and complex for computer use.

To support the needs of computer and other generic data transport applications, the HP HDMP-1000 gigabit link (G-link) chipset has been developed. It is the first commercially available 1.4-Gbaud link interface in two chips, a transmitter chip and a receiver chip, requiring no external parts or adjustments.

The architecture of the G-link chipset greatly eases the job of the system designer. Communication between the chipset and the user's system takes place through a low-speed parallel interface. All gigabit-rate signals, with the exception of the serial electrical data stream, remain internal to the chips and are never routed on the printed circuit board. Thus the designer is able to use standard printed circuit board design techniques to deliver gigabit-rate performance. For fiber optic applications, the high-speed serial signals are easily connected to lightwave transmitter and receiver modules. To simplify the designer's job further, a link-management state machine controller implemented on the receiver chip insulates the user from many of the details associated with link startup and error monitoring.

The chipset was designed in HP's 25-GHz f_T silicon bipolar process and incorporates patented circuit techniques developed at HP Laboratories, namely the encoding scheme and the phase-locked loop circuit. These new techniques, described later in this paper, represent departures from traditional telecommunication practice and have made practical the integration of an inexpensive and easy-to-use gigabit-rate chipset.

Overview

Fig. 1 shows a typical G-link application supporting a full-duplex interconnection between two hosts. One transmitter and one receiver chip are used for each end of the link.

From the user's viewpoint, the chipset behaves as a "virtual ribbon cable" for the transmission of parallel data over serial links. Parallel data is serialized by the transmitter chip and deserialized by the receiver chip into the original parallel form. The chipset hides from the user all the complexity of

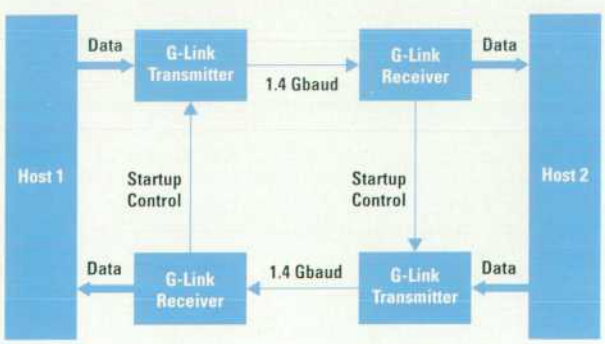


Fig. 1. A duplex link built with the HP HDMP-1000 gigabit link (G-link) chipset.

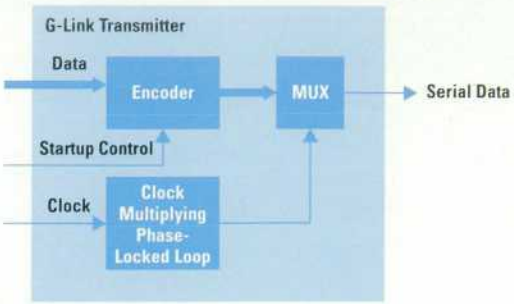


Fig. 2. Simplified transmitter chip block diagram.

encoding, multiplexing, clock extraction, demultiplexing, and decoding needed for high-speed serial data transmission.

The transmitter chip (Figs. 2 and 3) accepts the user's parallel data word and clock. The word-rate clock is internally multiplied up to the serial rate in the transmitter chip phase-locked loop. This high-speed serial clock is used to multiplex the encoded data. The encoding algorithm, called *conditional inversion with master transition*, or CIMT,[1] creates a frame* for data transmission by appending four coding bits to each input data word. The resulting frame is then transmitted in either normal or inverted form,[2] as necessary, to maintain dc balance of the serial bit stream for transmission over optical links or coaxial cables. This CIMT line code distinguishes itself by being efficient and simple to implement compared to other line codes such as 8B/10B.

To support modern network protocols, the chipset allows the transmission of three different types of frames. Generic user data is transmitted with *data frames*. *Control frames* are the second type of frame, and are used for the transmission of information that should be treated separately from data, such as packet headers. *Fill frames* are the third type of frame, and are sent automatically by the link during startup and to maintain synchronization when the user has neither data nor control information to send.

In the receiver chip (Figs. 4 and 5), the clock and frame alignment are extracted from the incoming data stream with a phase-locked loop. The data is then demultiplexed and decoded back to its original parallel form. In addition to these basic functions, the receiver chip also includes a state machine controller, which performs an end-to-end handshake and provides both bit and frame synchronization. This handshake avoids the false lock problems that are typical with clock extraction circuits that accommodate a wide range of clock frequencies.

An unconventional "bang-bang" phase-locked loop[3] is used in the transmitter and receiver to provide adjustment-free bit retiming at very high data rates. Using the special master transition built into the line code, the phase-locked loop provides frame synchronization without the periodic insertion of special frame synchronization words.

A very compact chip layout was achieved by using three layers of metal and a quasi-gate-array ECL design methodology.

* In this paper, a frame is defined as an encoded input word.

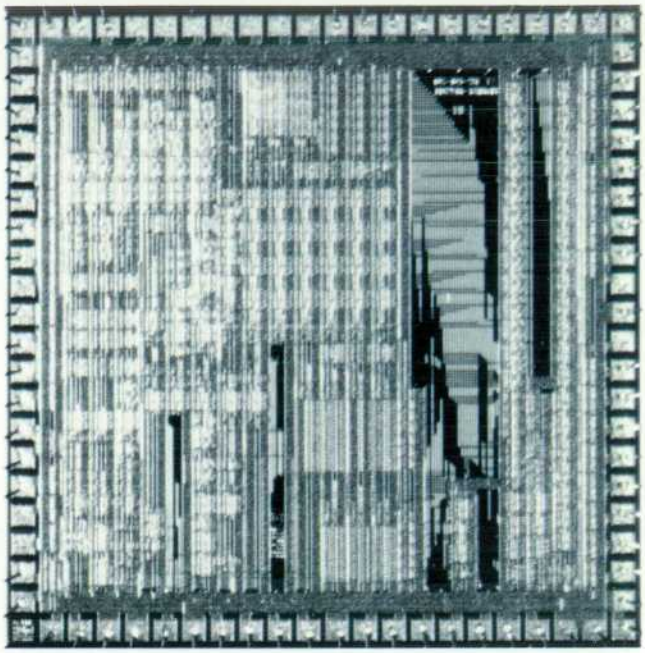


Fig. 3. Photomicrograph of the transmitter chip.

The 68-pin surface-mount package (Fig. 6) is designed to maintain good performance for 1.4-GHz signals.

The key features of the chipset are:

- Parallel ECL bus interface
- 16 or 20 bits wide, pin selectable
- Flag bit usable as extra data bit (17th or 21st)
- CIMT encoding and decoding
- Ac/dc coupled
- 110 to 1400 Mbaud serial line rate
- On-chip phase-locked loops for transmitter clock generation and receiver clock extraction
- Local loopback mode for troubleshooting
- Single –5V ±10% supply voltage
- 2W power dissipation per chip (typical)
- Can be used with fiber optic links
- On-chip equalizer for use with coaxial cable
- Standard 68-pin CQFP (ceramic quad flat package).

Because of the simplicity and flexibility of the G-link chipset, it can be used for a wide variety of applications, including computer backplanes, video distribution, peripheral channels, and networks.

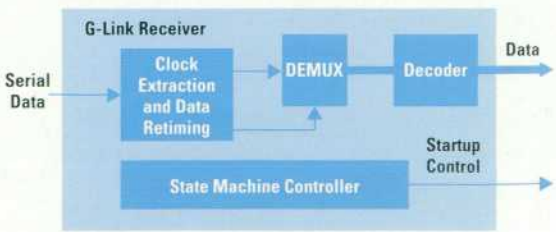


Fig. 4. Simplified receiver chip block diagram.

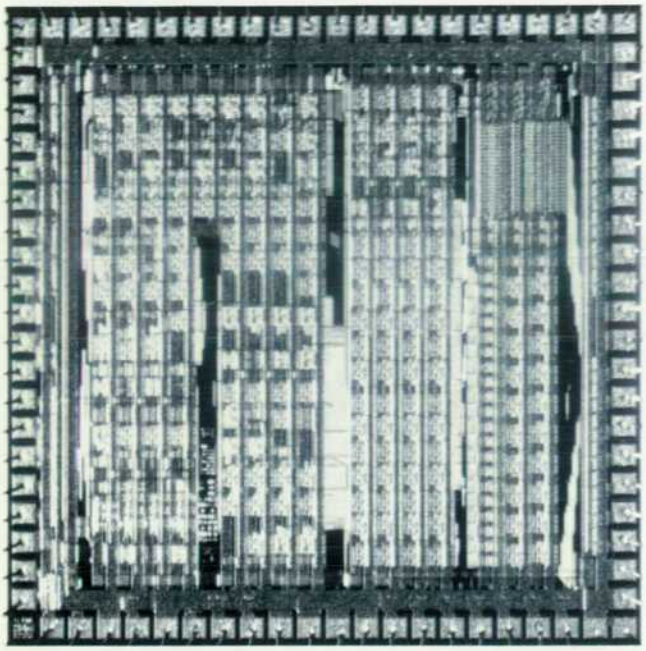


Fig. 5. Photomicrograph of the receiver chip.

G-Link Line Code

Many coding schemes have been developed to allow communication of information over various types of channels. In synchronous communication links, clock and framing information must be transmitted along with data in such a way that the clock and data can be recovered at the receiving end of the link. Therefore, it is necessary for the transmitted encoded serial bit stream to have enough embedded clock information for the receiver to recover the serial clock. There must also be some method of frame alignment so that the boundaries of a frame can be located at the receiver.

In optical links, it is desirable to ac couple the data signals to simplify laser bias circuitry and optical receiver design. This is also true in repeater design, since the components are commonly ac coupled. A problem with ac coupled systems is that the baseline will shift when the transmitted digital data is not dc balanced. This shift makes detection difficult and degrades the system noise margin. To overcome this problem, arbitrary data is typically encoded before transmission to achieve dc balance. The receiver restores the data to its original form by decoding.

In the G-link chipset, the CIMT coding scheme performs the following tasks:

- The transmitter chip supplies a master transition in every frame for clock recovery and frame alignment at the receiver.
- Frames are conditionally inverted as necessary to maintain dc balance.
- Information is provided in the transmitted frame about the type of frame transmitted and whether or not the frame was inverted.
- At the receiver, decoding is done to determine what type of frame was received and whether or not the frame was inverted.
- If the frame was inverted at the transmitter, it is inverted again at the receiver to restore the information to its original form.
- The receiver performs error checking on portions of the frames to detect loss of lock.

This method of encoding and decoding has several advantages:

- Clock information is available in each frame, indicating both phase and frequency alignment.
- There is no need for the user to send any special characters to indicate the start of a new frame. The G-link chips perform frame alignment transparently.
- There are no restrictions on the user's input bit patterns. Dc balance is maintained by frame inversion and a maximum run length is guaranteed by the master transition.
- By checking for framing errors, the receiver can detect loss of lock and reinitiate the link startup process. (A discussion of link startup can be found under "Startup State Machine Controller" on page 109.)

Data is encoded by appending four extra coding bits (C-field) to the input data (D-field). The serial combination of the D-field and the C-field makes a frame. The user can choose to transmit either data frames or control frames. In addition, two types of fill frames are internally generated for transmission when there is no input supplied by the user or during startup. To maintain dc balance, data and control frames are either inverted or not inverted. Information about inversion and the type of frame is contained in the C-field. Unlike typical codes with fixed data width, the CIMT code can accommodate multiple data widths.

The G-link chipset is designed to transmit either 16-bit-wide or 20-bit-wide data words. Both the transmitter chip and the receiver chip have an input pin that allows the user to select

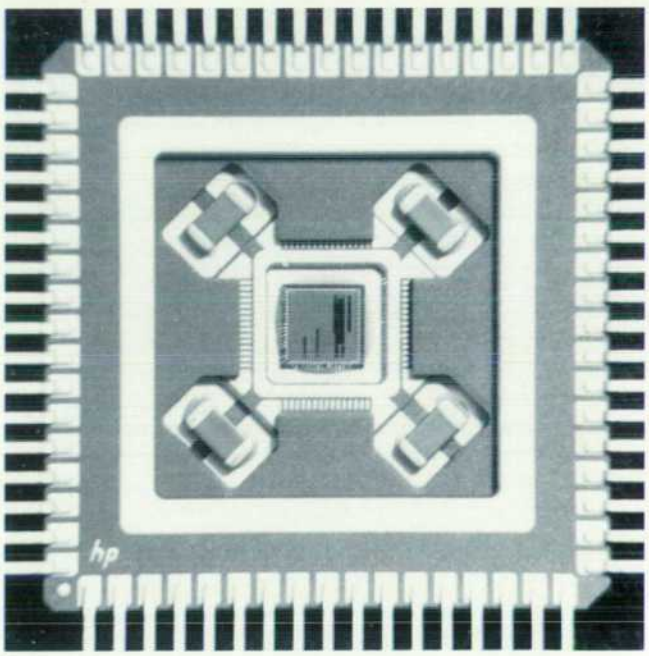


Fig. 6. Transmitter chip in 68-pin ceramic quad flat package (CQFP).

the parallel word width. There is also a flag bit, which can be used as an extra data bit. A frame consisting of the D-field plus the appended C-field is then either 20 or 24 bits long. In the case of control frames, two bits in the D-field are used for encoding, resulting in 14 or 18 bits available for transmitting information. The flag bit is obtained by selecting between different sets of coding bits in the C-field.

Table I shows the contents of different frames generated at the G-link transmitter for the case of 20-bit data. DAV (data input available) and CAV (control input available) are supplied by the user to indicate what type of user input is to be transmitted. If neither data nor control inputs are available, a fill frame is sent. FLAG is the additional flag bit input. D0 to D19 are the parallel inputs. INV is a logic signal internally generated on the transmitter chip that indicates whether the frame is to be inverted.

Table I
Contents of Different Frame Types for CIMT Encoding of 20-Bit Data

FLAG	DAV	CAV	INV	D-Field	C-Field (MT ↓)	Frame Type
X	0	0	X	111111111 10 000000000	00 11	Fill (FF0)
X	0	0	X	111111111 00 000000000	00 11	Fill (FF1L)
X	0	0	X	111111111 11 000000000	00 11	Fill (FF1H)
X	X	1	0	D0-D8 01 D9-D17	00 11	Control
X	X	1	1	$\overline{\text{D0-D8}}$ 10 $\overline{\text{D9-D17}}$	11 00	Inverted Control
0	1	0	0	D0-D19	11 01	Data, FLAG Low
0	1	0	1	$\overline{\text{D0-D19}}$	00 10	Inverted Data, FLAG Low
1	1	0	0	D0-D19	10 11	Data, FLAG High
1	1	0	1	$\overline{\text{D0-D19}}$	01 00	Inverted Data, FLAG High

X = Don't Care MT = Master Transition

The C-field bits were chosen so that a master transition always occurs between the second and third bits of the C-field. For data and control frames, this transition can be in either direction. The C-field bits were also chosen so that the codes for data and inverted data frames are complements of each other. The same is true for control frames. This allows the entire frame to be inverted with the correct C-field bits for a particular type of frame.

There are two types of fill frames, referred to in Table I as FF0 and FF1. FF0, a training sequence used during startup, has a single rising edge at the master transition and is a square wave with 50% duty cycle. The receiver's clock recovery circuit is able to lock onto this signal, extract the serial clock, and provide frame alignment. FF1, another training sequence used during startup, is also sent after startup whenever the user does not supply inputs for data or control frames. FF1 is similar to FF0 except that the position of the falling edge moves by one bit forward or backward, creating a square wave that is two bits heavy (FF1H) or two bits light (FF1L). The decision to send either FF1H or FF1L is made depending on the disparity* of previously transmitted bits, in an attempt to reduce the disparity to zero. Since FF0 is dc balanced and the two types of FF1 frames are sent to reduce disparity, fill frames are not inverted.

Noninverted control frames have the same C-field as fill frames, but are distinguished from fill frames by the center two bits of the D-field, which are 01. Control frames are inverted when appropriate, but then have a different, unique C-field.

All other possible C-field codes that are not listed in Table I are not allowed and are considered to be errors if received. The receiver detects the loss of a master transition or a forbidden C-field code as a frame error. This information is used by the receiver's state machine to derive the link status. In addition, if the flag bit is not used by the user, it is used for additional frame error checking. The flag bit is alternated internally by the transmitter and this alternation is checked at the receiver.

Coding Implementation

Fig. 7 shows a block diagram of the transmitter chip. The user supplies the parallel inputs D0-D19, a frame rate clock, the DAV and CAV inputs, and the FLAG input (optional). The high-speed and subrate clocks are derived from the frame rate clock by a phase-locked loop circuit. "System I/O"

* Disparity is the number of 1s minus the number of 0s.

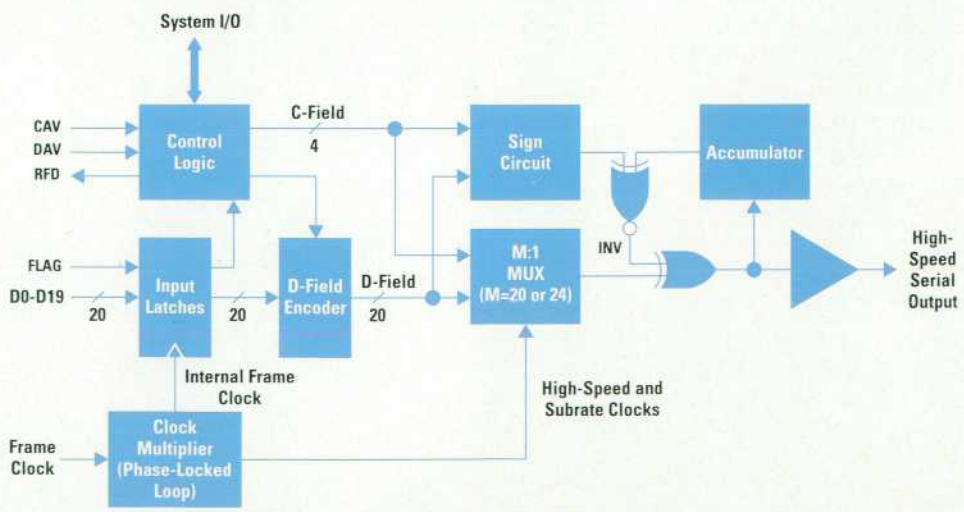


Fig. 7. Transmitter encoding circuitry.

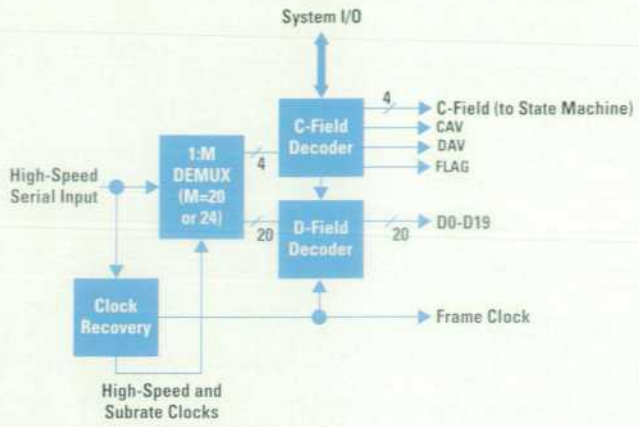


Fig. 8. Receiver decoding circuitry.

refers to other signals that are involved in the link's configuration and status. RFD (ready for data) is an output indicating to the user that the link is ready to transmit data. The D16-D19 inputs are ignored when the user selects 16-bit parallel word width.

Depending on the DAV, CAV, and FLAG inputs, the C-field coding bits are generated and any necessary encoding of the D-field is performed. Then the C-field and D-field bits are evaluated in a sign circuit whose output is the sign of the disparity of the frame. A separate accumulator keeps track of the disparity of previously transmitted bits. The decision to invert or not to invert a frame is made based on the outputs of these two circuits and is indicated by the signal INV. If the signs of the disparities of the current frame and the previously transmitted bits are the same, INV is high and the current frame is inverted. If they are not the same, INV is low and the frame is not inverted. Only data and control frames are inverted; the invert function is disabled for fill frames. The frame is serialized with a circuit that multiplexes the parallel inputs into a serial bit stream and performs any necessary frame inversion. The output of this circuit is then transmitted across the serial link.

A block diagram of the decoding portion of the receiver chip is shown in Fig. 8. After startup, the serial clock and the framing information are produced by the receiver's clock recovery circuitry, allowing the receiver to recover the serial data and demultiplex it back to parallel form. The frame clock is provided as an output for use in the user's system.

By examining the C-field bits, the C-field decoder determines what kind of frame has been received and whether or not it has been inverted. With this information, the D-field decoder restores the parallel data back to its original form. In addition, the C-field decoder provides DAV, CAV, and FLAG information back to the user. These signals have the same definitions as the corresponding transmitter inputs. The C-field bits are also used by the receiver's state machine to check for frame errors.

Encoding Circuitry

Encoding on the transmitter chip is performed mainly by logic cells and two on-chip programmable logic arrays (PLAs). However, there are two special parts of the frame inversion function. The first is an analog sign circuit which determines whether a frame has more high or low bits. The second is an accumulator which keeps track of the disparity of the previously transmitted data.

The sign circuit on the transmitter consists of one differential pair per bit, a summing circuit, and a comparator. To prevent errors in determining a frame's sign, it is important for the differential pairs to have matched current sources. Therefore, each differential pair is supplied by two current sources from an array of current sources laid out in common centroid fashion. This reduces the effects of process and temperature gradients on the value of each pair's combined current source. In addition, large-geometry resistors are used to improve matching of the current sources.

The currents are summed at shared collectors through resistors, creating a differential voltage proportional to the difference between the numbers of 1s and 0s in the frame. When there are more 1s than 0s, this voltage is positive; when there are more 0s than 1s, it is negative. This voltage then drives a comparator, which produces a high or low logic signal depending on the sign of the input voltage. This method of determining the sign of a frame is simpler and faster than a digital solution.

The accumulator circuit keeps track of the disparity of previously transmitted bits. It is implemented with a 6-bit up/down counter. To relieve timing constraints, the counter operates on two bits at a time. This allows it to operate at a clock rate that is half the serial output rate.

The counter can count from all 0s to all 1s and is reset at startup to the midpoint, which is considered a balanced state. The range of this 6-bit counter is then –32 to +31 bits, where 0 is the balanced state. With two input bits, there are four possible combinations: 11 which has a disparity of +2 bits, 00 which has a disparity of –2 bits and 01 or 10 which are balanced with zero disparity. Since we only need to count up or down by multiples of 2, we can allow one bit of the counter range to correspond to a disparity of 2 bits. Thus the effective counter range, in bits of disparity, becomes –64 bits to +62 bits. The worst-case disparity that can occur with this coding scheme is ±31 bits, which is well within the range of the counter. The most-significant bit of the counter is compared with the output of the sign circuit to decide whether to invert the frame.

Accumulating two bits at a time is the most convenient approach. If the counter were to operate on one bit at a time, it would still have to count either up or down and one bit of the counter range would correspond to one bit of disparity. Thus, the range of a 6-bit counter would be –32 to +31 bits of disparity, which would not have enough margin beyond the worst-case disparity of ±31 bits. A higher-order counter would be required, and it would also have to run at the full serial output rate, resulting in increased power consumption.

If the counter were to operate on four bits at a time, it would have the benefit of running at one fourth of the serial rate, but it would have to count up and down by 4, up and down by 2, or remain unchanged. One bit of the counter range could correspond to two bits of disparity as in the case implemented, but the counter design would be more complex.

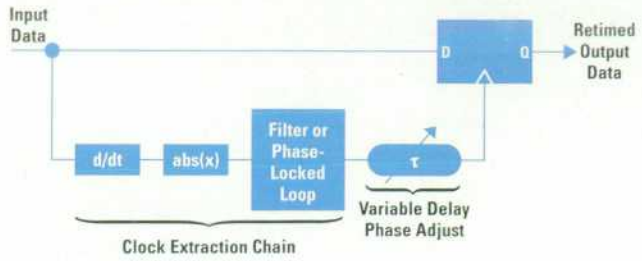


Fig. 9. Typical clock extraction and data retiming circuit requires phase adjustment and wide bandwidth.

Phase-Locked Loop

In a serial data link, the clock signal is not explicitly transmitted, but is instead implied by the transitions of the data stream. By examining the transitions in the data stream with a clock extraction circuit it is possible to create a replica of the original clock that was used to transmit the data. This recovered clock can then be used to sample and restore the potentially degraded analog input.

Many high-speed clock extraction techniques exist, but most have been developed for long-haul telephone applications. Telecom systems are designed to maximize the distance-bandwidth product of the link. This criterion minimizes both the number of physical repeater sites and the number of fibers that have to be installed in a given run. As a result, a much higher premium is placed on clock-extraction performance than on cost-effectiveness. These objectives have made this class of clock extraction techniques unsuitable for datacomm applications.

Traditional Telecom Clock Extraction Circuits

Fig. 9 shows a representative clock extraction and data retiming circuit that is used for high-bit-rate telecom systems. The incoming analog data stream is split into two parallel paths: the clock extraction chain and the data retiming path.

Because an NRZ (nonreturn to zero) data stream does not have a spectral component at the clock frequency, some nonlinear process must be used to derive a clock signal from the data stream. In the typical circuit of Fig. 9, a time derivative is applied, followed by an absolute value function. This combination of elements creates a narrow unidirectional pulse for every transition of the data. This new waveform contains a spectral component at the clock frequency. Once the clock component has been created, it can be isolated either by a filter, typically implemented with a SAW (surface acoustic wave) device, or by a phase-locked loop.

There are two problems with this configuration. The first is that, although the circuit extracts the correct clock frequency, it does not extract the correct phase. There is a large phase shift between the input data and the recovered clock. The phase relationship between the clock and the data must then be adjusted somehow to compensate for process and temperature variations. The second problem is that the creation of narrow pulses requires high circuit bandwidth. This is often the speed-limiting factor for gigabit-rate clock recovery circuits.

G-Link Solution

A design goal of the G-link chipset was to eliminate all external parts and user adjustments and effectively hide the system complexity from the user through monolithic integration. The clock extraction circuit was most impacted by these requirements. To achieve these aggressive goals, a new phase-locked loop circuit was developed based on a binary-quantized ("bang-bang") phase detector.

The phase-locked loop circuit used in the G-link chipset (see Fig. 10) works hand in hand with the CIMT line code to avoid both the phase adjustment problem and the bandwidth requirement of the traditional techniques. In this circuit, the incoming data splits into two paths (just as in the traditional telecom approach). Instead of a complex phase detector, which is potentially mismatched in delay to the retiming latch, two matched latches are used at the front end of the circuit. One latch is used for retiming and the other for phase detection. Because both latches are laid out identically on the chip, their delays are well-matched.

The two latches are driven by the VCO through a complementary buffer. If the VCO is properly aligned, the top latch samples the center of the data cell on rising edges of the clock while the lower latch samples the data transitions on the falling edge of the clock.

Because the G-link line code provides a guaranteed transition at a fixed, defined location in every frame, the sample of this transition can be used as an indication of the loop phase error. The VCO output is divided by either 20 or 24, depending on the selected word width, to produce one sampling pulse per frame. That clock pulse is used to take a sample in the vicinity of the master transition so that a phase update is generated, once per frame, indicating whether the VCO is early or late with respect to the master transition. Assuming a rising master transition, as shown in Fig. 11, if the VCO is too high in frequency, the sampling point drifts to the left of the master transition and a low value is sampled. If the VCO is too low, the sampling point moves to the right and a high value is sampled. This circuit then produces a one or zero indication from the phase detector that tells whether the VCO is early or late with respect to the incoming data.

Since the fastest operating element in this circuit is a latch operating at the serial rate, this circuit is usable up to the

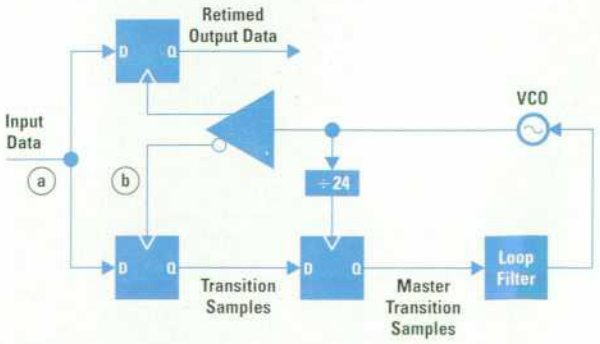


Fig. 10. Simplified diagram of the G-link binary-quantized (bang-bang) phase-locked loop and data retiming circuit.

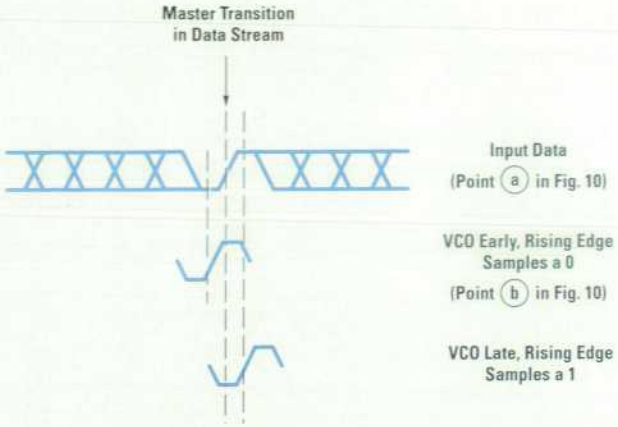


Fig. 11. Once per frame, the phase-locked loop detects whether the VCO is early or late with respect to the master transition encoded in each frame.

highest frequency at which a given process is capable of making a functioning latch. In addition, the circuit inherently provides excellent phase alignment between the VCO and the data. Note that the output of the phase detector latch is not linearly proportional to the loop phase error, but is instead a binary-quantized representation of the error. This characteristic renders the loop equations nonlinear and requires unconventional design methods (see "Bang-Bang Loop Analysis," page 110).

False Locking and Frame Synchronization

During initial link startup, it is necessary to ensure that the phase-locked loop correctly determines the frequency of the incoming data and finds the location of the master transition.

In many clock extraction circuits, the clock frequency is extracted from a coded, random data stream. A common difficulty with this approach is the problem of the phase-locked loop locking onto wrong frequencies that are harmonically related to the data rate. To avoid this problem, most systems limit the VCO range so that it can never be more than a few percent away from the correct frequency.

A narrow-band VCO using external components was not consistent with the goal of building a completely monolithic chipset. Integrated oscillators rely on low-tolerance IC components and are typically limited to ±30% tolerance on the center frequency. For customer flexibility, it was desired to extend the oscillator range to cover at least an octave. This range, in conjunction with digital dividers, allows the G-link chipset to operate over a range of 110 to 1400 Mbaud in four bands.

A second design problem is frame synchronization. At the receiver, some method must be employed to determine the boundaries between frames so that they can be properly deserialized back into the original parallel words. The G-link chipset establishes and monitors frame synchronization by using the embedded master transition. Unlike other links, the G-link chipset allows the continuous transmission of unbroken streams of data, without the insertion of special frame synchronization words.

Startup State Machine Controller

To eliminate the problems of false locking and frame synchronization, the G-link chipset uses a startup state machine and the special training fill frames.

Because the internal VCO is capable of operating over nearly a 3:1 range of frequencies, a frequency detector is necessary to avoid false locking problems. The frequency detector operates only when simple square-wave fill frames are being sent. A conventional sequential frequency detector, built of two resettable flip-flops, determines the sign of the frequency error. When the phase error is less than ±22.5 degrees, the output of the phase detector is used. Otherwise, the loop filter is driven by the frequency detector output. Because the frequency detection circuit cannot operate on data frames, the state machine controller must disable the frequency detection circuit before allowing data to be sent.

Neither node of a duplex link can achieve lock unless the opposite side is sending special fill frames. Neither side of the link can stop sending fill frames and start sending data unless the other side has successfully achieved lock. The state machine uses the two distinct fill frames FF0 and FF1 to allow one side of the link to notify the other side of its current locking status. This guarantees that fill frames will be sent whenever needed to restore lock, and only as long as necessary to achieve lock.

As described previously, FF0 is a 50% balanced square wave with equal numbers of 0 and 1 bits. FF1 consists of two modified square-wave patterns. These two patterns are used as needed to maintain dc balance on the link. Both FF0 and FF1 have a single, rising transition, which is in the same position in the frame as the master transition of data and control frames. The rising edge of the fill frames is used initially to establish an unambiguous frame reference. After initial lock, the master transition of the data frames is used to maintain frame lock.

Fig. 12 shows the state machine handshake procedure for a full-duplex link in greater detail. Both the near and far ends of the link independently follow the state diagram of Fig. 12. The three states are defined by the state variables STAT0 and STAT1. At power-up, each end of the link enters the sequence at the arc marked "Start."

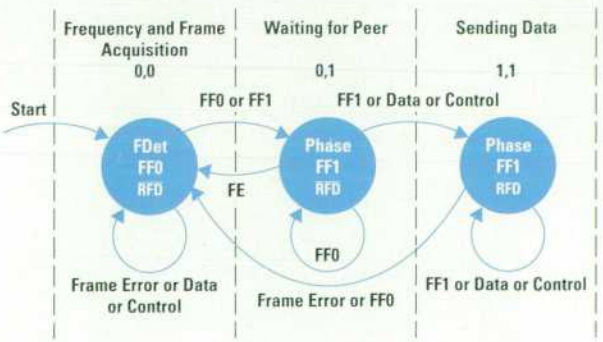


Fig. 12. State machine handshake procedure for a full-duplex link, showing the values of the state variables STAT0 and STAT1 (0,0, etc.).

Bang-Bang Loop Analysis

A simplified version of the clock recovery phase-locked loop of the G-link chipset is shown in Fig 1. Only the transition sampling latch is shown, and the input is assumed to be a square wave at the same frequency as the VCO.

The VCO is controlled through a loop filter that consists of the sum of an integral signal and a proportional signal. Because the phase detector is quantized, the VCO frequency switches between two discrete frequencies, causing the VCO to ramp up and down in phase, thereby tracking the incoming signal phase.

If the loop is properly designed, the system can be considered to be composed of two noninteracting loops. These are the paths labeled proportional branch and integral branch in Fig. 1. The first loop includes the connection of the phase detector to the VCO input through a proportional attenuator, while the second loop drives the VCO through an integrator.

The proportional signal tunes the VCO, causing the output of the phase detector to switch rapidly between 1s and 0s at a fairly high frequency. Other than the dc component, the bulk of the phase detector output signal spectrum falls outside the effective passband of the integrator branch of the loop. Thus the integrator branch operates on just the dc component of the phase detector output. Its job is to servo the center frequency of the VCO so that the two discrete VCO frequencies programmed by the proportional input will always bracket the frequency of the incoming data signal. This frequency adjustment occurs so slowly that it does not materially affect the operation of the high-frequency bang-bang portion of the loop.

Proportional Branch

To simplify the analysis of the first branch of the loop in Fig. 1, the integrator output can be replaced with a constant reference voltage so the proportional tuning input will cause the VCO to bracket the incoming frequency. The VCO will then run at two discrete frequencies: at a frequency slightly higher than the incoming data, thereby advancing the phase, or at a lower frequency, thereby retarding the phase.

If the incoming frequency is midway between these two discrete frequencies, the loop will switch between the two frequencies with approximately a 50% duty cycle. If the incoming frequency is slightly higher than the nominal VCO center frequency, the duty cycle will shift such that the loop will spend a higher percentage of time at the high frequency than at the low frequency. In general, it can be shown that the duty cycle present at the output of the phase detector is proportional to the difference in frequency between the incoming signal and the nominal VCO center frequency.

Integral Branch

The second branch of the loop contains the integrator. Because the integrator effectively filters out the oscillatory portion of the phase detector output and only reacts to the average value of the phase detector output stream, the proportional branch of the loop can be ignored here by replacing the phase detector with a virtual frequency detector. The integrator extracts the dc component and thereby tunes the center frequency of the VCO so that it is always equal to the incoming data rate.

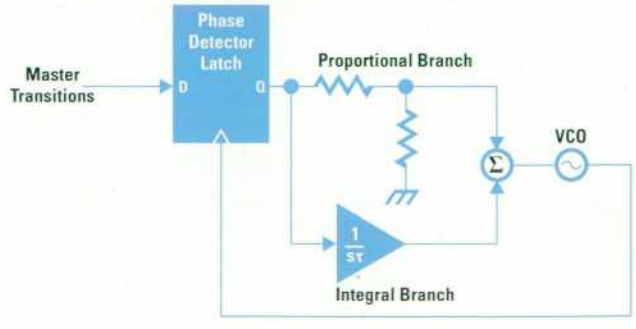


Fig. 1. Simplified version of the phase-locked loop. For analysis, the loop can be considered a combination of two noninteracting loops: a proportional branch and an integral branch.

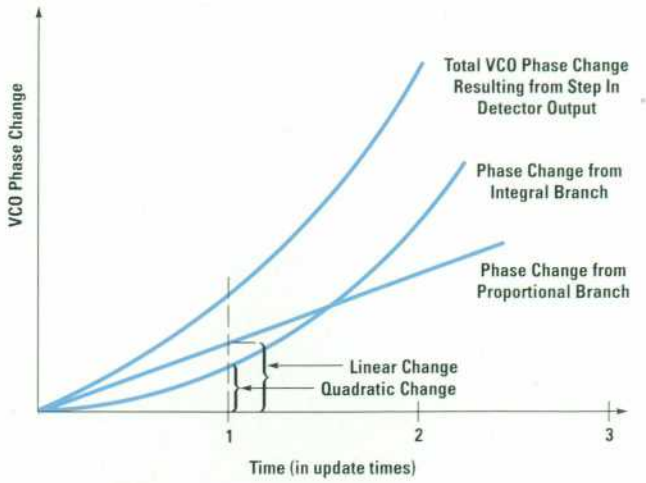


Fig. 2. Contributions to VCO phase changes. Stability factor is the linear phase change divided by the quadratic phase change in the same time.

In a conventional linear phase-locked loop, the loop error signal is proportional to phase error but is used to control the VCO frequency. This introduces an integration in the loop transfer function. This integration, in conjunction with the loop filter, creates a second-order feedback loop. Such loops can exhibit an underdamped response to changes in input phase, leading to an undesirable exponential buildup of jitter in systems with long cascades of repeaters.

In the G-link phase-locked loop, the phase-detector dc component is proportional to frequency rather than phase. Because the the frequency of the VCO is controlled by a frequency error signal rather than a phase error signal, no extra integration appears in the loop transfer function. This means that no jitter buildup results from the action of the integral branch of the loop. The jitter statistics are simply dominated by the hunting behavior of the high-frequency proportional branch of the loop.

Loop Stability

To reach a qualitative understanding of the loop behavior, the two branches of the loop were assumed to be noninteracting. For this assumption to be valid, certain conditions must be met.

It is important that the loop be set up so that, between phase samples, the action of the proportional branch of the loop dominates over the action of the integral branch. This can be verified by creating a step change from the phase detector and tracking its effect on both halves of the loop. Fig. 2 shows the contributions to the VCO phase change. In the proportional path, the VCO is programmed to make a small step change in frequency, which causes a linear ramp in the phase error. In the integral path, the integrator programs a linear ramp in VCO frequency, which causes a quadratic walk-off in the VCO phase.

The ratio of these effects at the end of one frame update time gives a figure of merit for the loop design. The phase change from the proportional branch of the loop must be greater than or equal to the phase change from the integral branch of the loop for the system to be stable. In the G-link design, this stability ratio is designed to be always greater than 10.

Richard C. Walker
Principal Project Engineer
Hewlett-Packard Laboratories

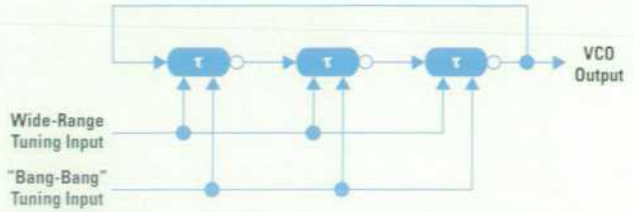


Fig. 13. The VCO consists of three variable-delay cells configured as a ring oscillator.

Each state in the state machine has three notations. The top notation is either "FDet" or "Phase." FDet stands for frequency detect mode, and implies that the frequency detector has been enabled in the receiver chip phase-locked loop. When the chip is in this mode, it is important that no data be sent, because the frequency detector is only able to lock onto one of the special training fill frames FF0 or FF1. The Phase notation means that the receiver phase-locked loop has been switched to phase-detect mode and is ready to allow data transmission. The middle notation in each state is the fill word that is currently being sent by the node's transmitter chip. The last notation is the ready-for-data (RFD) status of the transmitter chip. When RFD is low, the transmitter chip signals the user to hold off any incoming data while it is sending fill frames. When RFD is high, data is sent if available, and if not, fill frames are sent to maintain link synchronization.

The two bits bracketing the master transition are monitored by the receiver chip to detect a locked condition. If these two bits are not complementary for two or more consecutive frames, it is considered a frame error. The receiver chips at both ends of the link are able to detect data, control, FF0, and FF1 frames and frame errors. Transitions are made from each of the states based on the current status condition received by the receiver chip. Each of the arcs in Fig. 12 is labeled with the state that would cause a transition along that arc.

If either side of the full-duplex link detects a frame error, it notifies the other side by sending FF0. When either side receives FF0, it follows the state machine arcs and reinitiates the handshake process. The user is notified of this action by the deasserting of RFD.

This startup protocol ensures that no user data is sent until the link connectivity is fully established. The use of a handshake training sequence avoids the false lock problem inherent in phase-locked loop systems that attempt to lock onto random data with wide-range VCOs.

Loop Implementation

The VCO is built from three variable-delay cells configured as a ring oscillator (Fig. 13). The ring provides a wide-range tuning input and a small "bang-bang" tuning input. The wide-range input adjusts the delays of each stage from one gate delay to three gate delays, thus giving a 3:1 VCO frequency range. This wide range allows the final system to be specified with a 2:1 range over both process and temperature variations. The bang-bang tuning input programs a small change in the VCO frequency and is driven by the proportional branch of the loop filter.

The loop filter is implemented with a charge pump integrator and a 0.1-μF external capacitor, which is housed within the package. The integrator is based on a unity-gain positive feedback technique (Fig. 14) which cancels out the droop in the integrator filter capacitor. The effective dc gain of this circuit approaches infinity as the feedback gain approaches unity. The unity-gain technique achieves high dc gain while avoiding the stability and noise sensitivity problems of on-chip high-gain operational amplifier designs.

G-Link Chipset Implementation

To achieve the best speed and power performance, the G-link chips were designed using the HP B25000 25-GHz f_T silicon bipolar process. This process allows mixed-mode designs ranging from dense low-power logic structures to high-performance analog cells. A three-layer metal system allows compact layouts, minimizing chip area and cost. This process features transistors with minimum pitch of 2.6 μm. Only simple npn transistors and p+ and p– resistors were used in the design.

Building Block Design

The G-link chipset is a fully custom circuit using specially designed cells as building blocks. These include (1) logic cells consisting of gates, latches, and flip-flops, (2) PLAs for low-speed logic, and (3) I/O cells, which include all of the low-speed ECL and high-speed input and output drivers. A band-gap reference was also designed to stabilize chip performance with variations in temperature and power-supply voltage.

Logic Cells and Arrays. Since logic elements are used most widely in the G-link chipset, considerable effort went into optimizing their performance, power, and active area. A three-level tree structure was chosen to implement the logic functions. All signals are differential to improve noise margins and to reduce ground currents, which could disrupt the analog circuitry. The inputs and outputs of these gates and latches are fully level-compatible for ease of routing. Each functional cell has resistor options by which the speed can be traded off with power. In all, there are four power classes for each logic cell. An example of a master-slave flip-flop with a 2:1 input multiplexer is shown in Fig. 15. This circuit is designed to operate up to 2 Gbits/s at a junction temperature of 125°C with a fanout of 10.

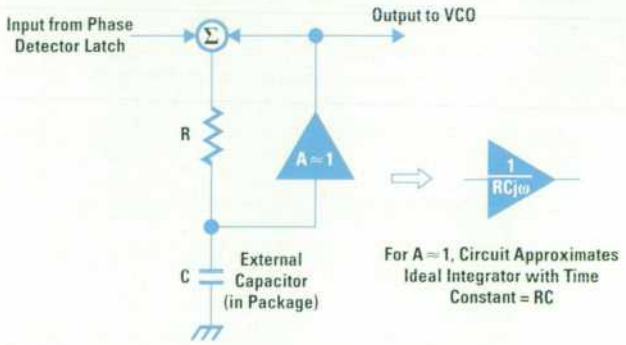


Fig. 14. The loop filter is implemented with a charge pump integrator based on a unity-gain feedback technique.

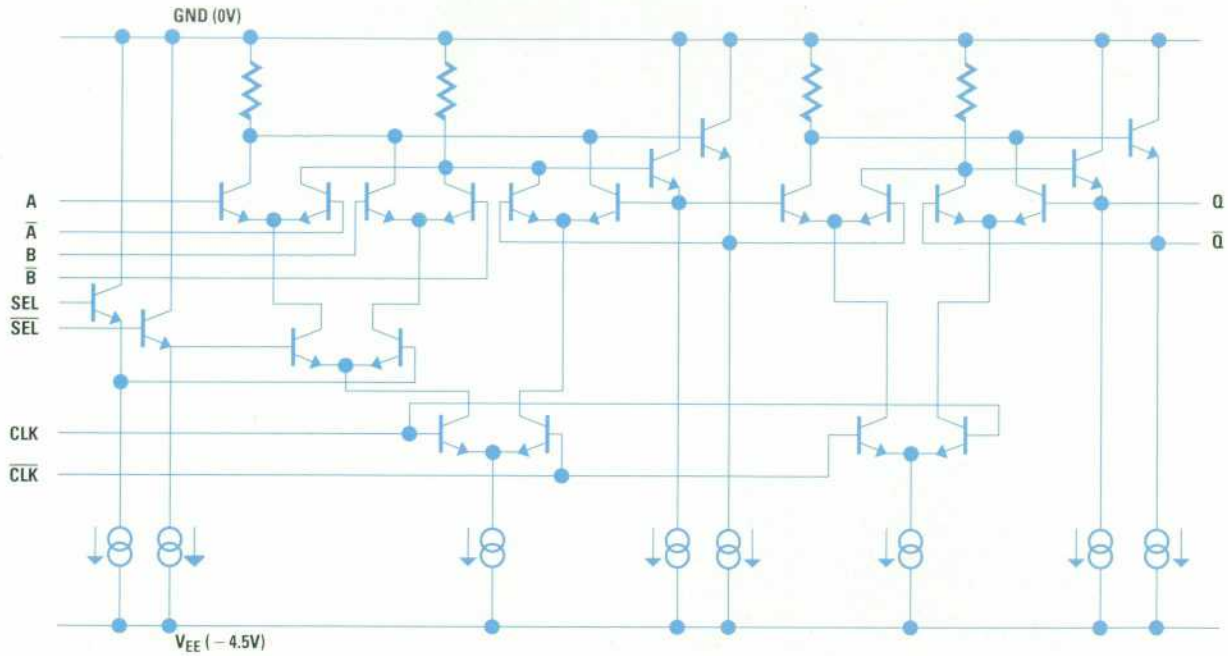


Fig. 15. Schematic diagram of the master-slave flip-flop with 2:1 input multiplexer.

Low-speed logic is implemented, where possible, with array structures for compactness and reduced power. The single-ended logic PLAs with AND-OR planes are designed to be programmed using only metal layers. Altogether, two PLA cells are used in the transmitter and one in the receiver.

Input/Output Cells. An effort was made in the I/O design to make the chips easy to use. Except for the high-speed serial signals, all of the chip I/O is 100K ECL-level-compatible. To minimize the power dissipation of the chip, ECL outputs are limited to driving 10 cm of transmission line with a minimum characteristic impedance of 50 ohms terminated into 300 ohms. For added convenience, unconnected inputs are internally biased to low ECL logic levels, and are sensed as high levels if the inputs are grounded.

A special input cell was designed for all gigabit-rate input signals. Both differential inputs of the cell are biased to ground with 50-ohm terminating resistors. This configuration allows singled-ended or differential input signals to be conveniently ac or dc coupled. This cell is used for the strobe and high-speed clock inputs of the transmitter and for the data and high-speed clock inputs of the receiver.

The G-link chipset is designed to work with either optical fiber or copper coaxial cable media. For cable applications, the data input cell of the receiver has an optional equalizer to extend the usable distance of the link. The equalizer circuit is designed with 3 dB of gain peaking at 600 MHz to compensate for signal roll-offs caused by the skin loss effect in coaxial copper cables. Operating at 1.2 Gbaud with RG-58 coax, the equalizer extends the usable cable length by over 50% for a given bit error rate.

All high-speed outputs are driven by buffered-line-logic cells. Buffered-line-logic drivers[4] provide differential outputs capable of delivering 0.7V into 50 ohms, ac or dc coupled to ground. If dc is coupled into −1.3V, the levels are ECL-compatible. In addition, the source impedance of the driver is matched to 50 ohms with a VSWR of less than 2:1. This makes the high-speed connections of the G-link chips very convenient and easy to use. The only requirement is that unused outputs be terminated into 50 ohms.

Band-Gap Reference. To minimize circuit drifts caused by environmental changes, a band-gap reference with power supply compensation was designed. This circuit provides a reference voltage that powers up all cells in both chips. A power-down feature in this circuit enables portions of the chips to be turned off to conserve power.

Layouts

To minimize the design and layout effort, a generic design structure was used as the basis for all cell layouts. Each of the various logic cells was built from the generic array of transistors and resistors by customizing the metal interconnections. The ratio of devices used to total devices available reached over 95% in this design. This layout technique has the advantage of easy reconfiguration for design revisions. The I/O port locations are uniformly defined for all cells to simplify cell interconnection.

An example of a master-slave flip-flop with a 2:1 multiplexer input is shown in Fig. 16. This circuit array, measuring just 104 by 135 μm, is customized with two layers of metal.

All cells and power buses are designed to be placed using a coarse grid. This simplifies the placement of cells in the system design level. Another feature is that all cells have test probe points accessible at the top metal such that all connection signals can be test probed for diagnostic purposes.

The transmitter and receiver chips each measure 3.5 mm on a side. The high-speed and low-speed pads for each chip are arranged so that a single package design accommodates both chips.

The design of the chips relied heavily on simulation and verification tools such as the Spice simulation program and HP's proprietary Bipolar-Chipbuster IC layout system. The Spice

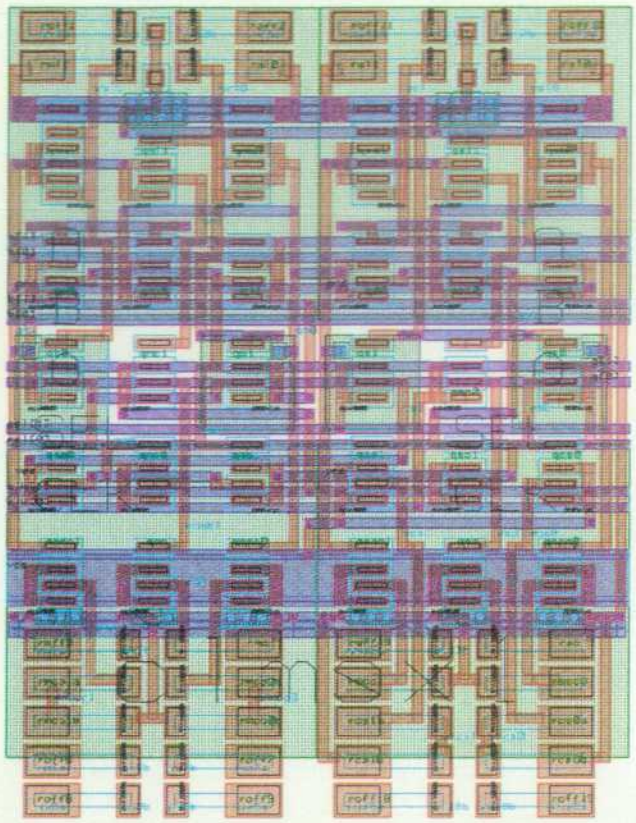


Fig. 16. Chip layout of the master-slave flip-flop with 2:1 input multiplexer.

circuit description files were extracted from the artwork including parasitic capacitors for final simulation before fabrication.

Packaging

A custom 68-pin ceramic quad flat package (CQFP) was designed specifically for the G-link chipset. It features 50-ohm transmission lines for the high-speed I/O pins and internal 0.1-μF capacitors for power supply bypassing and for the integrator of the phase-locked loop. It also has internal ground vias to minimize inductance, thereby reducing noise. Its outline conforms to standard 68-pin packages. The typical chip-to-case thermal resistance is under 14°C/W. Both the package and the chips are compatible with automatic assembly techniques for high-volume low-cost manufacturing.

After the chips and capacitors are mounted and the packages sealed on the lead frame, the units are placed onto plastic carriers for lead protection. A special test fixture was designed to test the final parts in this carrier at full speed.

Electrical Performance

The G-link chips' power dissipations are both under 2.5 watts worst-case. The 20%-to-80% rise and fall times of the high-speed data outputs are under 200 ps. The chipset is specified from 110 Mbaud to 1.4 Gbaud under all conditions. The lockup time of the phase-locked loop including frequency acquisition is less than 2 ms.

Features and Applications

The features and flexibility of the G-link chipset make it ideal for a wide variety of applications. These applications range from computer backplane links a few meters in length to wide area networks 10 kilometers long. The low cost and high integration level of the G-link chipset make it attractive for systems requiring serial transfer rates up to 1.4 Gbaud. It can serve as a generic virtual ribbon cable or can be used to build complete networks and peripheral channels. The G-link coding scheme has been accepted by the Serial-HIPPI (High-Performance Parallel Interface) Implementors' Group, and by SCI-FI (Scalable Coherent Interface-Fiber), an IEEE standard.

This section describes the features that allow the G-link chipset to be applied to this broad range of applications. It also describes a few specific applications, including generic data transport, networking standards, and simplex applications.

Ease of Use

Since most computing equipment both sends and receives data, the great majority of these applications are full-duplex. The state machine controller included on the chipset takes care of all the details of starting up such a duplex link. The designer needs to be concerned with only two signals: ready for data (RFD) and data available (DAV). RFD is the signal the state machine provides to indicate that the link is ready for data transmission. DAV is a signal the user controls to mark the availability of data. At the receiver, this signal is recovered and used to discern the beginning or end of data transmission.

Some applications generate data in bursts or as packets. Such bursty data is handled automatically by the chipset. When no data is available to transmit, the user simply deasserts the DAV line at the transmitter. The link will transmit FF1 as an idle code to maintain link lock and framing. At the receiver, a deasserted DAV signal indicates that data is not being received. At the start of the burst of data, the user asserts the DAV line at the transmitter. The data is transmitted across the link and marked as valid data at the receiver by the receiver's DAV signal. Thus the DAV signals can mark the beginning and end of packets while adding no burden to the system design.

More complicated packet headers can be created using the control available (CAV) signal. This signal works like the DAV signal, but instead of marking the data as valid data words, it marks the data as special control words. A system designer can use these to send packet header information, link or system control information, or anything that needs to be treated separately from data. At least 2^{14} control words are available, so they can be used to indicate a large number of packet addresses or special functions. Few communication links have such a rich selection of nondata words for control and signaling.

Flexibility

Flexibility was a major goal of the G-link design. To make this a high-volume, low-cost part, the chips were designed to meet the needs of as many different systems as possible. As

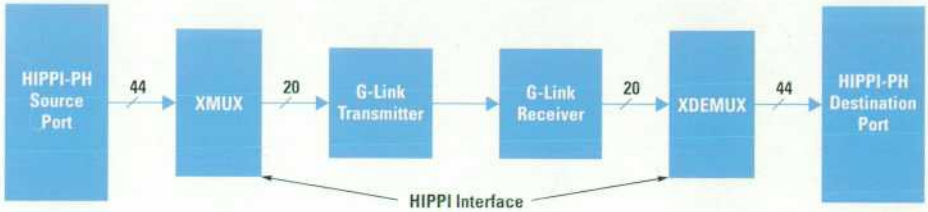


Fig. 17. Serial-HIPPI (High-Performance Parallel Interface) system implemented with the G-link chipset.

described earlier, the G-link line code can accommodate various word widths. This is very different from block codes such as 4B/5B and 8B/10B, which have fixed word widths. The G-link chipset readily accommodates data words of width 16, 17, 20, 21, 32, or 40 bits. The chipset has two fundamental word sizes: 16 or 20 bits. In addition, the flag bit is available. Therefore, 17-bit-wide words can be accommodated by selecting 16-bit frames and using the flag bit as a 17th bit. 21-bit words can be transmitted similarly. 32-bit words are supported by sending them as two 16-bit frames in a row. In this case the flag bit is used to distinguish the first 16-bit frame (e.g., flag = 0) from the second 16-bit frame (e.g., flag = 1). It is a simple matter to build the off-chip 32:16 multiplexers and 16:32 demultiplexers since the flag bit automatically keeps track of the necessary frame ordering. 40-bit-wide words are supported analogously. The transmitter chip accepts either a full-frame-rate clock or a half-frame-rate clock for multiplication up to the serial clock rate. In other words, for an 800-Mbit/s data rate and 16-bit words, the chip will accept a 50-MHz frame clock. When 32-bit words are transmitted it accepts a 25-MHz frame clock. This saves the system designer the trouble of doubling the word clock outside the chip.

The G-link chipset supports a wide range of serial transfer rates ranging from 110 Mbaud all the way up to 1.4 Gbaud. This wide range makes it attractive for many types of data. Because the chipset requires no off-package tuned elements or adjustments, it can be digitally switched between data rates. This is unlike other systems, which require tuned elements and precise adjustments and operate over very narrow ranges of frequencies. Switching between data rates aids testing and debugging. It can also be used to establish a standard physical layer that spans several operating frequencies.

Generic Data Transport and Proprietary Channels

The most prevalent application of the G-link chipset is generic data transport. In these applications, the chipset acts as a point-to-point unswitched bus extender, or virtual ribbon cable. A great advantage of the G-link chipset is that it automatically handles startup and framing. Once the link is operating, the user can send data continuously, without having to insert extra framing characters or form special packets. Other links typically require that special framing characters be periodically inserted into the data stream. For systems transmitting data continuously for long intervals, periodically inserting these special characters can be difficult and inefficient. Other link chipsets do not have a built-in hardware controller that signals when the link is operating improperly. Without these signals the system designer must depend on upper-level protocols, resulting in uncertain time delays.

In many applications, a point-to-point unswitched bus extender is sufficient. In these applications, the G-link chipset is all that is required and can form a complete communication link. The chipset can also be used in more complicated networks because it transports any data format across the link. Examples of standard data formats that can benefit from a point-to-point bus extender within private networks include SONET/SDH, Fiber Channel, and ATM data. SONET/SDH is a telecommunication standard that specifies data rates of 155 Mbits/s, 622 Mbits/s, 1.24 Gbits/s, and higher. Fiber Channel is an ANSI standard (X3T9.3) that covers a variety of data formats and rates. The IEEE 802.6 standard is an example of an ATM (asynchronous transfer mode) network.

The flexibility and ease of use of the G-link chipset enable it to fit a wide variety of applications. High-data-rate connections to disks and other peripherals are typical uses. These applications benefit from the very low overhead, simple operation, and high integration of the G-link chipset. For example the HP 27111A, introduced in 1988, is a fiber optic connection for disk arrays at 80 Mbits/s. With the tremendous increase in computing power and I/O rates in the last few years, the G-link chipset is well-suited for this type of application.

There is growing interest in using serial links for computer backplanes. Computer backplanes are typically jammed with hundreds of signals at data rates exceeding 100 Mwords/s. It can be difficult to control the skew on parallel data paths at high data rates. In addition, transmitting the data in parallel can require significant space. Serial links using optical fiber or coaxial cable may be the only way to transmit data without degradation by skew, loss, or reflections, while saving space.

Serial-HIPPI

In May 1991 the G-link chipset was accepted as the basis of the Serial-HIPPI standard. Serial-HIPPI is a specification for an 800-Mbit/s serial data link that has been agreed upon by over 40 vendors and users. Serial-HIPPI transmits data between HIPPI-PH nodes, up to 25 meters in coaxial cable, or 10 km with optical fiber. HIPPI-PH is an ANSI standard (X3.183-1991) for transmitting digital data in parallel between data processing equipment nodes. It is prevalent in supercomputing and high-end workstation environments. Fig. 17 shows a diagram of a complete Serial-HIPPI system using the G-link chipset. HIPPI-PH data consists of 44-bit-wide words at 25 Mwords/s. This data includes 32 data bits, 4 parity bits, 7 control bits, and the clock. Ahead of the G-link transmitter there is an additional circuit called the XMUX. This circuit reduces the data from 44 bits to 40 bits by replacing two control signals with the chipset's RFD, replacing the HIPPI-PH clock with the clock derived from the incoming serial data, and encoding three of the other control signals into two lines. The XMUX then multiplexes the data 40:20. This data is transmitted with the G-link chipset as 40-bit

words, in the manner described previously. At the receiver, the XDEMUX demultiplexes the recovered data from the chipset 20:40, then restores the additional four signals.

Serial-HIPPI is intended to provide compatibility at the serial optical and serial electrical interfaces. Therefore, the G-link startup sequence and line code are part of the specification. Hewlett-Packard is supplying the G-link chipset. The XMUX and XDEMUX are commercially available.

SCI-FI

The G-link chipset was also chosen as the transport mechanism for the SCI-FI (Scalable Coherent Interface-Fiber) standard (IEEE P1596). SCI is intended to replace traditional backplane buses, which are limited in physical length and in the number of elements that can be connected to the bus, and are limited to only one transaction on the bus at a given time. SCI solves these problems using point-to-point communication between nodes. For short distances, data can be sent in parallel on metallic conductors. For longer distances, better noise performance, and smaller physical size, optical fiber or coaxial cable is used to transmit the data serially. Data is transmitted as 17-bit words (16 bits plus flag) at 1.25 Gbaud. No additional circuitry is required assuming that the data is already in the P18 form specified in the standard (parallel data, 16 bits plus flag and clock).

Simplex Applications

The applications discussed up to now have all been full-duplex; there is a forward data path and there is an equivalent reverse data path. This covers the great majority of applications. However, there are some applications in which a particular piece of equipment is primarily a data source or sink but not both.

These applications can be divided into two categories. In the first category are systems where data primarily flows in one direction, but there is a lower-speed method of communicating in the reverse direction. An example of this kind of system is a video terminal, which might receive a tremendous amount of video data and send back only the very low-rate typed commands of a user. Fig. 18 shows the G-link chipset connected for this type of system. A transmitter at the data source is connected with a single fiber to a receiver at the destination. The low-speed return path is used to pass only the STAT1 indication from the state machine controller at the receiver back to the source. This signal makes a transition only after power-on, or if lock is lost for some reason. It indicates to the transmitter when the receiver is locked and ready for data, or if the receiver needs to be sent FF1 to become locked. Since this signal changes state infrequently

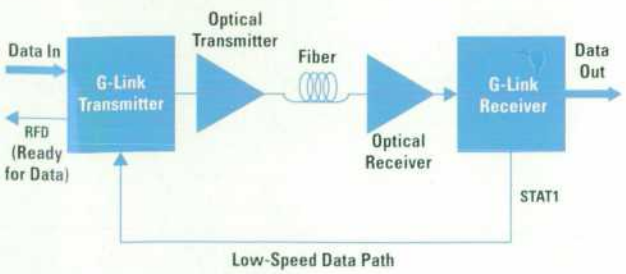


Fig. 18. Simplex system with low-speed return path (e.g., a video terminal).

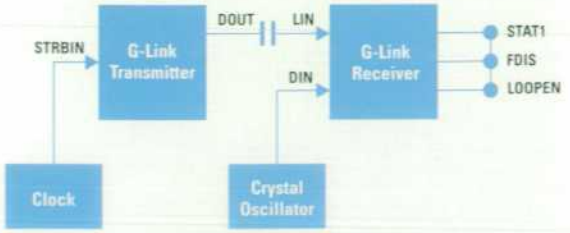


Fig. 19. In a simplex system with no return path, such as a video distribution system, the receiver input is automatically switched (by the STAT1 signal) between the external crystal oscillator for frequency lock and the input data for phase lock until the master transition is acquired. STROBIN is the transmitter data clock input. DOUT is the normal transmitter serial data output. LIN is the receiver loopback serial data input. DIN is the normal receiver serial data input. STAT1 is the receiver state machine status output. FDIS is the receiver frequency detector disable input. LOOPEN is the receiver loopback control.

and its timing is not critical, it can be sent over any available return path, possibly over a single dedicated metallic wire, or mixed in with other low-speed data on an RS-232 connection or telephone line.

There are a few communication systems that are strictly simplex and have no return path at all. These systems are inherently problematic for ensuring data integrity because there is no way of knowing the status at the receiving end of the link. A typical example of this type of network is video distribution. Special techniques should be applied when the G-link chipset is used for these types of applications.

A practical and inexpensive solution is shown in Fig. 19. Three receiver pins are connected together. This solution takes advantage of the G-link receiver's state machine to monitor and switch data paths depending on the lock conditions. The only additional component required is an inexpensive crystal oscillator operating at the frame rate, which is used to frequency lock the receiver VCO. After frequency lock, the state machine automatically switches the receiver to the data input (LIN) and phase lock takes place. If the receiver does not lock onto the master transition, internal error checking will cause the state machine to reset and switch the receiver back to the crystal reference. The small frequency differences between the transmit and receive oscillators will provide a phase shift that will allow the receiver to lock onto the master transition correctly. Typical lock times will be on the order of a few milliseconds, which can be improved if necessary by pulling the crystal reference slightly off frequency. Lock times under 200 μs are achievable by adding phase modulation or programmed delay in the crystal oscillator path.

Summary

The G-link chipset is a highly integrated, compact, silicon chipset with features that enable it to serve a number of application areas. It performs its own startup and framing. This allows a user to transmit data continuously, without inserting extra characters, in a virtual ribbon cable mode. The chipset includes data available and control word signals which allow the creation of simple packets. The chipset accepts a wide range of input word widths, allowing a good match to a variety of computer buses. The wide range of serial data rates makes it an ideal transport vehicle for

125-Mbit/s FDDI data to 1.24-Gbit/s SONET data. The chipset can work in simplex systems, allowing its use for distributing video. Two widely accepted networking standards, Serial-HIPPI and SCI-FI, are tailored to the operation of the G-link chipset. The production volume made possible by this broad range of applications should make possible truly low-cost gigabit-rate data links.

Acknowledgments

We would like to thank Tom Hornak for his many contributions to this project, and his guidance throughout its development. Thanks to J.T. Wu for his contributions to the receiver chip design and layout. Kent Springer, Rasmus Nordby, Craig Corsetto, and Doug Crandall all had an early influence on the project. Hans Wiggers, David Cunningham, Steve Methley, David Sears, and Richard Dugan have been responsible for the G-link's success in the standards arena. David Yoo, Jean Norman, Natalia McAfee, and Lie Lian-Mueller have all helped with the assembly and packaging of the chipset. Special thanks to Don Pettengill, Shang-Yi Chiang, and the HP B25000 process team, without whose excellent work and cooperation this project would not have succeeded.

References

1. D. Crandall, et al, *DC-Free Line Code for Arbitrary Data Transmission*, U.S. Patent no. 5,022,051, June 4, 1991.
2. R.O. Carter, "Low-Disparity Binary Coding System," *Electronic Letters*, Vol. 1, no. 3, May 1965, pp. 67-68.
3. C. Corsetto, et al, *Phase-Locked Loop for Clock Extraction in Gigabit Rate Data Communication Links*, U.S. Patent no. 4,926,447, May 15, 1990.
4. B. Lai, "A 3.5 Gb/s Fully Retimed Decision Circuit with Temperature Compensation," *Proceedings of the 1988 Design Technology Conference*, Hewlett-Packard, May 1988, pp. 296-303.

HEWLETT-PACKARD
JOURNAL

October 1992 Volume 43 • Number 5

Technical Information from the Laboratories of Hewlett-Packard Company

Hewlett-Packard Company, P.O. Box 51827
Palo Alto, California, 94303-0724 U.S.A.

Yokogawa-Hewlett-Packard Ltd., Suginami-Ku Tokyo 168 Japan

HEWLETT PACKARD

5091-5273E