MARCH 1983

HEWLETT-PACKARD JOURNAL

HEWLETT·PACKARD JOURNAL

Technical Information from the Laboratories of Hewlett-Packard Company

Contents:

MARCH 1983 Volume 34 • Number 3

In this Issue:

The electronic bench is the universal development system that a designer or a team of designers uses to develop hardware and software for any kind of electronic product at all. At each designer's command are all of the computer-aided design tools and analyzers that might conceivably be needed in the development effort. All of these subsystems are linked so that their combined power far exceeds that of individual tools used by isolated designers. As you've probably guessed, this single do-everything system doesn't exist. However, the subject of this month's issue is a system that aspires to become the electronic bench for designers of products based on commercial microprocessors, those little computers on a chip that are turning up in everything from scales to automobiles these days. First featured in our October 1980 issue, the HP 64000 Logic Development System is much closer to its goal today than it was then.

How close is it? Let's look at what it takes to develop a microprocessor-based product. First, the hardware must be designed and a prototype built. Perhaps it's a microwave oven, and somewhere inside is a printed circuit board and on it a microprocessor in a socket. Concurrent with the hardware development is the software development effort. Someone must write the program the microprocessor will execute. In the final product this program will be stored permanently in a read-only memory chip (a ROM), at which point it's no longer software but firmware. With prototype hardware and software available, the testing and debugging begin. Hardware and software are tested separately, then together until the entire system is working properly.

The 64000 today is a pretty complete system for software development and for testing and debugging software and the digital portion of the hardware. It can't design printed circuit boards or tell you if the oven does justice to a casserole. The 64000 plugs into the socket on a prototype board that's normally occupied by the microprocessor, and it emulates or pretends to be the microprocessor. The product doesn't know the difference, but the designer can now control and observe everything that happens. Today's 64000 System can emulate and produce software for a long list of widely used microprocessors, including the newer, more powerful 16-bit models. For testing and analysis, the system offers a new software state analyzer and a new hardware timing analyzer. A new portable development station and flexible disc drives make the system useful for servicing complex equipment and for development at remote sites (cover photo). A development station, portable or benchtop, can be a laboratory's total development system, performing all functions, or it can be dedicated to a single function such as software development. Up to six stations can be linked in a cluster, so several designers can share a common data base. It's a formidable array of capabilities, even if it doesn't add up to the ultimate electronic bench.

-R. P. Dolan

Editor, Richard P. Dolan • Associate Editor, Kenneth A. Shaw • Art Director, Photographer, Arvid A. Danielson • Illustrators, Nancy S. Vanderbloom, Susan E. Wright • Administrative Services, Typography, Anne S. LoPresti, Susan E. Wright • European Production Supervisor, Henk Van Lammeren

© Hewlett-Packard Company 1983 Printed in U.S.A.

© Copr. 1949-1998 Hewlett-Packard Co.

Extensive Logic Development and Support Capability in One Convenient System

HP's 64000 Logic Development System gets closer to the concept of an "electronic bench." Real-time emulation, configuration flexibility, and integrated analysis functions are some features of this latest version of the 64000 System.

by Michael W. Davis, John A. Scharrer, and Robert G. Wickliff, Jr.

DURING THE DESIGN of a microprocessor-based system, a large percentage of the time, typically 30%, is spent on debugging, integrating, and optimizing hardware and software. This phase often generates design changes that must be implemented quickly for maximum productivity. Not infrequently, software development must proceed with a prototype hardware system, or the hardware design must proceed with only skeleton software. The system integration phase is the first time that all parts of the microprocessor system are brought together. This ongoing process of refinement and change can be further complicated by the use of multiple processors within the same system, and these processors may be from more than one vendor.

The development tools used by designers must offer flexibility, power, and ease of use. They must be appropriate for

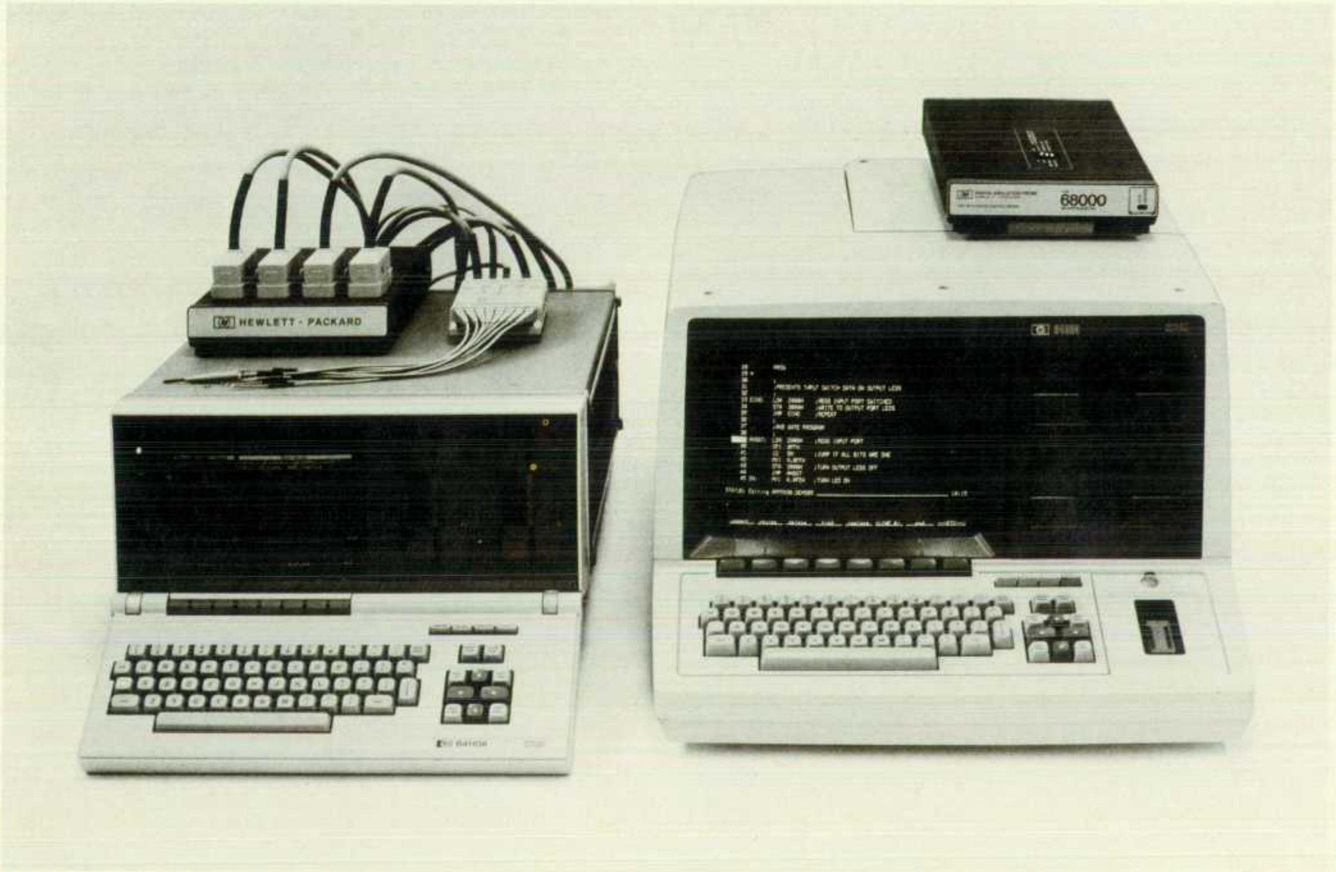


Fig. 1. *The HP 64100A Development Station (right) includes a keyboard, display, host processor, space for 10 option cards, dual flexible disc drives, and power supply. The optional emulator for the 68000 microprocessor is shown on top of the 64100A. The HP 64110A Development Station (left) is the transportable version of the 64100A and has the same features except that the number of option cards is limited to five. Some of the cables and a pod for the HP 64620S Logic State/Software Analyzer option are shown with the 64110A.*

© Copr. 1949-1998 Hewlett-Packard Co.

projects ranging from a single-person task to a large-team software task with a huge data base. The HP 64000 Logic Development System provides a comprehensive solution for these varying design requirements. Facilities for hardware timing analysis, state/software analysis, and software performance overview provide debugging and integrating tools that improve the designers' efficiency. These analysis tools can be used independently or in conjunction with the 64000's emulation and software development features. As a result of recent architectural enhancements, the 64000 System can be used in different development environments where the user may work independently, with a team, or at a station connected to a larger CPU. Offering all of these features and settings while retaining a common human interface, the 64000 is a highly integrated logic development system with many of the attributes of the "electronic bench."[1]

System Configurations

An HP 64000 Logic Development System may be a single station configured as an HP 64100 Development Station with an HP 64941A Dual Flexible Disc Drive installed, or as an HP 64110A Development Station. Either station (Fig. 1) can edit, assemble, compile, link, and store program modules. A compatible HP printer can be added for hard copy, and a compatible HP hard disc memory can be added for greater storage capacity and higher performance. The system can be expanded to a cluster of as many as six development stations, each with its own host processor, sharing a hard disc and line printer. The stand-alone mode can be used for smaller software projects or analysis and emulation. The multistation cluster has the advantage of a shared data base and shared peripherals for a team of designers. A station having the flexible disc drives can be disconnected and used independently at any time. Software is compatible between the hard disc and the dual flexible disc drives.

In the stand-alone configuration, a development station can be connected to an HP-IB (IEEE 488) controller and used as a typical controlled instrument. In either the stand-alone or cluster configuration (Fig. 2), any development station can be connected by an RS-232-C/V.24 interface to a host CPU such as an HP 3000 Computer. A communications protocol and terminal emulation software permit uploading and downloading of both source and absolute files between the 64000 and a host computer (see box on page 6). This gives the 64000 the flexibility to use software tools available on the host computer, or to use its own built-in software tools, reserving the host for archiving and management control. Also, the host computer can send a command file to a 64000 Station and cause it to execute those commands.

Each 64100A and 64110A Development Station can be configured in many different ways. Adding an HP 64032A Memory Expander with 32K words of RAM provides additional symbol space for compilers. Both Pascal and C cross-compilers are available for a number of microprocessors, and a host Pascal compiler is available to execute on the 64000 System. The addition of emulator options with up to one megabyte of independent memory in 32K, 64K, or 128K-byte increments gives the user an executing and debugging environment and a tool for integrating

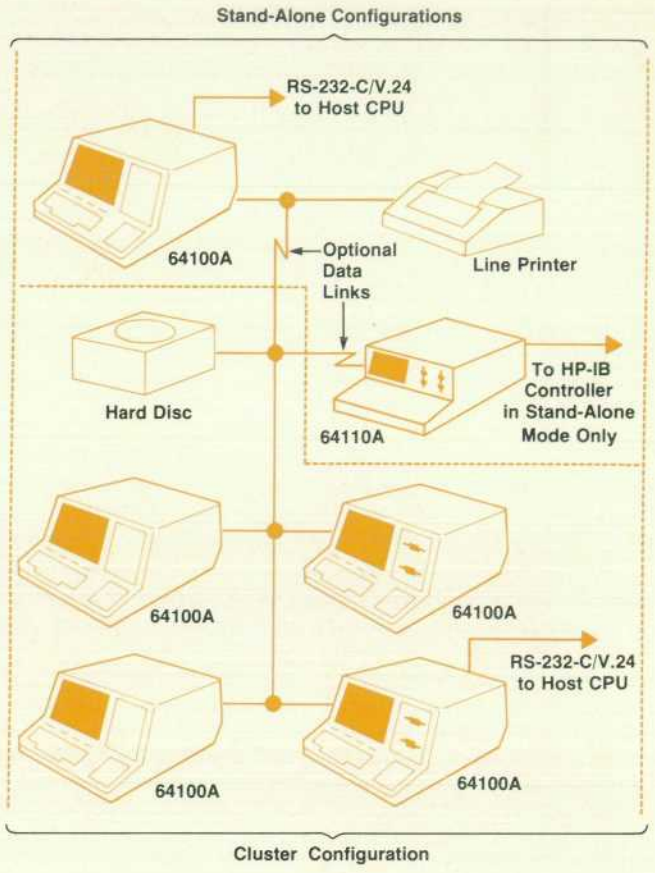

Stand-Alone Configurations
RS-232-C/V.24 to Host CPU
64100A
Optional Data Links
Line Printer
Hard Disc
64110A
To HP-IB Controller in Stand-Alone Mode Only
64100A
64100A
RS-232-C/V.24 to Host CPU
64100A
64100A
Cluster Configuration

Fig. 2. *The 64000 Logic Development System can be configured with any 64100A or 64110A Development Station in a stand-alone mode with or without a listen-only line printer, and can be connected to an HP-IB controller, if desired. The cluster configuration can be expanded to as many as six stations connected to a hard disc memory. Any station, either in a cluster or a stand-alone system, can be connected to a host CPU via an RS-232-C/V.24 interface. Stand-alone stations can also interact with each other or a cluster configuration by using optional data links.*

hardware and software in the early phases of development. As software modules are completed, they can be mapped into the target system's RAM or stored in programmable read-only memories (PROMs) using the HP 64500A PROM Programming System. The HP 64302A Logic Analyzer is a single-option-card state analyzer which can be added to any emulator to monitor address, data, and status of the target microprocessor system. For complex debugging and integration, a user can add an HP 64620S Logic State/Software Analyzer (see article on page 16) with external or internal probes. The 64620S is expandable from 20 to 120 channels and has a real-time overview of state events for software performance evaluation. It also has access to the 64000 data base for symbolic debugging. For hardware debugging and integration, an HP 64600S Logic Timing Analyzer (see article on page 23) can be used for the monitoring of control, status, and logic levels. It is available with 8 or 16 channels. Both the 64620S and the 64600S can be installed as separate subsystems or in conjunction with other HP analyzers and emulators.

© Copr. 1949-1998 Hewlett-Packard Co.

Development Stations

Both the 64100A and 64110A Development Stations have an integral keyboard and display (Fig. 1) to provide the interface between the operator and the logic development system. Both have the same host processor system, although because of available space, it is partitioned differently in the two stations. The host processor is a custom 16-bit microprocessor manufactured by HP. The smaller station, the 64110A, is rack mountable and transportable, easily moved about the laboratory or used for production and service applications. The larger station, the 64100A, is better suited for fixed benchtop applications.

Each station has an option card cage (Fig. 3) to house circuitry for the various system options. The 64110A has five option card slots and a 250W power supply and the 64100A has ten option slots and a 400W power supply. The development station bus is the interface between the host processor and the option cards. Each option card is identified by the host processor when the station is turned on, communicating through 16K words of memory-mapped I/O so that the option software and directed-syntax softkeys are self-configuring. Station setup and the HP-IB are controlled by rear-panel switches.

All option cards use the same directed-syntax human interface used for the system monitor, editor, and software tools.[2] This results in ease of use, quick learning, and better user productivity. The emulation system uses a separate emulation bus to communicate between emulation control, emulation memory, and analysis cards. The analysis cards also share an intermodule bus for measurement control and interaction. The measurement system software can support any four emulator or analyzer subsystems within a single development station. This software treats the HP 64302A Logic Analyzer as an integral part of an emulator. The number of subsystems a station can hold is dependent on the number of option cards involved since some subsystems require more than one card.

Flexible Disc Drives

Dual 5¼-inch flexible disc drives, another development station option, make it possible to operate the 64000 System without a hard disc memory. Since the software runs on either a hard-disc-based cluster or flexible-disc-based stations without change, these new stations are not simply an add-on to the 64000 product family. They can be used in a cluster system, and when a problem arises in a field application, the software needed to check out the remote system can be recorded on flexible discs directly from the shared cluster disc. This includes user-developed programs as well as HP system software. Thus, the station and the flexible discs can be taken to the problem. These new station options also lower the entry cost of a 64000 System. The minimum configuration is reduced from a station and a hard disc to just a station, with the assurance that upgrading to a hard disc memory will not cause any disruptions. Operation of a flexible-disc-based station is identical to the operation of a station within a cluster. All files transport from one environment to the other without change. The file manager is the same in both environments; only the disc driver code is different. Flexible disc interfaces to the file

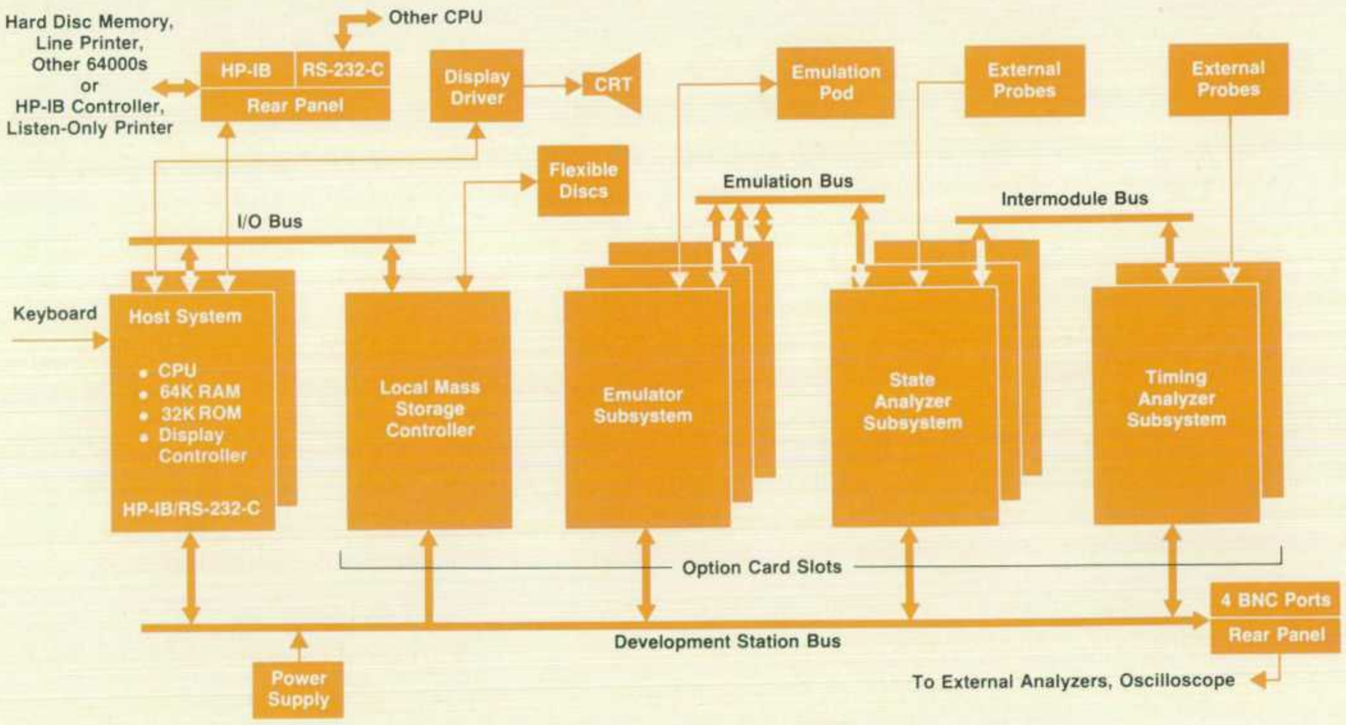


Fig. 3. *The host system and the emulated microprocessors have independent buses and can run simultaneously. Emulation and analysis can be controlled for coordinated measurements, allowing software development concurrent with emulation and analysis. The assortment of nine option cards shown above is possible only in the 64100A Station since the 64110A Station is limited to no more than five cards.*

© Copr. 1949-1998 Hewlett-Packard Co.

HP 64000 Terminal Software

by Paul D. Bame

The need for rudimentary communication between the HP 64000 Logic Development System and other devices was recognized at the inception of the system. To handle this need, an RS-232-C/V.24 port was designed into the development station, and a simple copy command was implemented in the system monitor. When used with the RS-232-C/V.24 port, the copy command allows the user to transfer files between the 64000 System and a remote device. All 64000 file types can be transferred. During transfers of text files (source and listing), the upper bit of each byte is stripped. For other file types, all eight bits of each byte are transferred. The maximum transfer rate is 9600 baud, with pacing required for rates greater than 1200 baud. To achieve this pacing, an XON/XOFF protocol is used. Error detection is provided on nontext files; however, no error correction or retransmission capabilities exist.

Recently, terminal software has been added to the system monitor. It allows a development station to be used as a conversational terminal with file transfer capabilities. The requirement was for the development station to be able to replace, but not to emulate many common terminals on popular mainframes. The development station should be able to plug into a normal terminal port, allowing the simplest interface between a user's mainframe and the 64000 System.

Because of system considerations, it was desirable to implement only asynchronous communication capabilities. With this decision made, the next step was to determine the most general way to perform the data pacing. This pacing, or flow control, is very important for reading data rates above 1200 baud reliably. Many mainframes cannot support a sustained input data rate of even 300 baud, especially when supporting a heavy timesharing load. The two most frequently used protocols were determined to be XON/XOFF and ENQ/ACK, and so these two were implemented.

Protocols

The XON/XOFF protocol is a start-stop protocol. When a device receives an XON character, transmission starts, and when a device receives an XOFF character, transmission halts. ENQ/ACK, on the other hand, is an interrogative protocol. When a device wishes to transmit, it sends an ENQ character. If the receiving device is ready to accept input, it responds with an ACK character.

Because in the past most terminals could never send data fast enough to cause problems, terminal drivers were not always written symmetrically. When a mainframe computer sends a file to a 64000 Station, the protocol running on the terminal driver prevents the 64000 Station from being overrun. However, some drivers do not respond correctly to protocol when the mainframe is the receiver. For example, many terminal drivers do not send XON or XOFF, or do not respond properly when they receive an ENQ. Although HP computers' ENQ/ACK protocol is not sufficient to control fast terminals like the 64000, this issue has been addressed. HP terminals often include tape cartridge units, which, if left uncontrolled, could cause the mainframe to be overrun. An additional level of protocol was added to control the tape units. A special character is sent to the terminal (tape unit) whenever the mainframe is able to accept the next record. This protocol was adopted in the 64000's terminal software to prevent a 64000 Station from overrunning a remote device or mainframe.

Data Transfer

Because of the internal characteristics of a 64000 Station, additional requirements are placed upon the protocols that are especially important when operating at the higher baud rates. The major challenge was coordinating the disc I/O with the RS-232-C/V.24 I/O during file transfers. During disc data transfers, the interrupt system may be unavailable for up to 6 ms. The universal asynchronous receiver-transmitter (UART), which is also interrupt driven, can buffer up to two characters internally before being serviced. This creates a situation where, if more than two characters are received during the time that interrupts are disabled, some data may be lost. Allowing for other factors that also may affect this, and leaving a comfortable safety margin, the 64000's terminal mode can run at up to 1200 baud with no protocol. Using a protocol, the terminal software can run at up to 9600 baud.

The protocol must ensure that no more than two characters are received during a disc transfer. This is not a problem while using the ENQ/ACK protocol. It is sufficient to ensure that all disc transfers occur after receiving an ENQ character and before replying with an ACK. With the XON/XOFF protocol, this is not so easy. In theory, when a XOFF character is sent, the disc transfer can be made and then a XON can be sent. In practice, there is no guarantee that the remote device will send no more than two characters after the 64000 Station sends the XOFF character. Many mainframes simply cannot stop transmission that quickly.

There are two solutions to this problem. First, a simple program described in the 64000's terminal software manual can be run on the mainframe, solving the problem at the mainframe. Second, a recent enhancement to the terminal software provides a configurable delay, forcing the 64000 to wait from 0 to 32,767 ms after sending an XOFF character before going to the disc.

The protocols available in the 64000's terminal software system are general enough so that even if a terminal driver is not compatible with either protocol, fairly simple mainframe programs can be written to bypass the terminal driver and interface with the 64000 System directly. To avoid interference with the mainframe terminal driver protocol characters, the controlling characters (XON, XOFF, ENQ, and ACK) are user-configurable to any seven-bit ASCII characters. This allows programs to be written without knowledge of how the terminal drivers work. The 64000's terminal software supports transfers of source (text) files and absolute (object code) files. Absolute files are transferred in one of three hexadecimal formats: Motorola S1/S9 format, Intel format, and Tektronix format. Most mainframe-based cross software produces one of these formats, so object code can be transferred to a 64000 System where the 64000's emulation tools can be used.

The 64000's terminal software has two major limitations. In addition to not supporting synchronous communication, it only permits transfer of source files and absolute files.

Paul D. Bame

Paul Bame is a software development engineer at HP's Colorado Springs facility. He received a BEE degree in 1981 from the University of Delaware and is a member of the IEEE. He lives in Colorado Springs, Colorado and his hobbies include photography and camping.

© Copr. 1949-1998 Hewlett-Packard Co.

manager are exactly the same as for the hard disc. The only assets a user of a flexible disc system gives up are data access speed and storage space.

In cluster operation, the flexible discs are used only for storing and retrieving user files. New system software associated with the new hardware retains the commands familiar to users of the earlier version of the 64000 System. Commands used to store and retrieve files have the same syntax as those used for these functions on stations containing DC-100 cartridge tape drives. Since there are two flexible disc drives in a station, but only one tape drive in an older tape-equipped station, the new software uses the first available flexible disc drive and continues on the second drive if necessary. In this way, older software can take full advantage of the new hardware without requiring command syntax changes or requiring the user to keep track of drive numbers.

Since flexible discs must be formatted before use, a software module was created to perform this function. (System modules consist of segments that are overlayed in 64K bytes of RAM within each station. Only one segment may be executing at a given time.[2]) Commands for testing media and duplicating and comparing entire discs are also in this software module, which is placed in a separate segment with its own command interface so that once this module is loaded, many operations can be performed without further reference to a system disc.

For a user to take advantage of the fact that all system software runs in both cluster and stand-alone modes, some means of moving system files has to be provided. Further, because of space limitations on flexible discs, only the software needed for a particular application is placed on flexible discs, leaving more disc space for user files on-line.

Single applications, such as the state analyzer, require many files for segment overlays. Keeping track of these files is a complex task. To free the user of this burden, a special file was created to group files under a single application name. The flexible disc system generator module references this file, showing the user which system modules are on each hard disc or flexible disc. Moving modules from disc to disc does not require knowledge of the module file structure.

HP-IB Available in Stand-Alone Mode

In a cluster, the rear-panel HP-IB port is used for system operation, but when a station is operating in the stand-alone mode, the HP-IB port is available for remote operation of the system. The 64000 Stations are not controllers; they function within an HP-IB system much like other instruments. The difference is that the 64000 is a system rather than a single instrument. Remote operation is therefore handled by passing across the HP-IB commands that would be typed under normal system operation. This effectively separates the HP-IB module from the measurement and control functions within the system. Command interpretation and error checking are handled by the receiving module just as if the command came from the keyboard. This also makes programming easy, since the same commands are used for HP-IB and manual operation. In addition, commands entered manually with the aid of the softkeys[2] can be logged to a file, and this file can then be used as a command file activated by a bus command, or the file can be sent over the bus for retransmission later.

The HP-IB interface can be programmed to request service from a controller on the occurrence of any of three conditions: awaiting command, error in last command, and measurement complete. This allows a controller to proceed with other tasks and detect errors until the 64000 Station is ready. A status byte can be requested to determine which condition caused the service request. Reading this byte also clears the interface to request service on the next occurrence of a programmed condition.

Two special commands give the HP-IB controller direct access to the display and the station beep signal. This allows remotely processed data to be displayed, as well as messages to an operator in a programmed system. Thus, the 64000 Station can be used within a complex automated test system in a manufacturing area or, with adapters, controlled at a remote site over a phone line.

Logic Analysis Subsystem

The problem of realizing a logic analysis and software development system that meets all of the various needs of the digital hardware/software design scenario can best be appreciated by looking at a model of the design process. As shown in Fig. 4, the design process spans a range of activities from developing analog and timing characteristics of devices (in a system environment) to high-level and abstract observations of system performance.

At the bottom and middle of the funnel shown in Fig. 4, designers are most concerned with "does it work?" At the top, the concern is "can it be made to work more efficiently (less time, less code)?" Some of the measurements applied to these issues are indicated along the right side of the funnel. As one might expect, there are significant differences in the instruments that address these various applica-

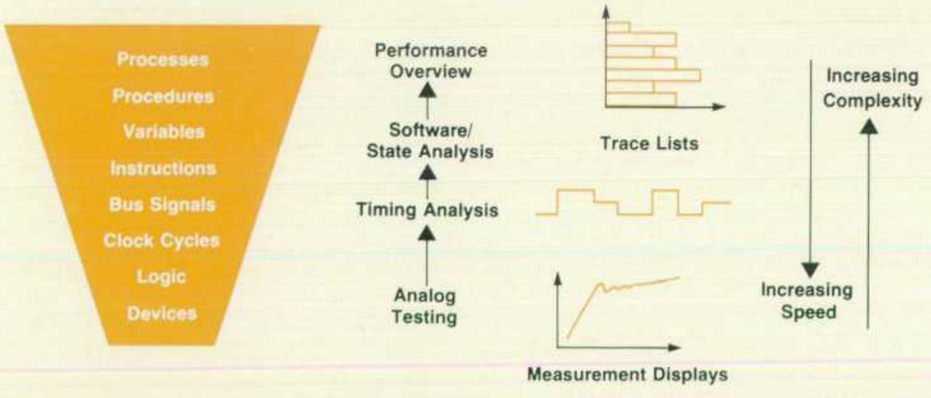


Fig. 4. *The digital measurement spectrum for microprocessor-based systems.*

© Copr. 1949-1998 Hewlett-Packard Co.

The HP 64000 Measurement System

by Kipper K. Fulghum

One of the major contributions introduced by the HP 64000 Logic Development System is the friendly integration of software and hardware development tools with software and hardware analysis tools. Specifically, microprocessor emulation allows the designer to exercise software and hardware in the target system, while internal analysis provides unobtrusive testing and debug facilities. With the introduction of the 64620S Logic State Analyzer and the 64600S Logic Timing Analyzer, the 64000 System now has extensive external analysis capabilities as well. By themselves, the state, timing, and existing emulation subsystems are powerful development and analysis tools. Intermodule communication between these subsystems provides the user with a state-of-the-art digital measurement system.

Intermodule Bus

Intermodule communication is accomplished via a high-speed, ECL low-true intermodule bus (IMB) consisting of five signal lines: master enable, delay clock, trigger enable, storage enable, and trigger (high-true). All signal lines use one driver, except the trigger line, which allows multiple drivers. Any number of receivers are allowed on all five lines. With this set of signals, complete intermodule control, sequencing, triggering, and store qualifying are provided. The capabilities of each analyzer are enhanced by the other subsystems on the IMB, and multibus and multiprocessor analysis and emulation can be done with one system.

Measurement System

Management of this multiple module analysis/emulation system is achieved with a software package labeled meas_sys on the 64000's softkeys. This measurement system is responsible for initiating, controlling, monitoring, and concluding any measurement session in the 64000 environment. It allows multiple analysis/emulation modules (up to four) to coexist in a single station and communicate with one another over the IMB.

The measurement system functions as the resource manager of the IMB. It enforces the global rules of the bus, coordinates its use, and prevents competition for the bus resources. The specific functions any single module can perform on the bus are determined by the nature of the module. Each module (state, timing, or emulation subsystem) defines what signals it will drive with what internal resources, as well as what it will receive.

The only evidence of the measurement system software visible to the user, once analysis/emulation has been requested, is the multiple module monitor. This monitor is entered only if there is more than one analysis/emulation module in the station. In all cases, the configuration and resident portions of the measurement system software are loaded from disc. If there is only one module present, that module is loaded and entered immediately by the measurement system software. But if multiple modules are identified during configuration, the measurement system monitor is entered instead. This monitor provides the means of loading software for any of the multiple modules. It displays all of the modules in the box, the card-cage slot number of each module's control board, each module's current status, and a description of each module. If any IMB specifications have been made, the current IMB configuration, including drivers/receivers of every IMB line, is displayed. Lines with possible competition, controllers of the rear-panel BNC ports, and emulators on the emulation ghost start line (a software-supported line for initiating multiple emulation) are also reported. The softkey labels, determined dynamically at configuration time, include one for each module in the box, with two or more identical modules differentiated by their control board's slot number. Depending on the current run status and the current IMB specification, an execute or halt softkey label may also be displayed. A softkey allowing output of the display to a printer is also shown. Once a module has been selected, entered, configured, and exited, the measurement system monitor is reentered, allowing selection of the next module to be configured. At each reentry, the display is updated with the modified IMB configuration, and the softkeys are labeled appropriately.

Slot Array Utilities

The measurement system maintains three major data structures. Each data structure has its own set of utilities for creating and interrogating the structure for pertinent information. The first structure, created by a card-cage poll during initial configuration of the measurement system, is called the slot array, an array of records indexed by slot number. Each record contains information about the board in a particular slot. The slot array utilities allow all analysis/emulation modules and the measurement system to access this information, which includes the board's select code, type, and module name. If the board is a control board, two other items are also maintained: an assigned module number and the RAM address of that module's relocated baby module.

A baby module is a small (maximum of 512 bytes), relocatable hardware-dependent module capable of basic identification, initialization, and control of its respective hardware set. This baby module, one per analyzer/emulator, is loaded by the measurement system and relocated to a more convenient RAM location during configuration. It is used by the measurement system and its parent module to start, monitor, and halt related hardware. The combined capabilities of all relocated baby modules allow the measurement system to control all the hardware sets in the station without having any of the parent analysis/emulation software packages resident in memory. This is a necessary condition since only one parent module can remain in memory at any one time, and none is available when the measurement system monitor is loaded.

Function Array Utilities

The second major data structure supported by the measurement system software is the function array. It maintains the current configuration of the IMB (all drivers and receivers of every IMB line), the rear-panel BNC ports, and the multiple emulation ghost line. When a module requests permission to drive or receive a particular signal through the function array utilities, this structure is updated to reflect the request and the caller is informed of any possible conflict with another module the request may have created. Only minimal data is kept in this structure. If more information about a certain module is required, the slot array utilities can be accessed to provide the necessary data. Given this cross-referencing of structures, each module can not only find out what bus conflicts there are, but also who is contending for the bus. This is important since a measurement cannot be allowed until all conflicts are resolved.

Run-Time Stack

The final crucial data structure used by the measurement system software is the run-time stack. This stack is a typical LIFO stack: last-in, first-out. It keeps track of the required starting order

© Copr. 1949-1998 Hewlett-Packard Co.

of the modules for a multiple module execution. A distinct starting order of the modules is imperative because drivers and receivers of certain lower-level signals should be delayed until the module that sets the context of the measurement, that is, the driver of master enable, is up and running. Also, emulation modules should start after analysis modules; otherwise, the analyzers may miss possible sequence or trigger conditions. Therefore, items are pushed onto the stack as follows: all emulators on the emulation ghost line, the driver of master enable, all drivers of the other signals, and finally all receivers. Thus, when the items are popped off the stack, the last analysis module popped is the driver of master enable, then all emulators, ensuring that the correct measurement context is set before the measurement begins.

Upon exiting the measurement system, all data structures and current baby modules are saved in one configuration file on the cluster disc or local flexible disc, depending on which is being used as the system disc. This file, unique to each station in a multistation system, contains all necessary information required by the measurement system software to reinitiate a measurement session. Thus, if all individual modules are exited cleanly, that is, if individual configurations are completed, the user can end a measurement session and return to the 64000's main system monitor to edit, compile, or perform other functions. Later, a return to the previous measurement session can be made via a continue request, and all measurement configurations and run status will be unchanged. This provides the user with an extremely friendly path from analysis to development tools and back again without having to reconfigure the instrument.

Kipper K. Fulghum

Kip Fulghum came to HP in 1979 after completing the work for a BS degree in computer science at Colorado State University. He worked on the flexible disc operating system for the 64110A Station and the IMB software for the 64000 System. Kip was born in Oklahoma City, Oklahoma and now lives in Colorado Springs, Colorado. He is single, and is interested in skiing, soccer, backpacking, and automobiles.

tion areas. For a microprocessor-based system, the stimulus for these measurements is the emulation capability of the logic development system. Despite these different needs, only one system is designed, and sometimes a single designer must use most, or all of the disciplines shown.

Performance and State Analysis

The performance portion of the 64620S Logic State/Software Analyzer subsystem is optimized to accumulate range data on address events and time events, in real time, in a large storage memory for postmeasurement processing. This gives the analyzer an overview capability that measures the performance of the system software by indicating the relative time spent doing tasks or the times spent doing a specific task. The data is displayed in histogram, graph, or list form. The input circuitry for the performance analyzer is identical to that of the state analyzer system. The performance analyzer has its own 4K-byte storage memory. A large memory is necessary since large amounts of data are required to give a meaningful overview picture. The performance analyzer and state analyzer share the same board set, but act as independent analyzers interrelated by data qualification and trigger mechanisms.

The state analyzer is optimized for qualifying measurements (trigger) and data (store qualify). Its multiple sequence detectors are invaluable in untangling the complex algorithms characteristic of software design. To achieve these capabilities, the state analyzer has many decision points within every clock period of the system under test. This decision-making time limits the maximum incoming clock rate that the analyzer can accept, but through the use of emitter-coupled logic and custom bipolar logic chips, the maximum clock speed of the 64620S Analyzer exceeds the needs of most processors. In a state analyzer, and in particular the 64620S, incoming data is highly qualified, and therefore having a large memory for storing states is not critical. Also, in state analysis, sampling is done by the clock of the system under test and the important parameters are the setup and hold times of data in relation to that clock. In general, if the system is to be sampled reliably, the setup time should be a minor portion of a clock cycle and the hold time should be zero or negative. This is accomplished in the 64635A Data Probes or the 64650A Preprocessor by a custom bipolar delay generator.

Timing Analyzer

Timing analysis has quite a different set of requirements. Incoming data is sampled by an internal asynchronous clock and all incoming data is sampled. Therefore, a large storage memory and an effective postmeasurement display system are important. Also, in timing, data is observed for timing relationships and race conditions, and therefore the timing resolution should exceed the minimum timing margins required by the system under test. The resulting constraint is that sample rate and input line skew are the important parameters for timing analyzer inputs. Setup time and hold time are not relevant in the timing analyzer, other than that their sum is usually an indicator of channel time skew. The delay lines present in a state analyzer would actually increase skew and deteriorate resolution if identical inputs were used for both state and timing.

Parametric voltage information is more important at this end of the design continuum, and the 64600S Timing Analyzer can capture and display three levels of voltage information. In this dual-threshold mode, a low, middle, or high logic level can be displayed. The two thresholds are usually set to a particular logic family's low and high input specifications (see Fig. 3 on page 26).

Effective triggering is also important in a timing analyzer. The 64600S can do parametric triggering such as on event times, event transitions, and glitches. Transitions are the dynamic entry to or departure from a specified pattern. A glitch on a data line is two or more transitions that occur between internal sample clocks. Using the dual-threshold measurement capability of the 64600S Timing Analyzer as an example, a typical trigger specification might read trigger

© Copr. 1949-1998 Hewlett-Packard Co.

on greater than 50 usec of CLOCK=Middle. Since the timing analyzer does not capture data synchronously with the clock of the system under test, it cannot do an effective job of triggering on synchronous data. It must rely on a state analyzer for this type of trigger capability.

State, Timing, and Software Development Together

As we have seen, the requirements for hardware and instrument characteristics differ quite a bit between state and timing analyzers. Thus, trying to make the same module do both tasks is not practical. The 64000 approach is to design independent modules, each optimized for a specific set of instrumentation tasks. However, because the state, timing, and emulation modules are controlling and measuring the same system, connections and synergism must exist among these modules.

There are three primary areas of interaction. The first is the real-time interaction of emulation, state analysis, and timing analysis. Since, in general, any analyzer takes only a relatively small snapshot of a system's performance, defining windows (specified events in time or address space) and indexing across module boundaries is a necessity.[3] In the 64000 System, this is done by the high-speed intermodule bus (IMB) which allows interaction of triggering, trigger arming, storing, store arming, system starting, and windowing of the functions of one module by another module.

The second area of module interaction is data base sharing between the software development system and the state analyzer. The symbols generated by the linker, assembler, and compiler are available to the state analyzer for the purpose of displaying symbols for addresses and also as addresses in operands. They are also used in setting up format and trigger specifications by appearing as softkey labels when appropriate.

The third area of interaction is related to the operator and is concerned with the commonality of instrument setup and syntax. The directed-syntax softkeys and grammar conventions of the 64000 System provided an excellent opportunity to achieve a setup and display synergism between the state and timing analyzers and emulation modules that allows commonality in operating all three types of instruments. A new software module has been added to the operating system to coordinate the interaction and start-up of the individual modules. This module also indicates the status of the modules during execution (see box on page 8).

Acknowledgments

The addition of analysis, 16-bit emulation, a new mainframe, dual flexible discs, and new operating configurations for the 64000 Logic Development System was accomplished by a number of people, most of whom are authors of other articles of this issue. One person who deserves special mention is Chuck House, now Corporate Engineering Manager. His encouragement, patience, and vision helped make this project a reality.

References

1. C.A. House, "Viewpoints—Chuck House on the Electronic Bench," Hewlett-Packard Journal, Vol. 31, no. 10, October 1980.
2. T.A. Saponas and Brian W. Kerr, "Logic Development System Accelerates Microcomputer System Design," Hewlett-Packard Journal, Vol. 31, no. 10, October 1980.
3. J.A. Scharrer, R.G. Wickliff, Jr., and W.D. Martin, "Interactive Logic State and Timing Analyses for Tracking Down Problems in Digital Systems," Hewlett-Packard Journal, Vol. 29, no. 6, February 1978.

John A. Scharrer

John Scharrer has been with HP since 1965. His contributions include various designs for the 1200 Series and 1707A Oscilloscopes, and plug-ins for the Series 180 Oscilloscopes. He was project leader for the 1615A Logic Analyzer and now is project manager for the logic analyzer modules used in the 64000 System. He has written two papers related to his work, one for the HP Journal. Born in Sheboygan, Wisconsin, John attended the University of Wisconsin and received a BSEE degree in 1963 and an MSEE degree in 1965. He is married, has three daughters, and lives in Manitou Springs, Colorado. Outside of work, he serves on several church committees and is interested in skiing, camping, and tennis.

Robert G. Wickliff, Jr.

Bob Wickliff is a native of Columbus, Ohio, and attended Ohio State University, earning a BSEE degree in 1969 and an MSEE degree in 1970. He joined HP in 1973 after doing research on radar backscatter at Ohio State University for four years. At HP, Bob has worked on the storage tube for the 1741A Oscilloscope and the firmware for the 1615A Analyzer. He was the group leader for the analysis software in the 64000 System and now is working on new analysis products. He is the author of one paper and co-author of four others related to his work (three of these papers have appeared in the HP Journal). He is married, lives in Colorado Springs, Colorado, and is interested in photography, travel, and camping.

Michael W. Davis

Mike Davis joined HP in 1970 with a year of experience in vehicle electronics and control systems. He has held several positions in production and manufacturing engineering and now is the lab section manager for emulation and analysis products used in the 64000 System. Mike is a member of the IEEE and on the board of trustees for his church. He received a BEE degree in 1967 from General Motors Institute and an MSEE degree in 1969 from the University of Colorado. Born in Brazil, Indiana, he now lives in Chipita Park, Colorado. He is married, has one son, and enjoys basketball, skiing, mountaineering, hiking, ice skating, and weight lifting. His primary interest is working on antique cars (he owns a 1935 Packard and is a member of the Veteran Motor Car Club of America).

© Copr. 1949-1998 Hewlett-Packard Co.

Mainframe Design for an Integrated Engineering Workstation

by Jeffrey H. Smith, Carlton E. Glitzke, and Alan J. DeVilbiss

THE NEW HP 64110A and upgraded HP 64100A Development Stations are the second-generation mainframes for Hewlett-Packard's 64000 Logic Development System. They allow the system to be used in many different ways, ranging from cluster to stand-alone applications. Rather than creating completely new stations, the task was to design a smaller, portable mainframe, the 64110A, while retaining absolute compatibility with the existing 64100A mainframe. At the same time, the 64100A was upgraded with a new power supply to handle new option cards. Both mainframes also received a dual flexible disc drive system that is compatible with the existing base of 64000 System hardware and software.

The larger of the two new mainframes, the 64100A (see Fig. 1 on page 3), is an evolutionary upgrade of the 64000 System introduced in 1979.[1] It has room for installation of the new dual flexible disc system. A more powerful, standard 400-watt power supply provides regulated power for its host system and for up to ten 64000 Series option cards and associated probes.

The smaller of the new stations is the transportable 64110A Development Station (see cover and Fig. 1). The mechanical design goals of the 64110A were to provide a transportable and self-contained mainframe compatible with the 64000 option cards. For an instrument to be transportable it must be compact, durable and easy to carry and move around. Some features of the 64110A are:

- A 9-inch-diagonal CRT, a full ASCII keyboard with softkeys and cursor control keys, and two flexible disc drives contained in the front of a 7-inch-high, standard HP System II frame
- A new thicker, softer, more comfortable, side handle (compatible with any 20-inch-long HP System II cabinet)
- A pivoting, locking keyboard for front-panel protection and compactness
- Injected-molded exterior parts of polycarbonate (strong, durable, UV stable and do not require painting)
- Adaptability to existing HP carts and folding airline luggage dollies
- Exposed edges and corners contoured where they might come in contact with the person carrying the instrument
- Feet on both sides, rear and bottom so the instrument can be set down or stored in any logical position
- Accepts any of the 64000 System options except the PROM programmer, which drops only into an opening on the right side of the larger 64100A Station's keyboard, and the earlier tape cartridge drive, which is not needed because dual flexible disc drives are standard for the 64110A
- Space for five option cards and a 250W power supply to power the mainframe and the cards.

The 64110A Development Station is operable in a number of environments. Normally the 64110A is set on a bench on its bottom feet or tilted on the front tilt bail. Tilting on the bail positions the keyboard for typing. If no table or bench is conveniently available, the 64110A has legs which pull out for stable floor standing operation. The 64110A also may be rack mounted using standard rack hardware.

The 64110A's keyboard adjusts to any angle and locks with the flip of a lever. Should the instrument fall or excessive force be applied to the locking mechanism while the keyboard is latched, the keyboard will slip and pivot without breaking.

An optional top-mounted pouch accommodates all cables, probes, emulators, and other accessories, making the 64110A completely self-contained. All probe cables come from the option cards directly into the pouch and then to the

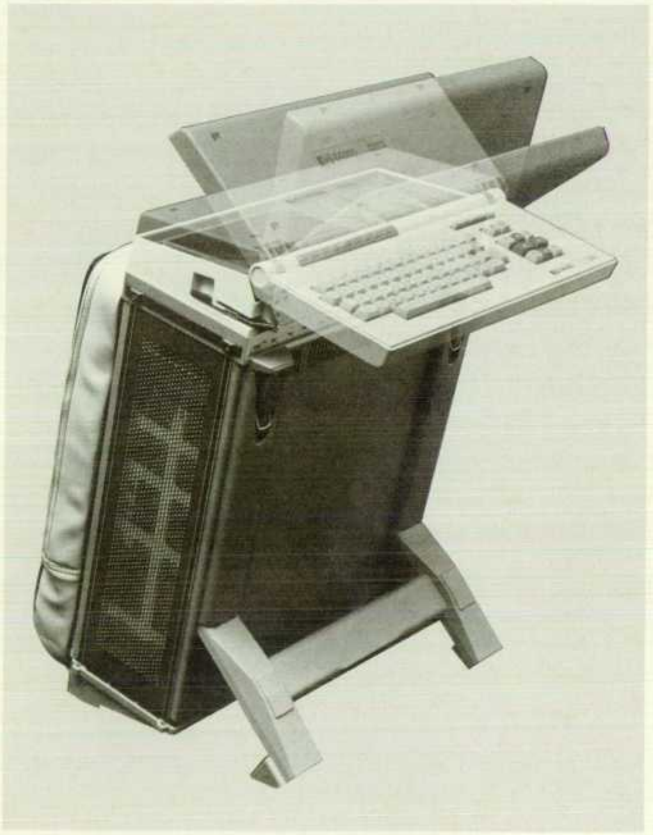


Fig. 1. *The 64110A Development Station is designed for easy transportability and can be hand carried or moved on a typical luggage dolly. The 64110A can be set on a bench, rack mounted, or set on the floor steadied by its rear feet as shown. The keyboard can be adjusted to a convenient work position and an optional pouch can be mounted on top of the 64110A to carry cables, connectors, and pods.*

© Copr. 1949-1998 Hewlett-Packard Co.

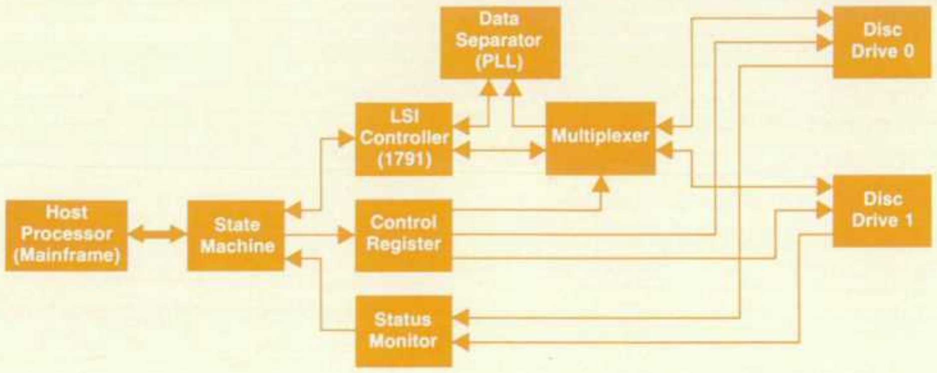


Fig. 2. *Simplified block diagram of the flexible disc drive system.*

system under test. Therefore, the probes may be disconnected from the system under test and stored in the pouch without disconnecting them from the 64110A.

Flexible Disc Drives

Both stations use the new 5¼-inch dual flexible disc drive system for backup and local mass storage. Compared to the earlier DC-100 tape cartridge system, this system reduces the average time required to back up a 20K-byte file from 51 seconds to 24 seconds. A 20K-byte file can be overlayed in RAM in 2.6 seconds. The use of two drives increases the on-line local mass storage to 540K bytes and makes it easier to duplicate discs for backup. Each flexible disc allocates 1½ tracks for a directory, 62½ tracks for data, 4 tracks for operating system storage, and 2 tracks as spares, which are used in case any bad tracks are found during formatting. The block diagram of the flexible disc system is shown in Fig. 2.

Control of seek, read and write operations, and the conversion between an 8-bit parallel format and the serial data stream used to store data on the flexible disc are handled by a 1791 integrated circuit. Two additional registers are used to control the drive motors, select the active drive and side, and monitor drive status.

Each drive is connected by its own bus to the controller, keeping each drive selected at all times. This permits the controller to monitor its status continuously. As an example of a status check, the controller is signaled whenever the user changes the disc in a drive; otherwise, it would be necessary to read the directory to determine if the proper disc is in place before each disc access. Because each drive contains a write-protect switch that rides against the jacket of the disc as it is inserted or removed from the drive, the status monitor can detect a closed-to-open transition of this switch and set an internal MEDIA-CHANGE bit. The MEDIA-CHANGE bit is also used to recover from some error conditions. A READY status signal is generated by retriggering a monostable multivibrator from the disc drive's index pulse detector output. This allows the controller to detect that the drive contains a disc and that the disc is rotating before attempting a read or a write. Separate buses also allow both drive motors to run simultaneously. This improves the speed of disc copy operations since discs can be copied track by track without waiting for the drive motors to restart each time.

DMA (direct memory access) is used to transfer all data directly between the disc drives and the station's RAM. This is a lower-cost solution than using a sector buffer within the controller, and permits higher throughput because no processor intervention is required during the transfer of up to one entire track of data. Formatting of a new disc becomes particularly easy since an image of the track, including interrecord gaps, is merely placed in RAM and then transferred directly to the disc.

A ROM-driven state machine handles all communication between the host processor, RAM, and the 1791 disc controller. This state machine is a necessary link because the host processor and RAM use 16-bit words while the 1791 processes data in 8-bit bytes. All commands are passed between the host processor and the 1791 without delay.

The soft error rate of the disc system is reduced by using a phase-locked loop (PLL) data separator to recover the clock from the serial data stream stored on the disc. The natural frequency of the feedback loop is much lower than the 250-kHz bit rate of the serial data stream. This provides a "memory" that minimizes the effect of a bit whose position is slightly misplaced. The natural frequency used is 15.9 kHz, which was determined empirically. If the frequency is too high, the PLL will have insufficient memory; if the frequency is too low, the loop will have an excessively long lockup time.

Power Supplies

The new subsystem options for the 64000 System are faster and more complex than the emulation systems and logic analyzer modules available earlier. These new options require a correspondingly larger amount of power from the mainframe. The greater number of option choices also increases the power requirements, since a mainframe can have several hardware subsystems in place at one time. To handle this power demand and further increases expected in the future, the new 64100A mainframe power supply is designed to deliver 5V at 45A (primarily for TTL and CMOS circuits), −5.2V at 25A (primarily for ECL circuits) and −3.25V at 30A (for HP-designed bipolar LSI circuits). The power supply in the transportable 64110A mainframe delivers 5V at 30A, −5.2V at 20A, and −3.25V at 20A. It was necessary to modify the fans and internal ducting of the mainframes to ensure that the air temperature rise is no greater than 15°C above ambient at any point on the option boards.

Both mainframes are powered by switching-mode power supplies operating directly from the ac line. The design is conventional except that two LC filter sections are used in

© Copr. 1949-1998 Hewlett-Packard Co.

each secondary circuit of the three high-current sources rather than the usual single LC filter section. A two-section filter needs less total inductance, and therefore occupies less volume than a single-section filter providing the same attenuation. The output ripple voltage of each high-current supply is about 20 mV, peak-to-peak. The supplies are cooled by fans located between the power supply and the card cage. These fans draw outside air through the card cage and exhaust it through the power supply. This arrangement ensures that the card-cage cards are cooled before the more heat-tolerant components in the power supplies. High-dissipation components within the supplies, such as high-current rectifier diodes, are located on finned heat dissipators placed in the high-velocity airstream exiting from the fans. The dissipators also help mask some of the acoustic noise generated by the fan blades. Both supplies contain thermal shutdown switches to help protect them from the effects of excessively high temperatures should the air openings become blocked or the fans fail.

The 64100A's supply is partitioned into four modules. Three of the modules can be changed without removing the power supply from the 64100A Station. The fourth module is isolated behind a 1.6-mm-thick aluminum deck in the bottom of the supply. This subassembly has the power line input circuit, line rectifiers, storage capacitors, control power transformer, and EMI attenuating elements. The connector from this board to the remainder of the supply passes through the isolating deck and handles only the dc rail (rectified power line) to the switching supplies and the 12V control power supply. The metal deck prevents the intense high-frequency electric and magnetic fields generated by the switching circuitry from radiating around the EMI attenuating elements and reducing their effectiveness.

The high-current (greater than 5A) circuitry is contained on one printed circuit board to reduce the number of high-current interconnections. This board is made with extra heavy copper lamination to minimize power losses. Efficiency of the supply is good (73% at full power). To deliver 400 watts to the host system and option cards, about 550 watts of power is drawn from the power mains, including 35 watts for the system cooling fans.

Safety is a very important issue in power supply design, especially in one of this current and power capability. Insulation and spacings are designed to IEC 380 standards and flame-retardant materials are used throughout. Voltage sources are equipped with individual internal load-impedance-sensing circuits to minimize the current and power delivered under fault conditions. Thus, the short-circuit current is much less than the rated current for the supply.

Monitor circuits independent of the regulator loops make sure that supply voltages stay within safe limits and follow a proper power-up sequence. These circuits prevent damage to the modules in the system card cage if the power supply malfunctions. Independent shutdown loops and crowbar circuits activated by these monitors will burn open the main fuse in the power supply to prevent overvoltage conditions. In addition, shutdown can be initiated by either an overtemperature sensor, an interlock (to detect a missing or unplugged power supply printed circuit card) or a primary overcurrent detector. Six LEDs indicate which of the monitors caused the shutdown. Cycling the power switch resets all monitors. To aid in system fault diagnosis, five more LEDs are separately powered by the 5V, −5.2V, −3.25V, 12V, and −12V supplies. If one of these voltages is low or missing, the corresponding LED will glow dimly or be off.

Compatibility of Subsystems

The CRT display, high-power switching power supply, and two flexible disc drives in each mainframe are not the most hospitable of neighbors, particularly when placed in very close proximity as in the case of the 64110A Station. One major problem was magnetic interference from the display deflection yoke and flyback transformer, which coupled into the adjacent disc drives and caused soft read errors. Another problem was magnetic interference from the transformers and inductors within the power supply coupling into the display and causing a beat with the scan rate of the display. This beat would appear as a swimming motion of the displayed characters.

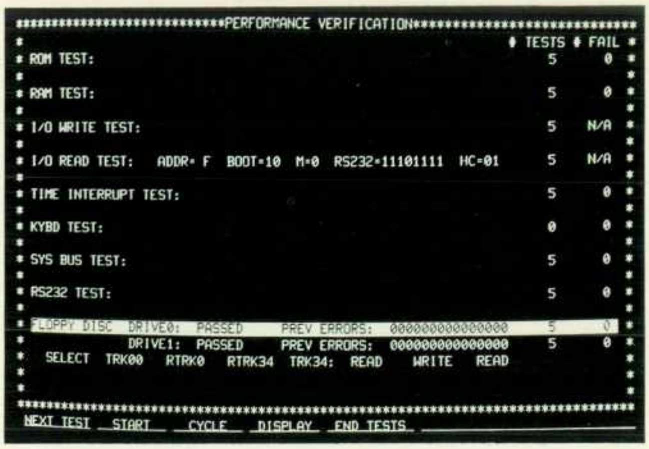
In both cases, direct measurement of the interference was extremely difficult. The design goal for the disc drive system was that installation in the station not seriously degrade the drive's specified soft error rate of no more than one error in every 10^9 bits read. Because the drive takes over 100 minutes to read 10^9 bits, it was very time consuming to verify the effectiveness of prototype changes. The CRT display presented different problems. When the display is operating, its yoke generates large magnetic fields in the vicinity of the CRT, making it difficult to measure any small interfering fields created by the power supply. Yet, a very small amount of display movement (less than ¼ dot width) caused by these small fields is visually apparent to a user.

To solve these problems, a directional magnetic probe was constructed by mounting a small, electrostatically shielded, multiturn coil at the end of a plastic rod. This probe was used to determine the sources of magnetic fields and to measure their relative intensities. A low-frequency spectrum analyzer was used to monitor places where the effects of the interference appeared in the form of electrical signals such as the output of the magnetic probe, power supply voltages, and signals in the disc drive read amplifiers. This made it possible to separate different sources of interference (by frequency) and to make quantitative measurements of their levels quickly so that the effects of any design changes could be measured. A technique that worked particularly well in the case of the disc drives was to reduce their timing margin artifically by skewing the read clock relative to the raw read data. This increased their sensitivity to interference so that any small changes in interference levels produced quickly recognizable changes in the error rate.

As a result of this testing, some of the internal sheet-metal pieces were redesigned to improve their shielding effectiveness. At the frequencies involved (20 kHz for the switching supply and 24.3 kHz for the display), the 1.6-mm-thick aluminum used for the internal sheet metal has a thickness of several electromagnetic skin depths and can be an effective magnetic shield. This avoids the cost or weight penalties associated with using high-permeability shield-

© Copr. 1949-1998 Hewlett-Packard Co.

ing materials.



```
****************************PERFORMANCE VERIFICATION******************************
*                                                               # TESTS # FAIL *
* ROM TEST:                                                        5       0   *
*                                                                              *
* RAM TEST:                                                        5       0   *
*                                                                              *
* I/O WRITE TEST:                                                  5      N/A  *
*                                                                              *
* I/O READ TEST:   ADDR= F  BOOT=10  M=0  RS232=11101111  HC=01    5      N/A  *
*                                                                              *
* TIME INTERRUPT TEST:                                             5       0   *
*                                                                              *
* KYBD TEST:                                                       0       0   *
*                                                                              *
* SYS BUS TEST:                                                    5       0   *
*                                                                              *
* RS232 TEST:                                                      5       0   *
*                                                                              *
* FLOPPY DISC  DRIVE0:  PASSED    PREV ERRORS:  000000000000000    5       0   *
*              DRIVE1:  PASSED    PREV ERRORS:  000000000000000    5       0   *
*  SELECT  TRK00  RTRK0  RTRK34  TRK34:  READ    WRITE   READ                  *
*                                                                              *
*                                                                              *
********************************************************************************
NEXT TEST  START   CYCLE   DISPLAY  END TESTS
```

Fig. 3. *Display on 64100A Development Station showing performance verification (self-test) menu. Flexible disc test line is shown in inverse video.*

Safety and Electromagnetic Interference

Another aspect of compatibility is the interaction of an instrument with its surroundings. Considerations of safety and electromagnetic interference revolve around a number of standards, which have been written both within the U.S.A. and abroad. These regulations are also covered by Hewlett-Packard design standards and required some redesign and retesting. Compliance was made more difficult by the increased power supply capability within the mainframes, which increased their noise-generating potential. The addition of state and timing analysis modules to the mainframes also made compliance more difficult because of their internal high-speed circuitry and because their probe cables can act as transmitting antennas, thus increasing radiated interference. To cover this situation, compliance testing was performed for a typical user setup, with the cables hanging over the edge of a table.

Both of the new mainframes comply with the following regulations: IEC 348, ANSI C39.5, CSA Bulletin 556B, VDE 0871 and VDE 0875 Level B, and FCC part 15, subpart J, Level A. FTZ RFI licensing is in process. The option cards are designed to meet VDE Level A except when probing open circuitry. In that situation, emissions will be a function of the target system.

Self-Test

To give the user confidence in the operation of the instrument and aid in fault diagnosis, approximately one-half of the 30K-byte internal ROM is devoted to the storage of self-test routines. When the instrument is first powered on, it computes and verifies a checksum for each ROM. The checksum for each ROM is unique. Thus, the routine can detect ROMs in the wrong socket as well as defective ROMs. RAM is tested by writing the value of a software counter to each location in sequence, then verifying that the stored value is correct. If so, the program waits one second to check the refresh circuitry and verifies the value again. If this test is passed, the program complements the count and repeats the sequence. If an error is found, the name of the failed component is displayed on the CRT. If the failure is so serious that the display cannot function, the CPU automatically enters a software loop that provides the proper stimulus for signature analysis.

Extensive self-test routines can be selected by means of a switch on the rear panel or by pressing the **CNTL** and **RESET** keys. The menu for these routines is shown in Fig. 3. The flexible disc self-test routine verifies the ability of each drive to perform seek, read, and write operations. The write test is performed on the spare track so that no user-entered data is destroyed. If the spare track is in use, a message is displayed and the write test is not performed. A separate menu allows access to a disc diagnostic program. This program enables service personnel to test the ability of the drive to read or write any track. It also allows them to perform extended error rate tests.

Serviceability

Two serviceability goals for the new mainframes were to reduce the mean time to repair (MTTR) to less than two hours and to troubleshoot and repair all boards to the component level, thus avoiding the inventory cost of stocking replacement boards in service centers. It is anticipated by our service group that component-level service should reduce repair costs to customers by 42% and decrease the time a failed instrument is out of service. The primary means used to achieve these goals is signature analysis (SA).[2] To implement SA, latches are designed into the mainframe circuitry to provide the start and stop pulses that define the SA window. Jumpers placed in several positions on the printed circuit boards allow hardware and software feedback loops to be broken so that signatures can be obtained regardless of the possible malfunctioning of other parts of the instrument. SA tables in the service manuals for each instrument contain over 500 individual signatures which, in most cases, permit troubleshooting to the component level.

Because of its compactness, it was anticipated that access for service could be difficult with the smaller transportable 64110A mainframe, so particular attention was given to this potential problem during its design. Its power supply module can be unplugged after removing only nine screws. Almost all of its circuitry is contained on three plug-in printed circuit boards. The boards in the card cage can be placed on extender boards for service. All of the remaining boards that contain active circuitry are designed to swing out to permit access to their components.

Reliability Testing

Another, more effective way to reduce the service requirements of an instrument is to reduce the probability that it will fail. Toward this end, both mainframes underwent extensive reliability testing during their pilot runs and first production runs. The goal of this early testing was to find failures and to make changes in the design, the production process, or the vendor parts to eliminate their causes. A lack of failures, although supportive to the egos of the design team, produces no useful data, so attempts were made to increase the amount of data gathered per unit hour of testing. Table I shows the results of an experiment conducted during the 64110A pilot run where the instruments

© Copr. 1949-1998 Hewlett-Packard Co.

Table I

Test	% of Total Test Time	% of Total Failures	Test Hours
Heat Box	43%	10%	13,357
Shake Table	24%	39%	7,247
Thermal Cycle	5%	39%	1,510
Drive Test	28%	12%	8,786
Total:	100%	100%	30,900

were subjected to three different types of reliability tests. Heat box is a classical heat run at 35°C to 40°C, to simulate accelerated usage. Shake table is a heat run during which the instruments are also subjected to vibration at 0.8g, 20 Hz, for 10 minutes every hour. Thermal cycle is a strife test[3] that cycles the instruments between −20°C and 65°C at a rate of 1°C/minute with a 30-minute dwell at the extremes. The power is cycled on and off three times at each extreme. Drive test is a separate disc drive test conducted at room temperature. The thermal cycle test was, by far, the most effective since it produced 39% of the total failures while consuming only 5% of the total test time. The failures generated during the thermal cycle test correlate with failures seen in field failure histories of similar instruments. This indicates that temperature cycling accelerates failures that would have occurred normally rather than generating new failures caused by excessive thermal stress.

Acknowledgments

Special thanks are due the members of the design team not only for their hard and innovative work, but especially for the patience and the versatility they showed in handling the many details that become a necessary part of any large project such as this one. The following people played major roles in the successful completion of the new 64110A and the revised 64100A. The members of the 64110A mechanical design team included Ric Bechter, keyboard, hinge, and front bezels, and Bob Self, rear frame, door, side feet, handle, and mechanical design finalization. Pete Rawson designed the supply for the 64110A. Mechanical design of the early prototypes of the new 64100A power supply was done by Don Bloyer. Ron Duffy did the fault tests and safety evaluations and Brent Peterson designed and implemented the stress/strife and extended heat-run testing plan. The self-test software and the flexible disc driver software were written by Doug Little and Kip Fulghum, respectively. The 64110A benefited from Rick Vestal's work on the display and from his suggestions for improving its serviceability. Roger Molnar did the early EMI studies on both stations. The clean and functional industrial design of both stations was achieved by Bill Fischer and Mike Walker.

References

1. T.A. Saponas and B.W. Kerr, "Logic Development System Accelerates Microcomputer System Design," Hewlett-Packard Journal, Vol. 31, no. 10, October 1980.
2. R.A. Frohwerk, "Signature Analysis: A New Digital Field Service Method," Hewlett-Packard Journal, Vol. 28, no. 9, May 1977.
3. K.F. Watts, "A Unifying Approach to Designing for Reliability," Hewlett-Packard Journal, Vol. 32, no. 7, July 1981.

Carlton E. Glitzke

Carl Glitzke joined HP in 1965 with six years experience in product design. At HP, he has worked in product design engineering, has supervised a thick-film production area, has been a manufacturing project engineer, and is now in product design in HP's Logic Analysis Division where he coordinated the mechanical design of the new 64000 Stations. Born in Canon City, Colorado, he is married, has four children, and lives in Manitou Springs, Colorado. During off hours, he is interested in four-wheel drive vehicles, off-road motorcycles, and outdoor activities with his family.

Alan J. DeVilbiss

Born in Roanoke, Louisiana, Al DeVilbiss attended Louisiana Tech University, earning a BSEE in 1960. He continued his studies at the California Institute of Technology, receiving an MSEE degree in 1961. With HP since 1965, Al has designed high-frequency amplifiers for oscilloscopes, the operating system for the 1602A State Analyzer, and the I/O drivers and multiterminal parts of the 64000's operating system. His work has resulted in two patents and three other HP Journal articles. He is married, has two teenage children, and lives in Colorado Springs, Colorado. Woodworking is a long-standing hobby. His most recent project, a violin for his daughter, took about 1½ years to complete.

Jeffrey H. Smith

Jeff Smith joined HP in 1963 after receiving BSEE (1962) and MSEE degrees at Stanford University. He contributed to several units of HP's sampling oscilloscope family, was project leader for the 1815A Reflectometer, was responsibile for four high-speed modules in the 1900 Series Pulse Generators, and contributed to the 1611A State Analyzer. Jeff worked on the 64000 Stations before transferring recently to HP's Telecommunications Division in Colorado Springs, Colorado. His work has been reported in two other HP Journal articles and two conference papers. Born in Portland, Oregon, he now lives in Colorado Springs. He is married, has two children, and enjoys running, skiing, woodworking, and restoring a vintage Corvette.

© Copr. 1949-1998 Hewlett-Packard Co.

A Modular Analyzer for Software Analysis in the 64000 System

by Richard A. Nygaard, Jr., Fredrick J. Palmer, Bryce S. Goodwin, Jr., Stan W. Bowlin, and Steven R. Williams

THE LAST TEN YEARS have produced a revolution in microprocessor technology. In 1972, the first-generation 8008 microprocessor was a novel element in product design. Program sizes were in the hundreds or perhaps thousands of bytes. Machine-code programming was not uncommon and assembly language was used for larger programs. In 1982, the third-generation 68000 and similar processors entered into their second upgrade and program lengths reached a megabyte and beyond. High-level languages and advanced data structure techniques are used extensively. Most programmers are no longer hardware designers, but rather are software engineers and computer scientists. The expense of software development is exceeding that of the hardware. This adds up to a necessary revolution in the techniques used to design microprocessor-based products.

Each generation of processors has been supplied with its own generation of development and analysis tools. These have progressed from HP's 1601A Logic State Analyzer plug-in for an oscilloscope, through the HP 1611A and 1610A Logic State Analyzers, to the current HP 64620S Logic State/Software Analyzer (Fig. 1). The change in emphasis from hardware to software design can be seen in these products, as well as in their capabilities. The 1601A was controlled by toggle switches, contained a comparative handful of ICs, and displayed its measurements in binary notation. It served random logic and discrete state machine design. The 1611A and the 1610A are microprocessor-based designs with menu control and display in mnemonics and selectable number bases, respectively. Their measurements are aimed at the needs of assembly language debugging. The 64620S is a microprocessor-based design with directed-syntax softkey commands. Its displays include program symbols as well as mnemonics and numerical data. It assists programmers in high-level languages with a full feature set that includes software performance measurements and extensive program tracing.

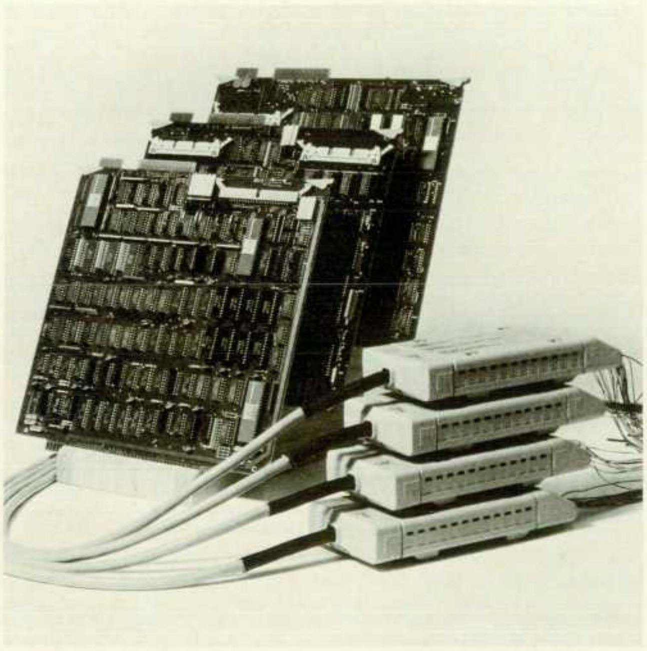


Fig. 1. *HP Model 64620S Logic State/Software Analyzer adds real-time, transparent software analysis to the HP 64000 Logic Development System. The analyzer can be configured with 20 to 120 input channels. General-purpose and dedicated interfaces simplify connection to target systems. The 64620S can be incorporated in a 64000 System cluster station, or in a 64100A or 64110A Development Station with a flexible disc drive as a stand-alone software analyzer.*

The wide variety of measurement situations to which the 64620S is directed requires a high degree of adaptability. The many types of target systems to be monitored require a probing system that can interface to the mechanical, electrical and functional characteristics of these systems. Also, the user interface to the analyzer needs to be configurable to the different ways in which information is represented. For these reasons, it was considered very important to make the 64620S as user-definable and configurable as possible.

Additional design constraints for the 64620S Logic State/Software Analyzer were generated by requiring that it be a module in the 64000 System. The software design had similar constraints, because it also has to operate within the environment provided by the 64000's operating system.

Modular Feature Set for Tracing Modular Software

An extensive feature set can be both a blessing and a curse, a blessing in that almost any measurement problem can be attacked, and a curse in that the choices are so many as to obscure the necessary ones. Today's software analysis problems require a very capable analyzer. Problems introduced by modular, high-level software are at a higher level than many of those encountered in assembly language programming. Traces of variables, tasks, and data structures are required.

A high-level language allows the designer to attack the problem in pieces, producing modular building blocks to construct the program and complete the task at hand. Each block performs a separate function, which is relatively independent of the way other blocks behave. The 64620S attacks software analysis problems in the same way. The three functions required to do a useful software trace are trigger

© Copr. 1949-1998 Hewlett-Packard Co.

(WHEN is program activity significant), storage (WHAT activity is important), and count (HOW MUCH activity occurred between important activities). The 64620S provides symmetrical capabilities for each of these functions.

In the 64620S State Analyzer, trigger, store, and count are equal functions. Earlier analyzers emphasized the trigger (breakpoint) function to the near exclusion of the storage and count functions. This approach is usually sufficient to trace assembly language programs where each bus cycle may be equally significant. However, the bus cycles executed by a high-level program are rarely of interest because most reflect code automatically generated by the compiler. Instead, the relatively infrequent accesses to variables and procedure calls are of primary interest to the programmer. In this environment, storage qualification takes on a role at least as great as the trigger function. And, since most bus cycles are now ignored by the analyzer, the importance of counting the time between the few that are stored also increases. A time count immediately shows the overhead involved in modifying data structures and the delays associated with each procedure or task. By providing equal capability for each function, the user can emphasize the functions(s) most needed for the measurement at hand.

Another key aspect of the building block approach to software analysis is a windowing capability. A window function is a repetitive search for an event to enable the function (open the window) followed by a search for another event to disable the function (close the window). Programs contain varying levels of control, from the main program to the utility or driver functions. The context in which a variable is changed is just as important as the fact that it was changed. Windowing is the way a user can tell the analyzer when the program reaches a context of interest.

Windowing can be applied to trigger, store, or count. The same window can control all three or a different window can be defined for each. Each window enables and disables one or more trace functions. In the 64620S, up to three different windows can be defined simultaneously. Thus, the trigger context can be defined by a trigger enable window, storage by a storage enable window, and count by a count enable window. Few measurements require such extensive control, but it is available when required.

The highest level of control in the analyzer corresponds to the highest level of control over a program—the scheduler. In a multitasking environment, programs are rarely run to completion. Instead, each task may be active for only a few milliseconds before deferring to another. The programmer often is not concerned with these details, preferring to view the program as executing continuously. The analyzer should also support this view. The 64620S can be told to suspend its operations whenever the program is suspended by the scheduler and to resume them when the program is resumed. This capability is called the master enable function; it freezes the rest of the analyzer including the trigger, store, and count functions. Program swapping is transparent to the analyzer and the program when the master enable function is in use.

Thus, the 64620S State Analyzer can extensively analyze software execution in real time without disturbing program execution in any way. The measurement specification complexity has been reduced by modularizing the measurement functions. Fig. 2 shows the hierarchy of these functional building blocks within the 64620S. Measurements can be built from any combination of blocks to solve particular needs. Furthermore, measurements can be refined easily by adding or modifying blocks to control the capture of data better, since each is displayed individually within the trace specification. Blocks not defined by the user are defaulted to an "always" condition. Therefore, if no blocks are specified, the analyzer captures the bus cycles seen after execution is begun. Specifying only the trigger condition is equivalent to using a breakpoint-only analyzer. A "trace triggers" analyzer is produced if only storage qualification is used. Running both simultaneously produces a powerful measurement, but one that still does not take full advantage of the analyzer's capabilities.

Overview for Performance Measurement

Tracing program execution is a traditional task for logic analyzers. A newer task, and one growing in significance, is monitoring program activity to provide data about the program's performance. Some of the features of the 64620S, particularly the time count, provide performance information. However, these features are optimized for tracing, not for overview. The 64620S Logic State/Software Analyzer addresses the need for performance measurements with a second overview analyzer that is separate from the traditional trace analyzer and capable of operating simultaneously with it.

The overview analyzer provides an overview of system activity. This analyzer captures events, not the bus cycles captured by the trace analyzer. This means system activity is analyzed at the level of procedures and tasks, not at the instruction level. Then too, the information is displayed most often in histograms and graphs, not in a list.

Three different overview measurements are provided, each with a different type of event definition. The simplest and most often used is overview on address. This mode monitors activity on the system address bus, allowing global views of program flow, data accesses, I/O activity, and other operations. Events are defined as ranges of address values. Each event can correspond to the program area containing a specific routine or collection of utilities. Or, an event can represent the entry point to a procedure or the area occupied by a data structure. Once the events of interest are defined, the overview analyzer monitors the sys-

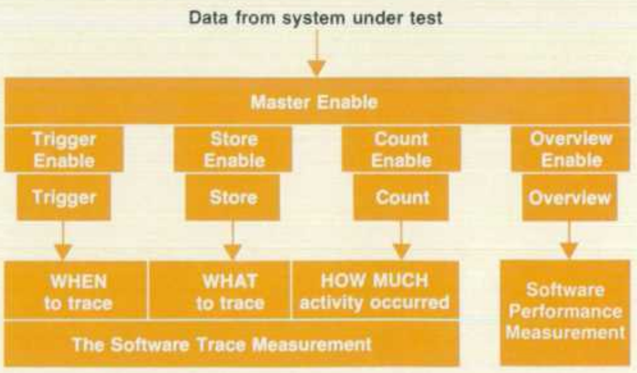


Fig. 2. *Diagram showing the hierarchy of the measurement function modules in the 64620S.*

© Copr. 1949-1998 Hewlett-Packard Co.

Range Detection in the 64620S State Analyzer

The comparison between an input value and a range of predefined values is a recurring need in logic analysis. The range of values may represent the address locations of the contents of an array, or it may be the boundaries of a program module. Such a range specifies the areas of address space of interest to a user. Earlier analyzers have provided only a single range, if any at all, because of the expense of range detection compared to simple pattern comparison. The user, on the other hand, would like many ranges since programs contain many arrays, modules, and other blocks. Ranges are the most general way of describing the boundaries of these regions in address space. Furthermore, overview measurements containing up to fifteen events place heavy demands upon range comparators because each event represents a different range of values.

Previous comparator designs relied upon RAM comparators followed by gating to detect the range. Such schemes required at least two RAM outputs and a gate for each range. In contrast, a simple pattern comparison can be accomplished with a single RAM output feeding a wired-AND circuit with no other gating.[1] The approach taken in the 64620S Analyzer replaces the gating with another RAM. This RAM is used as a programmable logic element to allow greater freedom in encoding the outputs of the range detector RAMs. The result is that two 1024-by-4-bit RAMs and one 256-by-4-bit RAM can detect four independent ranges on twenty bits. The same three RAMs can also be used to detect up to fifteen nonoverlapping ranges for use as overview events. Using the earlier method, four independent ranges would have required four 1024-by-4-bit RAMs plus four gates, while fifteen ranges would have required fifteen RAMs and gates.

Fig. 1 shows a block diagram of the range detector. The twenty-bit input value may represent either an address from the user's system or the output of a time interval counter. This value is split into an upper ten-bit portion and a lower ten-bit portion, each feeding a separate RAM, just as an older method would. The difference is that the output of these RAMs is not fixed in meaning, but reflects the specifications of all ranges to be detected. There is not a direct correspondence between each output and each range. Instead, the lower RAM decodes the outputs and restores the correspondence. Its four output bits represent either four independent ranges for trigger, store, or count qualification or one of fifteen events for overview measurements.

The value applied to the range decoder can be thought of as one point on a line containing 2^{20} hexadecimal values (00000 through FFFFF). The ranges specified by the user divide this line into segments bounded by the range endpoints. When a value is sampled from the user's system address bus, it falls at one point on the line. This point may be contained within none, one, or more of the originally specified ranges. The range decoder determines which case it represents as follows:

1. Values on the line are described by a base-1024 number system. Any value can be described with just two digits, call them J and K. The ranges specified by the user are redefined in terms of JK pairs and the line is segmented by the endpoints of the ranges. Because the ranges can overlap or not cover the entire line, each segment may represent none, one, or more ranges satisfied.
2. The line, divided into segments by the user specified ranges, is an ordered list of JK values. When an input value is received, it is placed on the line to determine which segment it is within.
3. Consider the J digit of the input value. It specifies a coarse position on the line. This may or may not be enough to decide which segment the input value is within. If it is, then the K digit is ignored. If not, then the K digit is consulted to complete the determination. In effect, a given J value asks a question about which segment the input value is in. The question may sometimes be answered immediately if the K value does not matter (point located in the middle of a long segment), but it may require knowledge of the K value as well (point located in a very short segment). The result is that a translation can be made from the large list of different J values to a much shorter list of J types. Each J type may represent a directive (if the K value does not matter) or a question (if the K value does matter). Each segment on the line may require one, two, or three different J types to decode it properly. One is required if the endpoints of the segment have the same J value, two are required if the J values differ by one, and three are required if the J values differ by two or more. The J-RAM in Fig. 1 translates from the ten-bit J value to the four-bit J type.
4. Now consider the K digit of the input value. It represents the ten least-significant bits of the value and, in conjunction with the J digit, completely specifies where the input value lies on the line. But the J value has already been replaced by the J type, and each J type represents a different question about the K value. Each K value will produce an answer to each of the J-type questions—either yes or no. Therefore, a given K value produces a given set of answers, one for each question. Again, many K values may produce the same set of answers. These may be grouped together into a single K type. The K RAM in the block diagram translates from the ten-bit K value to the four-bit K type.
5. The input value has now been replaced by a J type, which represents one of a list of questions, and a K type, which represents one set of answers. All that remains is to match question with answer. This is performed by the ID RAM in the block diagram. The combination of a particular J type with a particular K type produces a single answer, indicating which, if any, of the originally specified ranges are true. This answer is output from the ID RAM for use in trace qualification or in overview.

Sixteen J types and sixteen K types are sufficient to decode up to four arbitrary doubly bounded ranges. These ranges may overlap and may be inclusive (true within the bounds) or exclusive (true outside the bounds). This mode is used for trigger, store, and count qualification. Overview events, on the other hand, are defined to be nonoverlapping. This ensures that each input value produces only one overview event. With this restriction, five ranges can always be decoded correctly. However, in many

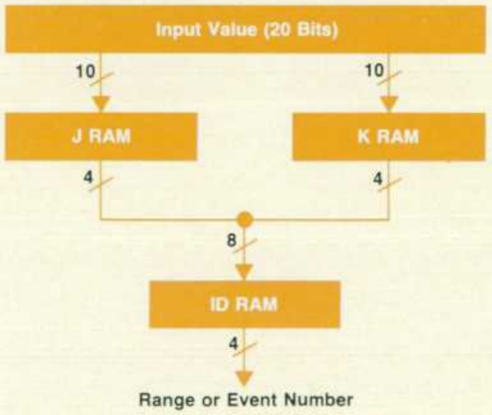


Fig. 1. *Block diagram of the range detector used in the 64620S Logic State/Software Analyzer.*

© Copr. 1949-1998 Hewlett-Packard Co.

cases, it is possible to take advantage of the flexibility of the range decoder to allow more events.

The limiting factor in most range specifications is the number of J types available. If each overview range is relatively small (i.e., less than 1024 locations wide) it will require no more than two J types to decode it. In this case, up to eight events may be specified. Similarly, up to fifteen events can be specified if each is a single value. The analyzer firmware counts the number of J and K types required to determine if the event specification can be loaded.

Reference

1. W.A. Farnbach, "The Logic State Analyzer—Displaying Complex Digital Processes in Understandable Form," Hewlett-Packard Journal, Vol. 25, no. 5, January 1974.

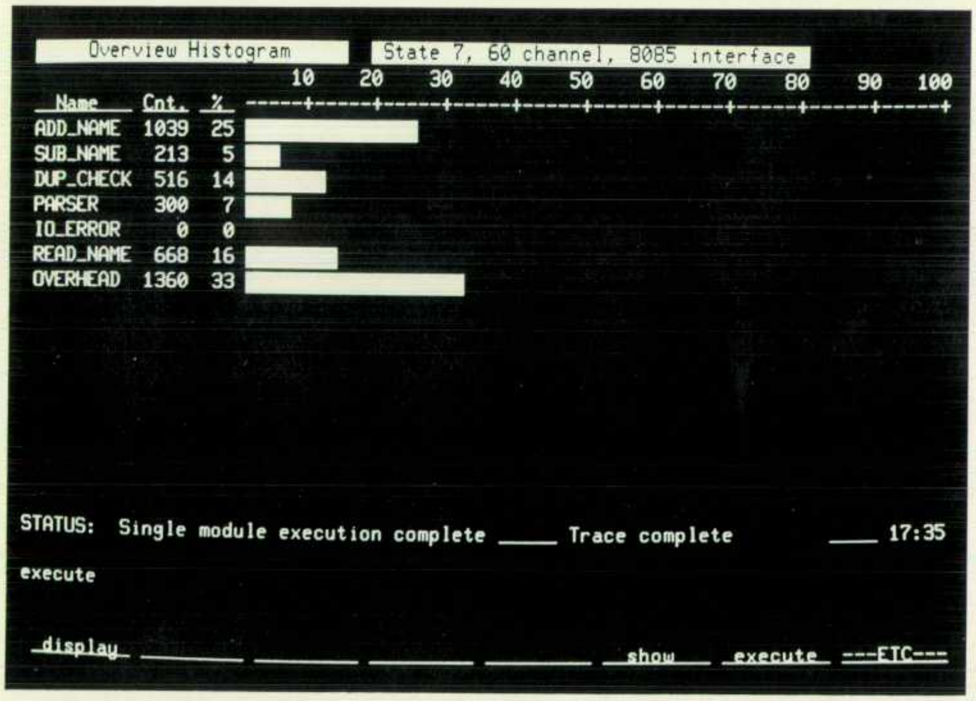
tem address bus for values, if any, corresponding to each event. As events are detected their number is accumulated in the overview memory and displayed either as part of a histogram (see Fig. 3), a graph, or a list. One useful event definition is everything-else. This groups all system overhead into a single event for comparison with other events defining foreground system activity. This mode, as well as the others, can use the analyzer's sequencer, which provides a wide variety of enabling functions for a more refined triggering definition, to window the overview. This provides a way to perform sampled, or statistical overview measurements.

The other two overview modes are similar in that the events in both correspond to ranges of count accumulated during an interval defined by a window. For example, event A represents a count of 0 to 5, event B represents a count of 6 to 12, and so on. The count can represent either elapsed time or the number of occurrences of special states of interest such as interrupts, procedure calls, or variable accesses. The window defines when the interval begins, resetting the counter to zero, and when it ends, decoding the count value into the associated event and storing the event's occurrence in the overview memory. Further windowing of the count is also possible by defining a different window. The result is a measurement that shows the distribution of the different amounts of time required by a task each time it is called or the variation in resource use as it is called again and again. These measurements are valuable ways to observe execution delays, message transmission retries, and other similar activities.

Events can also define unexpected performance. Some areas of address space may be undefined and should never be accessed. An I/O device may require service within a maximum amount of time. A task may take too long to execute. Problems discovered by the overview analyzer often require a trace of program execution to discover their source. The independence of the trace and overview analyzers within the 64620S makes these measurements possible. One overview event can be selected to trigger the trace analyzer, capturing the program execution leading up to an anomalous overview event. For instance, the overview analyzer provides a trigger when the execution time of a routine exceeds a limit or a data transmission attempt fails too many times before success. The parameters and program activity leading up to the failure can then be examined with the logic state analyzer.

Analyzer Hardware Architecture and Design

The hardware architecture and implementation of any analyzer limit what analysis functions can be performed in real time. The 64620S has a highly versatile measurement feature set optimized for monitoring software activity at both a detailed level and from an overview. Thus, system activity can be viewed at the level most appropriate to



```
  Overview Histogram          State 7, 60 channel, 8085 interface
                         10    20    30    40    50    60    70    80    90   100
 Name      Cnt.  %    -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
ADD_NAME   1039  25
SUB_NAME    213   5
DUP_CHECK   516  14
PARSER      300   7
IO_ERROR      0   0
READ_NAME   668  16
OVERHEAD   1360  33

STATUS:  Single module execution complete ____ Trace complete         ____ 17:35
execute

 display  ______  ______  ______  ______  show  execute  ---ETC---
```

Fig. 3. *Overview histogram indicating activity in various subroutines relative to the total program time.*

© Copr. 1949-1998 Hewlett-Packard Co.

observation of program flow, data flow, process timing, or system performance.

A block diagram of the 64620S is shown in Fig. 4. Physically, the analyzer consists of two or more circuit cards. A control card is combined with one or more acquisition cards to provide from 20 to 120 analysis channels, expandable in 20-channel increments. The control card contains circuitry independent of the number of data channels, while the 20- and 40-channel acquisition cards contain circuitry associated with additional channels. This architecture minimizes the cost per channel by including the analysis overhead circuitry only once. Also, connections between cards are few, which increases the reliability of the system. A synchronous expansion bus between the cards need carry only timing and pattern information, and it is not necessary to distribute the many data channels throughout the system.

Two Memories Are Better than One

Two separate memories are included in the acquisition system of the 64620S. A trace memory stores data from the input channels, as well as count information and sequencer status. This memory can store up to 256 states and can be displayed either in trace list format with inverse assembly or in a graphic format. An overview event memory is used to capture information that allows the relative occurrence and order of occurrence of defined events to be measured. This memory can store up to 4096 events and can produce a histogram, a graph, or a list output for display. This dual memory architecture offers simultaneous monitoring of program activity at both a detailed level and an overview level. This allows correlation of real-time information.

Since the trace memory stores information received on the input data channels, each acquisition card contains a section of that memory. The control card contains the section that does not change in size with increasing channel width, namely the state/time counter and sequencer status values. This distribution of the trace memory offers advantages over a centralized architecture, in particular that of expanding easily with an increasing number of channels. However, coordinating the activities of the memory becomes more complicated when it is distributed, especially when the storage control features are as extensive as those incorporated into the 64620S. Reading the trace memory during a trace with limited clock rate allows displaying acquired data before a measurement is completed. This technique, called interactive read, is important when the stored data is highly qualified, which means that only infrequently selected pieces of data are written into the memory. In this case, a measurement could take seconds, minutes, or even hours, and it is not desirable to stop the trace to see the data because some data might be missed.

The overview memory has the task of capturing decoded information called events. These events are decoded by the range decoder circuit (see box on page 18) and consist of simple four-bit values. The small size of the event number allows this memory to store more values. The overview memory also incorporates an interactive read feature, allowing histogram data to be updated as new events are detected by the overview analyzer.

The dual-memory architecture allows simultaneous, correlated measurements. In particular, a trigger from the overview memory to the trace memory allows tracing of detailed data relating to an overview event occurrence. For example, an overview event of time ranges can trigger the trace memory to trace the parameters passed to a routine that takes longer or shorter than a specified period of time.

Custom ICs Are the Key

Using custom integrated circuits in many functional areas allows the extensive set of real-time analysis capabilities in the 64620S. The interface to the 64000's high-speed intermodule bus (IMB) is closely associated with the analysis functions of the 64620S. Because of the bidirectional nature of the IMB and the number of controlled functions, the interface circuits were incorporated

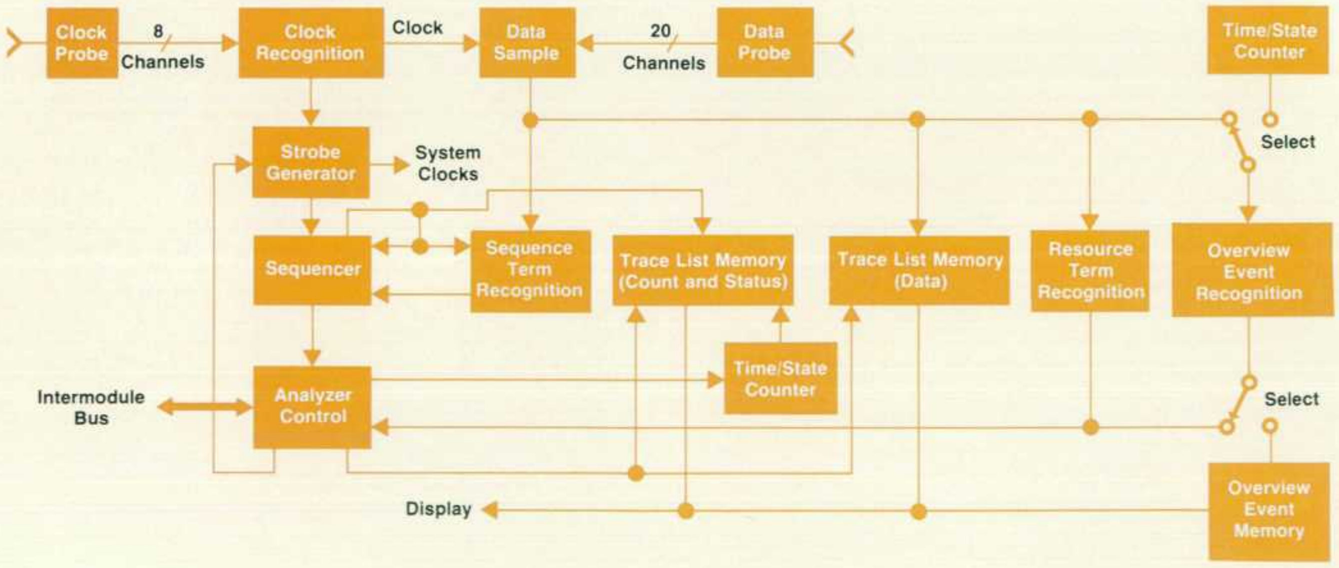


Fig. 4. *Block diagram of the 64620S Logic State/Software Analyzer. This configuration is a 20-channel analyzer with overview.*

© Copr. 1949-1998 Hewlett-Packard Co.

Inverse Assembly for a General-Purpose Logic Analyzer

An inverse assembler combines captured address, data and status information to generate a mnemonic representation of the machine code for the processor being monitored. An inverse assembler performs the opposite function of an assembler; it translates CPU machine code back into the mnemonic representations used in the generation of the CPU program.

Inverse Assembly Language

The inverse assembly software for the 64620S Logic State/Software Analyzer was developed using a specially designed language called the Inverse Assembly Language (IAL, see Fig. 1). Resembling an assembly level language for the most part, IAL also contains instructions characteristic of a higher-level language, several of which are dedicated to specialized inverse assembly related functions.

An interpreted code, IAL can be thought of as running on a 32-bit pseudomachine. A wide range of logical, arithmetic and utility instructions operate on a single 32-bit accumulator. In addition, a large number of 32-bit variables can be defined by the program, any of which can be transferred to or from the accumulator. Alphanumeric string constants of up to 64 characters can be defined, as well as 32-bit numeric constants.

Included in IAL is a CASE statement, whose index may be a group of accumulator bits or an entire 32-bit variable. An IF/THEN construct adds considerably to the high-level programming capability of the Inverse Assembly Language.

Several predefined variables pass address, data, and status information from the analyzer trace memory to the inverse assembler. In the case of multiple byte instructions, or other similar situations, the INPUT instruction provides the inverse assembler with the capability to read additional captured states from the trace memory. The inverse assembler uses the OUTPUT instruction to display its results.

For users of proprietary machines or other processors not supported by Hewlett-Packard, IAL provides an essential tool to developers of inverse assemblers. A specialized development language such as IAL is of additional benefit in simplifying development and reducing inverse assembler design-cycle time.

Inverse Assembler Operation

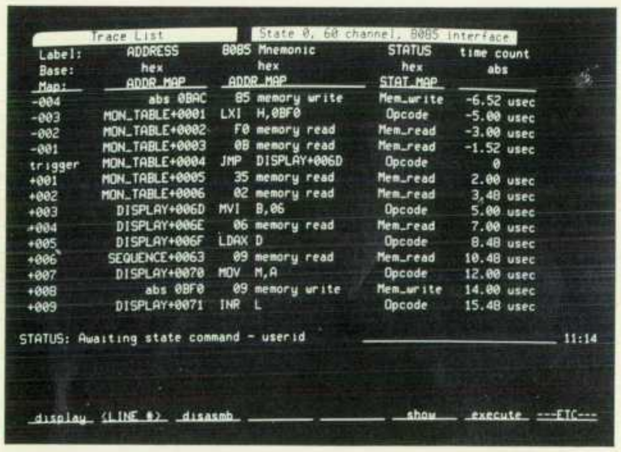
Upon a request to display a currently undisplayed CPU state, and if the mnemonics display option is selected in the trace list, the analyzer performs a call to the inverse assembler software module, passing to it address, data and status information for the new state. The inverse assembler then performs an operation appropriate to the status of the new state, formats a display and returns to the analyzer software. At this point, the inverse assembly display is written onto the instrument's CRT (see Fig. 2) and preparations are made to call the inverse assembler again if necessary.

Inverse assembly goes beyond simply displaying the mnemonic for the instruction being executed. Using the INPUT instruction provided in IAL, the entire instruction, including instruction operands, is displayed in the same form as was used in programming the CPU. In most cases, the mnemonic inverse assembler output can be assembled directly.

Many instruction types common to all CPUs specify source and/or destination addresses within the instruction (jump and call, for example). In writing programs using such instructions, programmers prefer using symbolic address labels rather than numeric addresses. The inverse assembler can obtain or calculate the address from the instruction itself. Using the address mapping function provided by IAL, it can then locate a corresponding symbol within the analyzer address map and display the same symbolic address as used by the programmer. Symbolic tracing clarifies and simplifies analyzer measurements, and relieves the programmer of having to remember often meaningless absolute addresses.

```
OPERAND_COLUMN      CONSTANT   6                 ;Display position for operand
STRING_JMP          ASCII      "JMP"             ;String used several times
REGISTER_NUMBER     VARIABLE   0                 ;Save register #, initially 0
DECODE_ASR_LSR_JMP                               ;Decode ASR, LSR, JMP
  LOAD INITIAL_DATA                              ;Reload initial opcode
  CASE_OF 2,1                                    ;Accumulator bit 2-1
    OUTPUT "ASR"                                 ;    Bit 2-1 = 00B
    OUTPUT "LSR"                                 ;    Bit 2-1 = 01B
    GOTO ILLEGAL_INSTRUCTION                     ;    Bit 2-1 = 10B
    OUTPUT STRING_JMP                            ;    Bit 2-1 = 11B
  CASE_END
  POSITION ABS,OPERAND_COLUMN                    ;Move to operand column
  CASE_OF 2,2                                    ;Accumulator bit 2
    CALL ASR_LSR_OPERAND                         ;Decode ASR and LSR
    CALL DISPLAY_DESTINATION                     ;Show JMP address
  CASE_END
  RETURN                                         ;Return to analyzer
ASR_LSR_OPERAND                                  ;ASR, LSR, subroutine
  INCREMENT INPUT_ADDRESS                        ;Point to next opcode
  INPUT ABS,INPUT_ADDRESS,QUALIFIED              ;Try to read next opcode
  IF INPUT_ERROR () 0 THEN GOTO ERROR            ;Leave if error
  LOAD INPUT_DATA                                ;Get new opcode value
  IF 7,7 = 1 THEN AND 00001100B                  ;Mask if necessary
  ROTATE RIGHT,2                                 ;Move to lower 2 bits
  STORE REGISTER_NUMBER                          ;Save register # for later
  CALL SHOW_REGISTER                             ;Display register name
  RETURN                                         ;Subroutine return
```

Fig. 1. *Example code from an inverse assembler written in IAL.*



```
         Trace List         State 0, 60 channel, 8085 interface
Label:      ADDRESS        8085 Mnemonic          STATUS    time count
Base:         hex              hex                  hex        abs
Map:        ADDR_MAP        ADDR_MAP             STAT_MAP
-004             abs 0BAC    85 memory write     Mem_write   -6.52 usec
-003     MON_TABLE+0001  LXI  H,0BF0             Opcode      -5.00 usec
-002     MON_TABLE+0002    F0 memory read        Mem_read    -3.00 usec
-001     MON_TABLE+0003    0B memory read        Mem_read    -1.52 usec
trigger  MON_TABLE+0004  JMP  DISPLAY+006D       Opcode          0
+001     MON_TABLE+0005    35 memory read        Mem_read     2.00 usec
+002     MON_TABLE+0006    02 memory read        Mem_read     3,48 usec
+003       DISPLAY+006D  MVI  B,06               Opcode       5.00 usec
+004       DISPLAY+006E    06 memory read        Mem_read     7.00 usec
+005       DISPLAY+006F  LDAX D                  Opcode       8.48 usec
+006      SEQUENCE+0063    09 memory read        Mem_read    10.48 usec
+007       DISPLAY+0070  MOV  M,A                Opcode      12.00 usec
+008            abs 0BF0    09 memory write     Mem_write   14.00 usec
+009       DISPLAY+0071  INR  L                  Opcode      15.48 usec

STATUS: Awaiting state command - userid      ________________ 11:14

 display  <LINE #>  disasmb  ______  ______  show  execute  ---ETC---
```

Fig. 2. *Analyzer trace list showing inverse assembler display with user-defined symbols.*

into a custom integrated circuit. This circuit, called the analysis controller, provides the necessary control for the trigger, store, count, sequence, and overview functions. By including the IMB interface on the analysis controller, tight coupling between these functions and the IMB functions is possible. An example of this coupling is the pervasive master enable function. Essentially, all control functions of the 64620S must monitor the IMB master enable function, suspending and resuming analysis activities in response to its changes.

© Copr. 1949-1998 Hewlett-Packard Co.

Besides the analysis controller, a floating-point Gray-code counter was developed, as well as a clock detection circuit and comparators for the clock and data probes. The custom counter generates the time or state count stored in the trace memory and acts as the interval counter for overview event time or state counts. The clock detection circuitry and probe comparators are responsible for capturing data on the input channels of the analyzer.

User Interface Design

The 64620S software provides a means for entering measurements and displaying the acquired data in useful formats. To handle the wide range of features that are available for effective state and software analysis, the user interface to the 64620S divides the various activities of setting up the analyzer into sections. Measurements are specified, for example, in the trace specification, while input channel formatting is done in the format specification. This division of tasks makes each section easier to understand and operate, and a wider variety of capabilities can be incorporated to improve the efficiency of performing each task.

In addition to separating the interface tasks, only a subset of the complete capabilities is normally available within each interface. This is most apparent in the trace specification where the measurement resources are controlled. If a default condition exists for a particular function (such as SEQUENCE or OVERVIEW), this condition is detected and that function does not appear in the interface display. The result is that the total capability of the analyzer need not be considered during the setup of most common measurement situations. For more demanding measurements, an extensive set of resources is activated. In this way, the user interface grows in a degree equal to the task at hand.

Symbolic Operation

In all input specifications and output displays, the 64620S software provides symbolic operation. Labels, such as the channel grouping Address, and symbols, such as STACK to represent the range of address values of the stack area, are user-definable. These labels and symbols are stored in a separate data structure in the analyzer. Also, the directed-syntax technique used in the 64000 System is extended to provide these labels and symbols on softkeys at the appropriate point in a command line. The association is made by grouping symbols together into a data structure referred to as a symbol map, and then defining a default map for each label. Thus, entering commands such as trigger on Address = range STACK can be done using only the softkeys.

Not only are the labels and symbols available on softkeys for entry of commands, but they are also used in the trace list to display acquired data in easily recognized forms. For each column label displayed in relative mode, a dynamic lookup in a symbol map is performed. Normally the default map is used, but any named symbol map can be specified. If a symbol corresponds to the data value, this symbol is placed in the list and an offset is added if the symbol represents a range of values. This eliminates the tedious task of mentally translating the data into a meaningful form.

The symbolic entry and output capabilities of the 64620S eliminates much of the detail work, very much like a high-level language in programming. It is not necessary to enter and interpret ones and zeros. This was deemed very important to an analyzer aimed at software analysis, and even in hardware analysis these techniques are useful.

Configurability of a General-Purpose Analyzer

An instrument is considered friendly if the user is comfortable working with it. With such instruments, users re-

Stan W. Bowlin

Stan Bowlin joined HP in 1979 as a development engineer with five years of work experience. He attended Colorado State University, receiving a BSEE degree in 1974. Born in Wichita, Kansas, he now lives in Colorado Springs, Colorado. He is married and enjoys outdoor activities such as bicycling and skiing.

Frederick J. Palmer

Rick Palmer received the BSEE, MSEE, and EE degrees from the Massachusetts Institute of Technology in 1977. With HP since 1977, he has contributed to the 1802 plug-in for the 1611A Logic Analyzer, the 64300A Analyzer for the 64000 System, and the 64620S Analyzer. Born in Detroit, Michigan, he recently transferred to HP's division in Boeblingen, West Germany as a marketing engineer. He lives in nearby Sindelfingen, is single, and plays guitar.

Richard A. Nygaard, Jr.

Rick Nygaard came to HP in 1977 after receiving a BEE degree from the Georgia Institute of Technology. Besides his work on the 64620S Analyzer, he has worked on firmware changes for the 1610B Analyzer. Born in Spokane, Washington, he now lives in Colorado Springs, Colorado. He is interested in woodworking and enjoys skiing, bicycling, backpacking, and playing softball.

Bryce S. Goodwin, Jr.

Bryce Goodwin earned a BS degree in electrical engineering and computer science at the University of Colorado in 1978 and joined HP shortly thereafter. He is a software designer for the 64620S Analyzer. Bryce lives in Colorado Springs, Colorado, where he was born. He is married, and has two children. Outside of work, he enjoys skiing (downhill and water), hunting, and fishing.

© Copr. 1949-1998 Hewlett-Packard Co.

quire relatively short learning periods before the instrument can be used. Instruments designed for very specific applications generally belong to this category. However, unless user configurability becomes a principal design goal, as in the case of the 64620S Analyzer, general-purpose instruments may fall short of these requirements.

Through an automatic configuration process, the general-purpose 64620S Analyzer appears to be specially designed for the process under test. In addition, the analyzer can be further configured to appear tailor-made for the program environment being monitored. This additional configuration process can occur as part of the automatic configuration or can be done later by the user.

If a CPU-specific interface module is used, automatic instrument configuration occurs upon entry to the 64620S. The interface probe module supplies an identification code to the analyzer, indicating the type of CPU being monitored. This identification code selects the proper configuration file. The subsequent automatic configuration process then adapts the analyzer's operation to the characteristics of the monitored CPU, creating the perception that the 64620S was designed specifically for that CPU.

Automatic instrument configuration is also provided through the general-purpose interface module. This device allows the user to design an interface module meeting the requirements of a specified CPU. A 16-position rotary switch, located on the general-purpose interface module, provides the 64620S with an identification code so that it can locate and automatically load the applicable, user-defined configuration file. Thus, the analyzer can be automatically configured for proprietary machines or other processors not supported by Hewlett-Packard.

Associated with the general-purpose interface module, the inverse assembly language (see box on page 21) provides for user-generated inverse assemblers. Inverse assemblers are specified by the automatically loaded configuration file, so that inverse assembly, as well as instrument configuration, may appear to be tailor-made.

General-purpose probes provide an alternative to the use of the general-purpose interface module or other HP standard interface modules. Although an identification code is not available to inform the analyzer of the type of CPU being monitored, modification of the default configuration file can still result in automatic instrument configuration.

Acknowledgments

Bill Martin served as project leader during the early phases, guiding much of the feature set selection, especially the overview measurements. The often thankless job of writing the hardware driver utilities went to John Martin, who contributed an especially intricate piece of firmware to load the range decoder circuit. Brad Yackle created the IAL structure. Electrical design for the general-purpose probes was done by Jim Dolan and Steve Peurifoy with Harry Short and Bob Self doing the mechanical design. And thanks to the Colorado Springs integrated circuit facility team who made the five custom circuits in the analyzer possible.

Steven R. Williams

Born in Urbana, Illinois, Steve Williams attended the nearby University of Illinois, earning a BS degree in electrical engineering in 1981. He then joined HP, worked on the development of the 64620S Analyzer, and now is a product marketing engineer for it. Steve has written several copyrighted computer programs and is deeply involved with church activities. He is single, lives in Colorado Springs, Colorado, and is interested in conducting music, playing trumpet, skiing (water, downhill, and cross-country), bicycling, hiking, and running.

A Modular Logic Timing Analyzer for the 64000 System

by Joel A. Zellmer, John E. Hanna, and David L. Neuder

A LOGIC TIMING ANALYZER asynchronously samples data flow in the system under test and is primarily used to troubleshoot hardware-related problems in digital circuitry. It is optimized for showing time relationships between digital signals, an area where oscilloscopes are often used. Timing analyzers, however, offer features not found in most oscilloscopes, making them especially useful in testing digital circuitry. The following characteristics of timing analyzers differentiate them from oscilloscopes:

- Two-level vertical resolution
- Single-shot recording of multichannel data
- Simultaneous display of up to 16 channels
- Display of data flow occurring before a trigger condition
- Triggering capabilities tuned to the multichannel digital environment.

© Copr. 1949-1998 Hewlett-Packard Co.

Other important features of timing analyzers include:

- Fine timing resolution
- Multiple modes of data acquisition, including high-resolution, missed data or glitch detection, and dual-threshold measurements
- Powerful and flexible triggering, including triggering from other digital analysis systems such as a synchronous state or software analyzer, or from an emulator
- Large memory
- Large number of input channels
- High-quality probing
- Ease of use (setting up and executing measurements and formatting of output).

The data acquisition modes of the new 64600S Timing Analyzer (Fig. 1) allow the user flexibility in troubleshooting. The high-resolution mode allows sampling at 400 MHz, giving excellent timing resolution often needed in examining timing margins, even on low-speed data buses. The memory depth of 8140 samples per channel gives a timing window 20 μs wide with 2.5-ns sample resolution. The dual-threshold mode, which displays three-level waveforms, simplifies troubleshooting such problems as bus conflicts, improper loading, slow rise times, and noise on signal lines. A glitch detection mode is useful when it is necessary to select slower sample rates to cover a long time window in a particular measurement, while not missing any short-duration activity occurring between samples. Glitches lasting only 3 ns can be displayed.

The triggering capabilities of the 64600S are designed to solve timing-related problems in multichannel logic environments. In addition, the analyzer uses a very easy and convenient operating interface to aid the new or occasional user in setting up measurements.

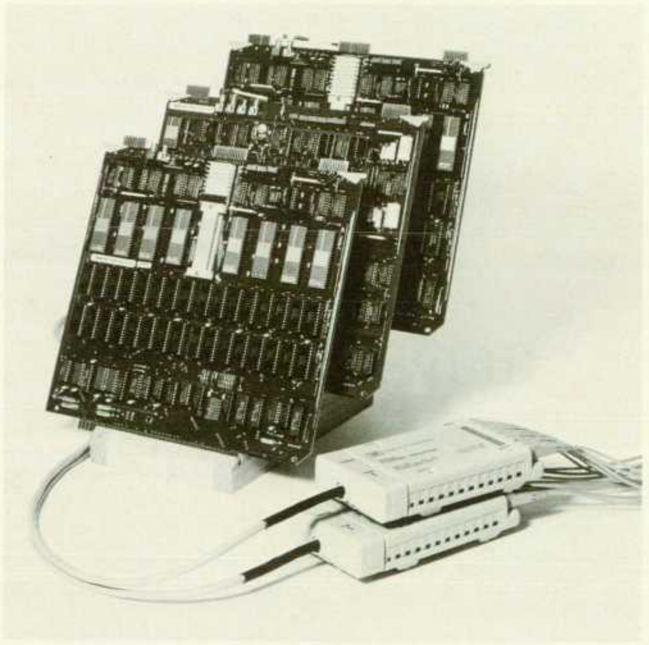


Fig. 1. *HP Model 64600S Logic Timing Analyzer adds powerful, high-resolution, asynchronous analysis to the 64000 Logic Development System. It has eight input channels and can be expanded to sixteen input channels. The 64600S can be added to either the 64100A or 64110A Development Station as part of a hard-disc-based cluster system or as a stand-alone analyzer in a flexible-disc-based station.*

There are many applications for the 64600S Timing Analyzer. It is a valuable tool for checking out new digital hardware and for troubleshooting faulty circuitry. To examine problems quickly, the 64600S can:

- Capture and display nonperiodic waveforms and single shot events
- Examine the time relationships between signals, setup and hold times, and other events.
- Detect unwanted transitions on signals (glitches)
- Detect fan-out problems, bad logic levels, and slow rise and fall times
- Detect conditions that last longer or shorter than some specified duration.

The timing system is modular, consisting of a control board and an acquisition board with accompanying 8-channel probe. One control board can drive one or two acquisition boards so that a single module can have 8 or 16 channels. Multiple timing or other modules can be connected through the 64000 System IMB (intermodule bus), allowing intermodule interaction. The timing system design allows the user to mix control and acquisition boards and probes without the need for hardware adjustments.

As part of the 64000 System, the 64600S provides other advantages. For example, because the 64600S is disc based, new postprocessing features can be added easily. Data files can be processed using a station's Pascal/64000 capability. Data measurements can be stored on flexible discs and brought to other systems for analysis. Detailed setups for particular measurements can be stored in configuration files, and then quickly reentered into the timing analyzer to be executed. The 64000's intermodule bus allows complex interaction with other modules such as emulators, state analyzers, or other timing analyzers. Using the terminal mode software, measurements can be performed at remote sites and the data transferred via RS-232-C/V.24 and modem interfaces to another unit at a central location.

Operator Interface

Before delving further into the measurement features of the analyzer, a discussion of the operator interface is important since this often determines the utility of an instrument. Any instrument that is easy to understand and use will be of more use and provide more data to the user. The 64600S's operator interface is designed to be a friendly interface by extensive use of directed-syntax and sentence-like commands, and display of only pertinent information. A directed-syntax structure prompts and directs the user through a command tree—freeing the user from having to remember keywords and key sequences. With directed syntax, the next level of valid keywords to complete a command is always displayed on the softkey labels. The softkeys eliminate the frustration of keying in an illegal key sequence, because they track only valid commands. A simple default execution of the 64600S allows the user to examine all signals input to its probes by specifying only two commands:

timing	Brings in the timing analysis software
execute	Display automatically changes to the timing diagram display

© Copr. 1949-1998 Hewlett-Packard Co.

The use of sentence-like commands brings clarity to the measurement setup and display of data. For example, to produce a timing diagram of a signal called SYNC, which has been connected to pod 1, input 0, the following commands are invoked:

timing	Brings in the timing analysis software
show format_specification	Moves to the format display
define SYNC pod_1_bit 0	Sets up the label SYNC
show trace_specification	Moves to the trace specification display
trigger on entering SYNC = 1	Sets up the trigger condition
execute	Display automatically changes to the timing diagram display
display SYNC	Sets up the timing diagram to display the signal labeled SYNC

The 64600S displays only pertinent information about the specifications that the user enters. As the user requests more complex measurements, the display specifications list the additional complexity.

The user interface is partitioned into four displays: format specification, trace specification, timing diagram, and trace list. In each of these displays the user can specify commands specific to that display. For example, in the format specification the user can specify labels for the probe inputs and enter threshold levels, while in the trace specification the trigger conditions and sample rate can be selected. In the timing diagram and trace list displays, commands specific to the measurement data display format are found. Also, in each of these displays the user can specify common operation commands to execute and halt a measurement. There is no need to go to a special display to run the 64600S Timing Analyzer. The user interface also allows measurement setups to be stored in and reloaded from files. Therefore, there is no need to remember an old configuration; instead, it can simply be brought back through the file handler of the 64000 System.

Acquisition Modes

The 64600S Timing Analyzer samples data on its probe inputs in four different modes: wide sample, dual-threshold, fast sample, and glitch capture. Each mode offers different views into the network under test. These choices are available within the same instrument, and all are under software control from the same displays (specifications). The consistent interface of the 64600S, independent of mode, enhances the user's ability to make the measurement and to interpret the data correctly.

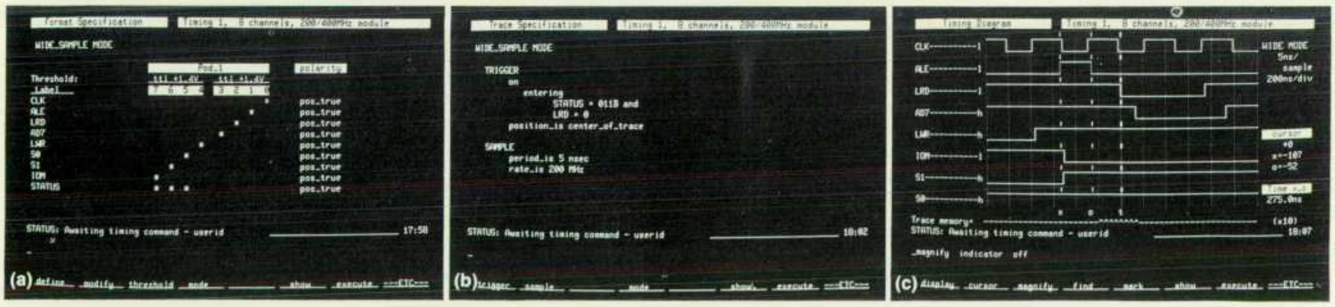
Wide Sample Mode

The most commonly used mode is wide sample, which gathers 4060 bits of data for each of eight inputs on each probe pod at sample rates from 2 Hz to 200 MHz (0.5-s to 5-ns periods). Depending on the analyzer option, there can be one or two probe pods and therefore 8 or 16 inputs. Within each probe pod, comparison thresholds can be set for two groups of four channels so that the user can, for example, probe ECL and TTL circuits simultaneously with one probe pod. Examples of a typical format specification, trace specification, and timing diagram for an 8-channel analyzer are shown in Fig. 2.

For a typical measurement, the user begins by defining labels to be associated with the probe pod inputs. These labels should be relevant to the names of the points probed. The labels shown on the left in Fig. 2a are mapped via the asterisks to a particular input or group of inputs. Thus, LWR is a label associated with pod 1, input 4, while STATUS is a multibit label associated with pod 1, inputs 5 through 7. Note that the multibit label STATUS is composed of three separate single-bit labels: IOM, S0, and S1. The user then has the choice of two ways of representing the input signals to be tested. For example, triggering on STATUS = 011 is the same as triggering on IOM = 0, S0 = 1, and S1 = 1. The comparison thresholds default to TTL levels as shown on the threshold line with a positive-true logic sense. These can be redefined by the user to positive- or negative-true logic values between +10V and −10V, respectively. The labels transfer automatically to all other specifications in the analyzer so that the user can define trigger conditions, display formats, and other parameters by using labels rather than, for example, pod_1_bit 0. This makes using the 64600S easier, faster and more accurate since the user works in terms associated with the user's system rather than in terms of the analyzer connected to it.

Dual-Threshold Mode

By pressing the mode softkey, the user is given a choice of


```
(a)
Format Specification     Timing 1,  8 channels, 200/400MHz module

WIDE_SAMPLE MODE

                         Pod_1                 polarity
Threshold:          ttl +1.4V   ttl +1.4V
 Label              7 6 5 4     3 2 1 0
CLK                                   *        pos_true
ALE                                 *          pos_true
LRD                               *            pos_true
AD7                             *              pos_true
LWR                         *                  pos_true
S0                        *                    pos_true
S1                      *                      pos_true
IOM                 *                          pos_true
STATUS              * * *                      pos_true

STATUS: Awaiting timing command - userid  ______________ 17:58

define  modify  threshold  mode  ____  show  execute  ---ETC---
```

```
(b)
Trace Specification     Timing 1,  8 channels, 200/400MHz module

WIDE_SAMPLE MODE

  TRIGGER
      on
         entering
               STATUS = 011B and
               LRD = 0
      position_is center_of_trace

  SAMPLE
      period_is 5 nsec
      rate_is 200 MHz

STATUS: Awaiting timing command - userid  ______________ 18:02

trigger  sample  ____  mode  ____  show  execute  ---ETC---
```

```
(c)
Timing Diagram     Timing 1,  8 channels, 200/400MHz module

CLK-----------l                                  WIDE MODE
ALE-----------l                                  5ns/
LRD-----------l                                  sample
AD7-----------h                                  200ns/div
LWR-----------h                                  cursor
IOM-----------l                                  +0
S1------------h                                  x=-107
S0------------h                                  o=-52
                                                 time x-o
                                                 275.0ns
Trace memory-                                    (x10)
STATUS: Awaiting timing command - userid  ______________ 18:07

_magnify indicator off

display  cursor  magnify  find  mark  show  execute  ---ETC---
```

Fig. 2. *Typical (a) format specification, (b) trace specification, and (c) timing diagram displays for an eight-channel 64600S Analyzer in the wide sample mode.*

© Copr. 1949-1998 Hewlett-Packard Co.

the remaining three modes. Selecting the dual-threshold mode provides more voltage resolution than is available with just a simple timing analyzer measurement (Fig. 3b). On a TTL bus, for example, conflicts or excessive loading can cause a line to cross the nominal threshold ($V_{th\ typ}$= 1.4V) and be detected as a transition by a simple analyzer, but may not produce a valid logic high (>2.0 volts) or low (<0.8 volts). Fig. 3a illustrates some of these conditions. Similarly with ECL circuits, weak pulldowns can cause a poor logic low.

These conditions can be found using the dual-threshold mode in which each input is compared to two thresholds, $V_{ih\ min}$ and $V_{il\ max}$, at sample rates from 2 Hz to 200 MHz. A three-level display (Fig. 3c) shows the time spent between thresholds and any incomplete transitions. Fig. 4 shows the specifications and displays of a typical dual-threshold measurement.

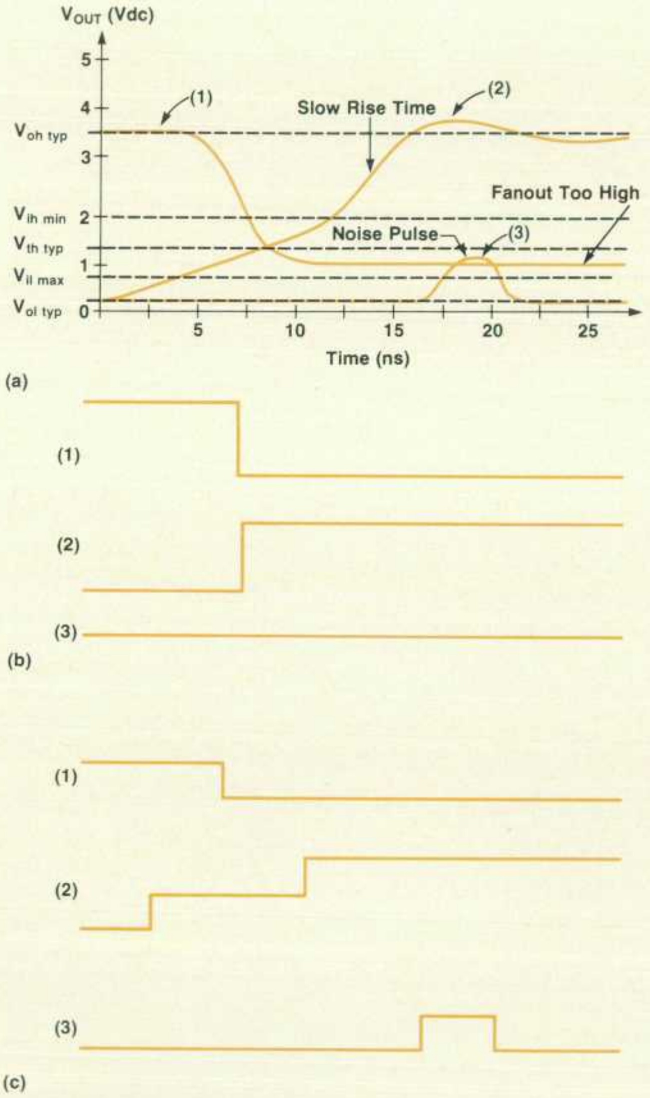


Fig. 3. *Use of dual-threshold mode to analyze LSTTL waveforms (a) which are degraded by high fanout (curve 1), slow rise time (curve 2), or noise (curve 3). (b) Detection of curves 1, 2, and 3 by a simple timing analyzer with a single threshold set to $V_{th\ typ}$. (c) By using the 64600S Analyzer's dual-threshold mode, curves 1, 2, and 3 can be detected as shown.*

In the dual-threshold mode, only four inputs are active on each pod because twice the information must be stored in memory for each input channel. The format specification shown in Fig. 4 illustrates the use of dual-threshold mode within the previous measurement setup.

Fast Sample Mode

Should more time resolution be required than available using the 5-ns sample period in the wide sample mode, the fast sample mode using a 2.5-ns sample period (400 MHz sample rate) can be selected. This is accomplished by allocating two samplers to each input with a 2.5-ns time separation between the samplers, and results in an 8K memory. This restricts the number of inputs that can be sampled to those in the lower half of the probe, but this sample rate is typically needed to compare data on a small number of channels.

Glitch Capture Mode

Despite the 4K memory of the 64600S, which provides 20 μs of storage at 200 MHz, there are instances when very long time spans must be observed and the user still wants to be aware of the occurrence of even brief events. The glitch capture mode monitors edges on the incoming data as well as sampling the data from 2 Hz to 100 MHz. If more than one edge occurs between two adjacent sample times, it records this event in a separate memory as a glitch. The presence of this glitch can alert the user to examine the data in this region more closely by using one of the other modes. Since separate circuitry and memory are used for glitch detection and recording, glitches do not distort normal edge locations, and glitches occurring close to or on edges are captured and displayed.

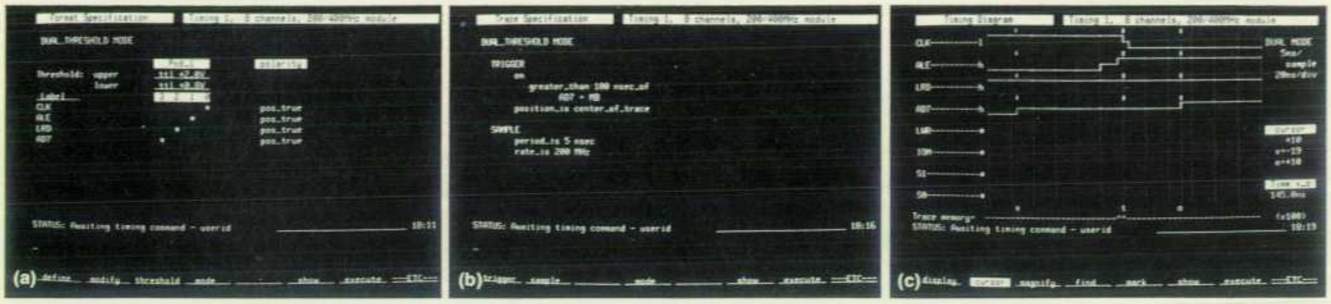
Triggering

The user can choose any of five types of triggering:
- Triggering upon entering a pattern
- Triggering upon leaving a pattern
- Triggering on greater than a specified duration of a pattern (including a middle level in the dual-threshold mode)
- Triggering on less than a specified duration of a pattern (including a middle level in the dual-threshold mode)
- Triggering on combinations of patterns and glitches.

In this discussion, "pattern" indicates the value of an ANDed group of inputs, or the complement of that value.

The five types of triggering qualify the trigger in ways not possible with a simple occurrence trigger. Triggering only on entering or leaving a pattern means that the analyzer will not trigger if the pattern is present when the analyzer is started. On the other hand, triggering on greater than or less than some time duration of the pattern produces a trigger whenever the qualified duration is reached. The duration trigger types are especially useful because they allow the user to set the duration to a value larger or smaller than any duration expected, and trigger on that event if it occurs. To illustrate this capability, refer to Fig. 5. Here it is possible to trigger the analyzer when the duration from REQuest to ACKnowledge is either too long or too short. An example trigger command might be trigger on greater_than 1 usec_of

© Copr. 1949-1998 Hewlett-Packard Co.



```
Format Specification        Timing 1,  8 channels, 200/400MHz module

DUAL_THRESHOLD MODE

                              Pod_1            polarity
Threshold:  upper           ttl +2.0V
            lower           ttl +0.8V
 Label                      [illegible]
CLK                                 *          pos_true
ALE                               *            pos_true
LRD                            *               pos_true
AD7                          *                 pos_true


STATUS: Awaiting timing command - userid  _______________  10:11

(a) define  modify  threshold  mode  ____  show  execute  ---ETC---
```

```
Trace Specification         Timing 1,  8 channels, 200/400MHz module

DUAL_THRESHOLD MODE

  TRIGGER
     on
        greater_than 100 nsec_of
             AD7 = HB
     position_is center_of_trace

  SAMPLE
     period_is 5 nsec
     rate_is 200 MHz


STATUS: Awaiting timing command - userid  _______________  10:16

(b) trigger  sample  ____  mode  ____  show  execute  ---ETC---
```

```
Timing Diagram              Timing 1,  8 channels, 200/400MHz module

CLK---------1                                        DUAL MODE
ALE---------h                                             5ns/
LRD---------h                                           sample
AD7---------h                                         20ns/div
LWR---------*                                           cursor
IOM---------*                                             x10
S1----------*                                           x--19
S0----------*                                           o--10
                                                        time x-o
                                                        145.0ns
Trace memory-                                           (x100)
STATUS: Awaiting timing command - userid  _______________  10:19

(c) display  cursor  magnify  find  mark  show  execute  ---ETC---
```

Fig. 4. *Typical dual-threshold-mode (a) format specification, (b) trace specification, and (c) timing diagram displays for the 64600S.*

REQ = 1 and ACK = 0.

Additional cross-pod triggering in a 16-channel 64600S Timing Analyzer allows conditional OR triggering, conditional duration triggering, and sequential triggering. An example of conditional duration triggering can be shown by again referring to Fig. 5. It is desirable to trigger on the occurrence of the acknowledge signal going high without a request signal. Note that triggering on ACK = 1 and REQ = 0 happens at the trailing edge of a normal handshake. However, triggering on ACK going high while REQ is low exactly catches the faulty condition. An example trigger command might be trigger on entering ACK = 1 when_greater_than 1 usec_of REQ = 0.

A programmable time delay is also available. This allows delaying trigger events (as described above) up to 32 million clock cycles. The delayed trigger point can be positioned anywhere in the acquisition memory: start, middle, end, or a definable percentage of the memory before the delayed trigger point.

Intermodule Bus Interaction

The 64600S Timing Analyzer, through the 64000 System's intermodule bus (IMB), can arm or trigger other modules, or can be armed, triggered, or delayed from other modules in a 64000 Development Station.

As an example of intermodule triggering, a state analyzer in the 64000 Station could be tracing a long sequence of events, and when this sequence is satisfied, arm the 64600S Timing Analyzer. The timing analyzer then triggers when it satisfies its own internal trigger conditions.

An additional autorestart function is also available when using the 64600S with the IMB. This is useful in correlating timing phenomena with subsequent faulty state flow. The timing analyzer can look for a pattern, trigger, complete its trace, and then wait for a state analyzer to tell it what to do via the IMB. If the state analyzer observes normal state flow, it can tell the timing analyzer to reset and start over again. If after the next timing measurement the state analyzer observes faulty state flow, further restarts would be inhibited and the timing trace containing the data that produced the faulty state flow can be observed.

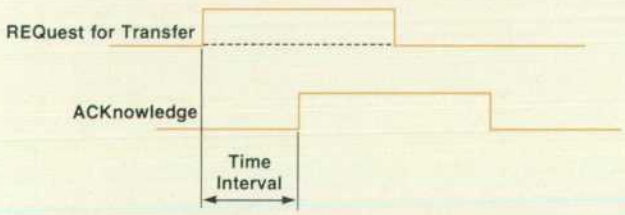


Fig. 5. *The different triggering modes of the 64600S Analyzer can be used to examine the relationships between a system's REQ and ACK handshake waveforms.*

Displaying Data

The 64600S Timing Analyzer can display measurement data in the form of either a timing diagram or a trace list. The timing diagram presents up to sixteen channels of measurement data. The channel ordering and spacing can be set up using labels that the user enters or default channel numbers. By selecting appropriate labels, the user can present the measurement data in a form that gives a clear description of what has been measured (see Fig. 2c and Fig. 4c).

Magnification, time cursors, and memory indicators are important features for study of the timing diagram. Magnification along the time axis allows the fine detail of a portion of the timing diagram to be expanded. Three powers of magnification are allowed: ×1, ×10, and ×100. In ×1, the measurement data (4060 samples) is compressed into 203 display characters by a 20:1 compression routine. Multiple transitions in each 20-sample group are indicated with a multiple transition character (glitch symbol). Thus, the user can use the ×1 magnification to find regions of activity and use the other magnifications (×10, ×100) to see more detail of each region of activity. This is significantly different from many analyzers, which compress multiple transitions into a single transition and make it difficult to distinguish regions of activity from simple transitions.

Multiple time cursors (x, o in Fig. 2c and Fig. 4c) are available in the 64600S to measure durations of events or intervals between events. Graticules and time-per-division information provide the user with another reference to the amount of time that is shown. Also, the position of the indicator (/\/\/\) under the timing diagram shows the portion of the trace that is currently being observed.

Hard copy of the timing diagram and the trace list are available to record the measurement data when needed.

Probing

The 64604A Timing Probe consists of a cable connected to the acquisition board in a 64000 mainframe, a detachable pod housing a hybrid circuit containing the active comparator, and eight detachable coaxial probe inputs similar to oscilloscope probes. As a result, all the accessories for

© Copr. 1949-1998 Hewlett-Packard Co.

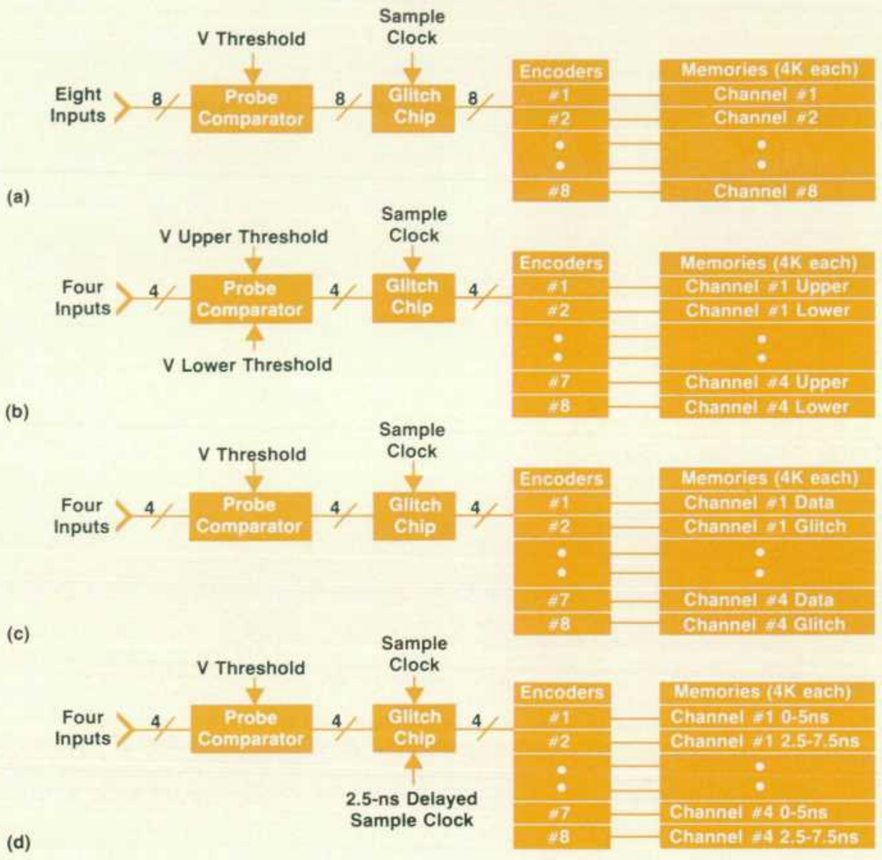


Fig. 6. *Data flow for the 64600S's four acquisition modes. (a) Wide sample. (b) Dual-threshold. (c) Glitch capture. (d) Fast sample.*

HP's 10017A Series Oscilloscope Probes (e.g., grabbers and clips) can be used with the 64604A. A new 20-pin dual-in-line package clip, the 10211A, has also been developed. This accessory allows easy connection to most 0.3, 0.4, 0.6, and 0.9-inch-wide dual-in-line IC packages. The 10211A is also stackable, end to end, to allow probing all the pins of 40-lead or 60-lead packages.

The probe inputs are compensated to provide the comparator in the pod a high fidelity reproduction of the signal at the probe tip, avoiding the ringing and resulting uncertainty associated with open-wire probes and fast edges. The input impedance at the tip is 100 kΩ in parallel with 6 pF.

The probe has two comparison thresholds, one for channels 0 through 3 and one for channels 4 through 7. The thresholds are set by software from −10V to 10V in 0.1V steps. The dynamic range of the probe is specified as ±10V. Exceeding this value, as might happen with CMOS circuits using 15V supplies, causes less than 1 ns of additional skew as the input clamps are activated, and essentially no change in loading.

Hardware Organization

The data acquisition path of the 64600S uses three custom, bipolar EFL integrated circuits: an input comparator chip, a glitch chip, and an array of encoder chips. Fig. 6 is a block diagram of the data acquisition path showing how these chips are used and the effect that changing the acquisition mode has on data flow and memory allocation.

The custom 8-channel comparator chip receives the input data through the passive RC dividers. Having all eight comparators on the same chip keeps the interchannel skew low without requiring delay adjustments. In the dual-threshold mode, the same input signal is sent to upper-threshold and lower-threshold comparators. An output data stream is generated for each threshold level. The comparator outputs drive complementary ECL signals down twisted-pair transmission lines to the glitch chip.

In the glitch chip, input data and glitches are sampled and basic trigger comparisons are made. The maximum sample rate of this chip is 200 MHz. In the fast sample mode, the data outputs consist of two 200-MHz data streams per channel, one delayed by 2.5 ns with respect to the other. Again, because the data from all eight input channels is sampled on one chip, the delays are inherently well matched. Special care was still required to adjust the input aperture for both positive and negative data transitions to be at the same point with respect to the sample clock.

Each of the eight outputs of the glitch chip is fed to an encoder chip to do a serial-to-parallel data conversion. This chip slows down the data rate to the TTL memories by a factor of sixteen. In other words, it effectively changes a 12.5-MHz memory to a 200-MHz memory. A block diagram of this encoder chip is shown in Fig. 7. The TTL data outputs of this chip are fully buffered, the data remaining constant at the memory inputs for a full write cycle time.

The above acquisition functions reside on the data acquisition board. The control board generates the sample clocks, processes the raw pattern trigger information for time duration specifications, and controls the measurement by starting and stopping the data acquisition cycle in ac-

© Copr. 1949-1998 Hewlett-Packard Co.

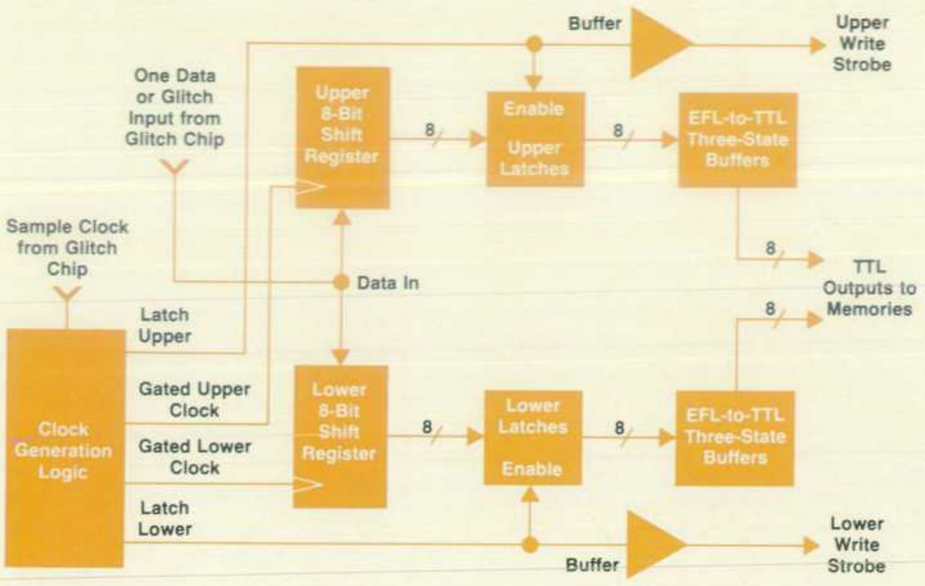


Fig. 7. *Encoder chip block diagram.*

cordance with the amount or pretrigger information desired. This board also generates the timing diagram.

The time-duration triggering organization is shown in Fig. 8. The three basic duration modes are transition (entering, leaving, glitch), time duration for greater than a preset value, and time duration for less than a preset value. These modes are generated by an edge detector coupled to circuitry that tests to see if the input pattern lasts longer than the preset value. The edge detector, together with the selectable inversion, generates triggers when the specified input pattern enters or leaves. The width-greater-than circuitry generates triggers when the duration for a pattern or the complement of that pattern is wider than a specified time. Outputs of both of these detectors are used to generate triggers for a less-than-time duration. Here the presence of an edge denoting that the trigger condition is going false is ANDed with the status of the width-greater-than detector. If the width signal is false, this implies that the pattern was narrower than the width specification.

Resolution

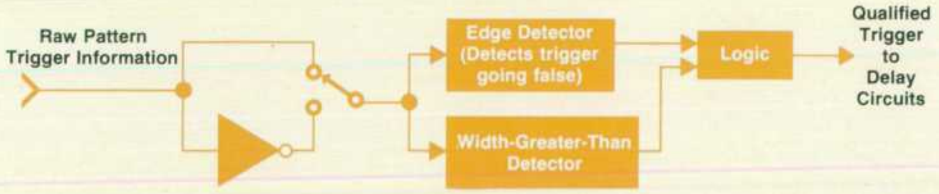
It is important to understand what determines the timing resolution of a timing analyzer to properly interpret the data it shows the user. Fig. 9a shows an example of input data and the corresponding sampled data information. Note here that different input data can result in the same displayed information, because the data sampler only looks at the incoming data at sample times. The resolution between edges on the timing diagram is limited to one sample period. Another problem, that of skew, or differences in delay between edges on the same or different channels, also strongly affects the available resolution. For example, if the timing analyzer internally delays the data from channel 1 of Fig. 9b by 1 ns longer than the data from channel 2, the edge resolution now becomes ±(sample period + 1 ns). Skew can also occur on a single data channel if the delay to the sampling aperture is different for a positive data transition and a negative data transition. This type of skew can stretch or compress pulse width.

In production, skew is measured using a routine present in the 64600S self-test software. This procedure uses a statistical beat-frequency approach to measure the skew of the acquisition circuitry. Consider the following conditions: the sample clock is operating at 200 MHz (5-ns sample period), the input data rate is 10.01 MHz (99.9-ns period), and the memory of the analyzer can store 4060 samples. This means that every time a new data edge appears at the data sampler, its time location with respect to the sample clock has been shifted by 100 ps, or for every 50 data edges, one complete sweep of edges will occur through the sample period (one beat). In this example, 4000 bits of memory would then hold approximately four beats as shown below:

$$(4000 \times 5\text{ ns})/(50 \times 99.9\text{ ns}) = 4.004\text{ beats}$$

To calculate the skew between a reference edge of particular polarity on one channel to an edge of either polarity on another channel, the following formula is used.

$$\frac{(\text{number of misaligned edges})}{(\text{number of edges compared})} \times \text{sample period} = \text{skew}$$



Raw Pattern Trigger Information
Edge Detector (Detects trigger going false)
Width-Greater-Than Detector
Logic
Qualified Trigger to Delay Circuits

Fig. 8. *Organization of time-duration triggering circuitry.*

© Copr. 1949-1998 Hewlett-Packard Co.

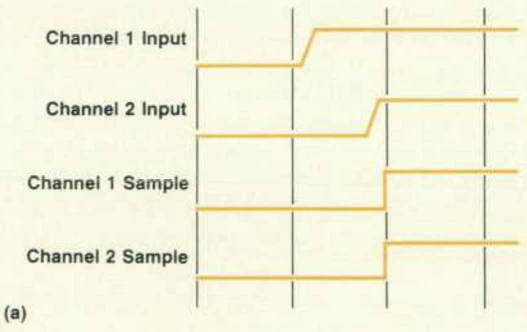
Software Solutions to Displaying Data

Displaying sixteen channels of data with 4060 samples per channel in a short amount of time can be a difficult task. A minor hardware change and a special software algorithm were developed to reduce this time.

The display memory consists of 240 characters per channel. Each character is composed of a two-bit pattern, GD, where G indicates if the multiple transition symbol (glitch) is to be displayed and D indicates if data is high or low. Each two-bit pattern is separately addressable. The acquisition memory consists of 4096 samples of data per channel with sixteen samples from each channel packed into an addressable 16-bit word. Originally, the data was packed in the form D0, D1, D2,..., D15, but trying to convert this into a format (G0D0, G1D1, G2D2,...) acceptable for the display memory was a problem. The solution is to modify the output of the acquisition memory so the packed word is in the form D8, D0, D9, D1, D10, D2, ..., D15, D7. Now, by simply masking the pattern with a hexadecimal 5555 mask, and shifting and masking again, the two words 0, D0, 0, D1,...,0, D7 and 0, D8, 0, D9,..., 0, D15 are produced. The glitch information is processed the same way and merged with the data providing the proper format for the display memory (G0, D0, G1, D1, G2, D2,...).

Acknowledgments

We owe thanks to many people for their help in the definition and design of this product. Kyle Black, Larry McBride, Steve Ratner, Don Zimmer, and many others in the IC department were instrumental in helping us get the critical integrated circuits completed. Dick Morrell handled the initial IC design and layout. Harry Short and Bob Self contributed to the mechanical design of the probe. Mike Sweezey wrote the software package for driving the hardware, and John Shada aided in the initial user interface design. Joe Hawk developed the performance verification software, and also aided in the final user interface. Steve Peurifoy did the electrical design for the probe. Dave Baumgarten did the mechanical design for the IC chip probing accessory. Larry Anderson helped in easing the product into production.



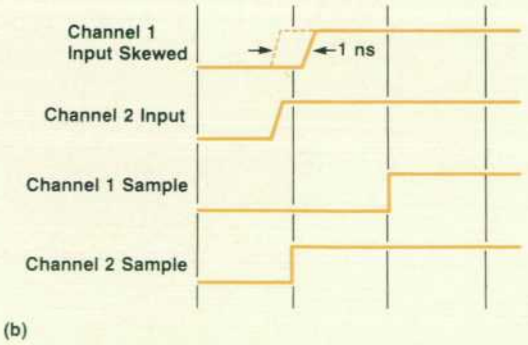


Fig. 9. *Example of timing resolution for input data and sampled data without (a) and with (b) skew.*

Reference

1. J.A. Scharrer, R.G. Wickliff, and W.A. Martin, "Interactive Logic State and Timing Analyses for Tracking Down Problems in Digital Systems," Hewlett-Packard Journal, Vol. 29, no. 6, February 1978.

David L. Neuder

Born in Wyandotte, Michigan, Dave Neuder attended Michigan State University and received a BSEE degree in 1977 and an MSEE degree in 1979. He joined HP in 1979 as an R&D engineer and worked on the hardware and software for the 64600S. Dave is a member of the IEEE and involved with working with the elderly in his community. He is single, lives in Colorado Springs, Colorado, and enjoys skiing and mountain climbing.

John E. Hanna

Ted Hanna received a BASc degree (1964) and an MASc degree (1970) in electrical engineering from the University of Waterloo, Ontario, Canada. After several years designing ICs for consumer electronic products, he joined HP in 1977. Ted worked on the acquisition hardware for the 64600S Analyzer. His contributions have resulted in four papers on IC design and applications and five patents related to IC structures and circuits. Born in Guelph, Ontario, Canada, he now lives in Black Forest, Colorado with his wife and an assortment of dogs, cats, and chickens. He has five children, coordinates the Black Forest van pool, and is interested in photography.

Joel A. Zellmer

With HP since 1966, Joel Zellmer's contributions have included leading the 1120A Active Probe and 64600S Analyzer projects and designing parts of the 8080A and 8082A Pulse Generators. His work has resulted in three earlier HP Journal articles. He earned a BS degree in 1964 and an MSEE degree in 1966 at the University of Minnesota. He was born in St. Paul, Minnesota and now lives in Colorado Springs, Colorado. He is married, has two children, and enjoys skiing, sailing, and camping in the Rocky Mountains.

© Copr. 1949-1998 Hewlett-Packard Co.

Emulators for 16-Bit Microprocessors

by David B. Richey and John P. Romano

THE ADVANTAGES OFFERED by microprocessors have resulted in their use in a large portion of today's electronic designs. Because of their versatility and complexity, microprocessors frequently create problems for the designer. Microprocessors reside in their systems, veiled behind their protective plastic or ceramic packages, so that their inner activities are invisible. Their control mechanisms are untouchable. The task of working with these components, therefore, is often greatly aided by removing the microprocessor from its socket and inserting an emulator in its place. An emulator provides a window into the inner operation of a microprocessor and simulates its activity, giving designers the feedback and control necessary for development work. The feature set of a typical emulator includes:

- Loading, displaying, and modifying memory
- Displaying or modifying I/O space as appropriate
- Displaying or modifying processor resources such as working registers, DMA registers, and counters
- Starting or stopping execution
- Various documentation-related capabilities, like listing the above information to disc files or a printer
- Display and use of symbols.

With such features, the use of in-circuit emulation to design, test, and service microprocessor-based products has become accepted as a productive technique, and has contributed to the widespread use of such products.

As the use of microprocessors became more commonplace, the desire for more capability led to 16-bit microprocessor designs. In addition to having wider data and sometimes wider address buses, 16-bit microprocessors have much greater complexity than their 8-bit predecessors. These changes in processor technology have required corresponding changes in emulation philosophy and hardware.

The 64000 Logic Development System's first-generation emulators[1] were designed to aid the development of an 8-bit microprocessor-based system. These software and hardware tools gave designers the power to complete a complex new product design efficiently, handling all phases of the design cycle from the early breadboard to the final system software/hardware integration and test.

HP's second-generation emulators provide support for a variety of new 16-bit microprocessors. The difference in complexity between the 8-bit and 16-bit microprocessors demanded that we design new hardware and software to support our earlier 8-bit implementation. This approach gave us the opportunity to expand the previous feature set, implementing new features and adding breadth and flexibility to existing ones.

Emulator Hardware Design

The 16-bit processors increase the need for an emulator to work symbolically with compilers, to be flexible enough to accommodate the diverse systems in which 16-bit processors are used, and to be included in cross-coupled measurements with other emulators or other instruments. Naturally, the new 64000 16-bit emulators use the 64000's directed-syntax user interface, which includes the capability to create command macros using command files.

The hardware design was simplified by basing it on the expandable emulation bus architecture originally used for HP's earlier 8-bit 8080/8085, 6800, and Z80 emulators. The emulation bus connects a set of standard cards that include emulation control boards, memory control boards, emulation memory board(s), and a state analyzer board. The emulation control board controls execution. The memory control board contains an address mapper to partition the address range between user memory and emulation memory and to apply type and protection attributes. Emulation memory boards are used for prototyping. The emulation bus is entirely separate from the 64000 host processor bus. This avoids interference with an emulator when the host processor is conducting 64000 System activity. Multiple emulators in one 64100A or 64110A Development Station can operate independently since each emulator uses a separate emulation bus. Separating the buses also makes it possible for the host processor to set up an emulator in a watchdog measurement, which continues while the host moves on to operate, for example, another emulator, an external analyzer, or an edit session.

Wider Addresses

The emulation bus is universal and expandable, but microprocessor and memory technology has gone beyond the original planning for several of the standard boards. The first emulation bus specified 24 address and 16 data lines, without extensions. During the original emulation design it was assumed that a microprocessor system requiring even that many address lines would be several years away. The original 64300A internal analyzer monitored only 16 address lines, and the original 64151A memory controller handled only 20 address lines. Emulation memory supported 128K bytes with 1K-byte resolution. Then the 68000 microprocessor with a 16-megabyte addressing range and the Z8001 microprocessor with an 8-megabyte segmented range appeared. A second-generation analyzer and memory controller are now necessary.

Internal Analysis

A new internal analyzer, the 64302A, is designed to cover all 24 address lines. In addition to extended address, the analyzer has an expanded number of IMB (intermodule bus) functions. This enhances the cross-coupled measurement capability that is so important when designing complex 16-bit systems with memory management or multiprocessing. A problem was encountered when tracing the data flow of 16-bit microprocessors. These processors can transfer

© Copr. 1949-1998 Hewlett-Packard Co.

data either as a byte or as a word. When transferring a byte, it is not usually known whether the byte will appear as an upper or lower byte on the bus. The emulation bus provides word, lower-byte, and upper-byte status to the analyzer, but that does not help the situation. To solve this problem, the 64302A's data inputs can function as two independent bytes. Measurements can be made by specifying the data redundantly in both bytes and specifying a byte transfer to set the trigger condition.

Memory Control

A new memory controller, the 64155A, works with the full 24-bit address bus. It supports up to one megabyte of emulation memory, and has two mapping resolutions: 256 bytes and 4K bytes. It is used with the new memory boards that provide up to 128K bytes of static RAM on a single board. The 64155A is also compatible with existing 64152B, 64153B, and 64154B static RAM boards. A significant portion of a user's investment in emulation is in the emulation memory subsystem. Expanding for the 16-bit emulators introduced a new problem—implementing a dual-port memory scheme. New microprocessors use a very high percentage of their available bus bandwidth. Therefore, we implemented a new mechanism for the memory controller which pauses the emulation processor's memory activity whenever the 64000 host processor accesses emulation memory. The new memory controller has a transparent dual-port mode, but the 8086/88, Z8000, and 68000 emulators need a level of performance that requires the pause mode. Of course, a user can select a real-time running option to ensure that these pauses do not disrupt the operation of the target system.

Emulator Transparency

An important issue for emulator designers is transparency—the ability of an emulator to perform in a target system exactly like a microprocessor. Transparency has four aspects: electrical, timing, resource, and functional. Electrical transparency encompasses factors such as input and output loading and propagation delays. Typically, an emulator uses a higher-speed microprocessor and LSTTL buffering. This combination closely approximates the microprocessor manufacturer's propagation specifications. For emulators of NMOS microprocessors, exact reproduction of a processor's loading characteristics was given a lower design priority. Our goal has been to use one LSTTL load per signal unless this becomes excessive. This strategy has seemed appropriate since most NMOS microprocessor systems use TTL circuitry.

Timing transparency is commonly a measure of an emulator's ability to run at the maximum rated speed of the microprocessor. Therefore, the designer of an emulator must anticipate a processor's fastest mature speed and design to that performance. A second consideration is whether or not an emulator imposes wait states. Since real-time execution is important to users, the imposition of wait states or other timing aberrations is strongly avoided.

Resource transparency refers to restrictions on microprocessor resources or features imposed by an emulator design tradeoff. For example, a given emulator may restrict the use of a certain address range or an interrupt. These are restrictions a user would not have to make if the actual microprocessor were plugged into the user's system.

Functional transparency reflects the ability of an emulator to execute instructions, perform bus activity, and respond to asynchronous inputs in precisely the same manner as the microprocessor. This area is HP's first design priority and is usually the most difficult to achieve because microprocessors are largely undefined devices. Microprocessor vendors specify instruction sets, bus timing, and pin definitions, but rarely declare functional interactions. Knowledge of these interactions may not be necessary for a system designer, but an understanding of these functions can be the deciding factor as to whether an emulator does or does not work in the target system.

The high level of complexity in these 16-bit processors caused us to rethink the approach we had been using for the 8-bit emulators. We had been maximizing resource transparency and keeping the user's required level of knowledge about the details of the emulated processor to a minimum. To accomplish this, the 8-bit emulators have an alternate address space containing memory called background. User programs are executed from foreground memory until a breakpoint is encountered, then the emulator moves into the background memory where it executes the background monitor. This monitor dumps the processor's registers into memory and performs other duties. This monitor is totally resource transparent because it doesn't occupy any of the user's address range. Emulation software loads the background monitor into background memory and maintains proper operation.

This approach is not as satisfactory for 16-bit microprocessors because functional transparency would be threatened by the increased complexity of such devices. For example, the 8086/88 and 68000 processors prefetch instructions. Multimode interaction occurs between asynchronous inputs such as HALT and BERR on the 68000 microprocessor and STOP and BUSREQ on the Z8000 microprocessor. The transition from foreground to background memory becomes more difficult. It was clear that the misplacement or rearrangement of one bus cycle while transitioning to background would invite trouble. Consequently, for 16-bit processors, the emulation monitor is placed in memory along with the user's programs.

This approach would be unacceptable for an 8-bit emulator because it would affect resource transparency. That is, using 1K bytes of address space for an emulation monitor would be a major intrusion for an 8-bit 8049 emulator because the 8049 microprocessor has a program space of only 2K bytes. The same emulation monitor does not intrude so overwhelmingly on the one-megabyte range of a 16-bit 8086 microprocessor and thus does not adversely affect resource transparency. This is particularly true in the 64000 System, given that the monitor can be placed anywhere in the user's program space.

For the new 16-bit emulators, the emulation monitor can be placed anywhere in the microprocessor's address range. Exact placement is determined automatically by the emulator. As part of the integrated 64000 System, the emulation software accesses the linked symbol table to determine the emulation monitor's location.

© Copr. 1949-1998 Hewlett-Packard Co.

Flexibility

Flexibility is another key issue. It is anticipated that the 16-bit emulators will have to operate in conjunction with memory management ICs, operating systems, and other complex environments. This dictated a design approach that allows a user to customize the emulator to whatever hardware is used. One possibility was to use a background monitor that could be modified by users. This was discarded because the background monitor is usually a complex program. The complexity is caused by the peculiar boundary conditions that can occur during the transition from foreground to background. A pending software or hardware interrupt could require the background monitor to do gymnastics to preserve the pretransition environment and to unravel whatever activity occurred.

The foreground emulation monitor avoids that complex interaction. It is distributed in source form along with the other emulation software. Users are encouraged to modify this program to accommodate specific requirements of the target system, even to pare it down in size when necessary. Numerous error checks and status messages can be transferred to the 64000 System's status line. These features make the monitor as friendly as possible for first-time users, but can be deleted to create a smaller monitor for experienced users. The foreground monitor also comes to the rescue when an emulator is used in a multitasking system where the focus is not on controlling the microprocessor, but on controlling the processes. Via the emulation monitor, the emulator can be linked to an operating system, allowing the emulator's run and breakpoint features to activate and deactivate tasks.

As another example, consider the need for performing both word and byte transfers. Since the fundamental addressing mode is a byte address for these processors, the emulation monitor performs all memory transfers as byte transfers. However, it is common for emulator users to design hardware that accepts only word accesses. Once again, by simply modifying the emulation monitor, address ranges can be specified for word transfers, and elsewhere, byte transfers can be used.

Once the design decision to go with the foreground approach was made, it became clear that a common emulation control board could accommodate all of the 64000's 16-bit emulators. As a result, the 64271A board was designed. It contains interface circuitry to make the emulation pod signals compatible with the emulation bus, creates a port through which the 64000 Station can program pod configuration registers, provides a fast address mapper for buffer control, and transfers various control and status signals between the station and the pod.

Special Considerations for Coprocessors

Coprocessor support required some special hardware considerations for the emulator pods. The 8087, coprocessor to the 8086, monitors instruction fetching and execution. Usually, when accessing emulation memory resources, the emulator places the data bus into a high-impedance state. To allow coprocessors to monitor emulation memory activity, a special mode is required for the 8086 emulator. As needed, the emulation memory cycles can be driven out on the microprocessor data pins. To avoid bus contention problems, short plug-in leads are added to bring out mapper and emulation memory ready signals for use in the target system. Another problem arose from the common practice of using special blocks of memory for pointers or interrupt vectors for the microprocessor and its coprocessor. Vectors can reside very close together in memory, which can restrict the use of emulation memory. To overcome this problem, all of the 64000's 16-bit emulators can be set up to become "memory emulators" during coprocessor or other DMA-type cycles. During such times, the emulator generates emulation memory strobes from signals applied to the memory strobe pins of the inactive processor.

One of the more subtle changes made in response to field inputs from users happened when redesigning the 64151A to become the 64155A memory control board. A standard feature of the memory controllers detects write cycles to address ranges designated to be ROM. Such ranges are write-protected when they are mapped to emulation memory. But, regardless of whether emulation or user memory is specified, a breakpoint is generated when a write is performed to ROM. Many 8-bit emulator users like to use these ROM areas for other purposes, such as a write-only I/O space. The new 64155A memory controller allows this; it can be configured to generate or not generate write-to-ROM breakpoints.

Emulator Software Design

The following discussion is restricted to a description of the software features that are additions to or changes from the first-generation emulators.[1] Unaffected features are not discussed. Emulation features changed or added to handle 16-bit microprocessors include emulation session entry, configuration, general control, symbolic interface, command file execution and control, memory interface, I/O interface, software breakpoint, and analysis.

Emulation Session Entry

There are two possible entry points to emulation, depending upon what hardware modules are present in the development station's card cage. If there is only one emulation card set (one emulator control board with pod, a memory controller board, and an optional 300 or 302 analysis board), then entry is directly from the development station via the emulate softkey. At this point the user has the following options:

- Begin a new emulation session and build a new emulation configuration command file
- Enter emulation configuration with a previous emulation command file name to edit the options contained therein
- Enter run-time emulation directly (option continue) with a valid emulation command file specified
- Specify a user program absolute file to be loaded after entering run-time emulation (this can be done with any of the above options).

If there are multiple module sets present in the station's card cage (emulator sets, state analyzer sets, timing analyzer sets), then entry is via the measurement system monitor which itself is entered from the development station with the command meas_sys. The measurement system is described in the box on page 8. Softkey labels appear in the

© Copr. 1949-1998 Hewlett-Packard Co.

measurement system display identifying the module and slot number of the control card or that module (e.g., em8086_4 or state_8 for an 8086 microprocessor emulator control board in slot 4 and a state analyzer control board in slot 8). Selection of an emulator with an optional emulation command file name transfers control directly to run-time emulation if the file matches the current hardware configuration. Otherwise, if the command file needs editing, or if none is specified and there is not a previous emulation session, the user is guided through configuration under the current session.

Configuration

The configuration session is a guided series of question and answer entries, except for the memory-mapping portion, which has a directed-syntax command entry format. Configuration consists of the following:

1. Validate and display the card selection (see Fig. 1a).
2. Clock source (internal or external).
3. Real-time option selection.
4. Memory blocking factor (256 or 4096 bytes).
5. Number of significant address bits (dependent upon block size and processor address size).
6. Break on processor writes to ROM option selection.
7. Memory-mapping session.
8. Simulated I/O address assignment, if desired, for display, printer, keyboard, RS-232-C/V.24, and/or disc file(s).
9. Emulator pod hardware settings (microprocessor specific).
10. Selection of alternate inverse assemblers for coprocessors or special operating modes (e.g., 8089 coprocessor or Z8001 nonsegmented mode).
11. Configuration analysis hardware interaction.
12. Naming the emulation command file to store the configuration information and subsequent run-time information on disc for repeated use.

See Fig. 1b for an example of a memory-map display. The memory control board provides the basis for wide flexibility in hardware control of the microprocessor address bus. The address space of the emulated processor can be partitioned into 32 segments (possibly disjoint and of variable size) using block sizes of 4K or 256 bytes. Each segment can be specified as emulation memory, user RAM, ROM, or guarded. With the forthcoming 128K-byte RAM board, a user will have the ability to use up to one megabyte of 64000 emulation memory, mapped anywhere in the user's microprocessor's address space. All the map data is stored in the emulation command file so that it does not have to be reentered each time, and a previously entered map can be easily modified when editing an existing command file.

As an optional feature for microprocessors with the ability to separate address space, two methods exist for overlaying memory address space (where two or more address inputs will map to the same physical blocks of memory controlled by the memory-mapping hardware). The first is to specify explicitly in the memory-map command entry that segments of equal size map to the same physical memory. The other is to "don't care" upper address bits by entering a reduced value in response to the configuration question "Number of significant address bits?"

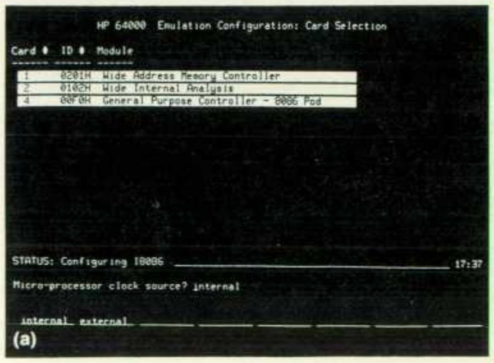
After completing configuration, the user initializes the necessary software and hardware and enters the run-time emulation monitor. If the user has specified a load file, it is automatically loaded at this time.

General Control (Monitor Functions)

The emulation monitor program, which is linked with the user's application program, provides the control mechanism for the emulation system. As the user completes the development cycle of a software/hardware project, this program can be left out to provide the final phase of system integration and test.

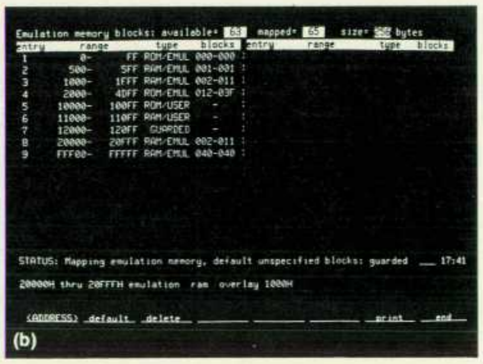
The emulated microprocessor has three basic states: reset, running in monitor (executing the emulation monitor program within emulation memory), and running (executing, but not in the emulation monitor program). Normally under either of the running states the emulation system software polls the target microprocessor with an "are you there?" protocol. This is accomplished through declaration of a global control word, MONITOR_CONTROL, in the emulation monitor program. If the target processor is executing in the monitor loop, the loop responds to the are-you-there? query and performs the requested action. Using this communication mechanism, the emulation software makes specific coded requests requiring the target processor's action such as dumping register values, accessing user memory or I/O ports, or checking for software breakpoint entry.

Under special circumstances the user can specify that runs be restricted to real-time (see configuration step 3). In this case, the emulation software does not poll the target processor and there are no breaks in the user's execution unless the user has issued an explicit break command. This command forces the microprocessor into the emulation



```
            HP 64000  Emulation Configuration: Card Selection

Card #  ID #   Module
------  ------ ------
 1      0201H  Wide Address Memory Controller
 2      0102H  Wide Internal Analysis
 4      00F0H  General Purpose Controller - 8086 Pod

STATUS: Configuring I8086 ______________________________________ 17:37

Micro-processor clock source? internal

 internal  external ________ ________ ________ ________ ________ ________
```

(a)



```
Emulation memory blocks: available= 63  mapped= 65   size= 256 bytes
entry    range          type    blocks  entry    range         type  blocks
 1         0-      FF ROM/EMUL 000-000 :
 2       500-     5FF RAM/EMUL 001-001 :
 3      1000-    1FFF RAM/EMUL 002-011 :
 4      2000-    4DFF ROM/EMUL 012-03F :
 5     10000-   100FF ROM/USER    -    :
 6     11000-   110FF RAM/USER    -    :
 7     12000-   120FF  GUARDED    -    :
 8     20000-   20FFF RAM/EMUL 002-011 :
 9     FFF00-   FFFFF RAM/EMUL 040-040 :

STATUS: Mapping emulation memory, default unspecified blocks: guarded ___ 17:41

20000H thru 20FFFH emulation  ram  overlay 1000H

 <ADDRESS>  default  delete  ________ ________ ________  print   end
```

(b)

Fig. 1. *A configuration session begins with (a) validation and display of the card selection. (b) A typical memory-map display.*

© Copr. 1949-1998 Hewlett-Packard Co.

monitor loop and allows the emulation system to begin the handshake protocol described above. When the user issues a run command, real-time execution resumes.

The emulation system control software is driven by HP's directed-syntax command monitor. The diagram in Fig. 2 roughly outlines the software architecture within the 64000's memory, which is separate from emulation memory. The dynamic overlay area provides space for software needed to complete a user-issued command. Keeping the size of the dynamic overlays small reduces delay when accessing mass storage and makes available a large number of software modules to service the extensive command set. Thus, major disc accesses occur only when loading, storing, or listing large user files, returning to configuration, or ending the emulation session. This architecture lets the user issue any command directly without having to request a certain interface first. The user is guided in command entry via directed-syntax softkeys, and at any point in the run-time monitor, all the possible commands are available. To simplify and guide command entry, the user is provided with the first keyword of each command on three levels of softkeys. The user can cycle through these levels by pressing the softkey labeled ---ETC---.

Symbolic Interface

The addition of rolling and paging functions for the display mode makes the display of global and local symbols more flexible. Also, the symbolic memory references are now separated into prog, data or comm types. There are three new symbol types available: #⟨NUMBER⟩, denoting a Pascal source line number, @:⟨NAME⟩, which gives the starting address of the named module, and the addition of lowercase identifiers to support HP's C compilers.

In the 8086/88 microprocessor family, the user can now enter addresses in two formats. The logical address format is the segment-offset format followed by Intel Corporation. This takes the form of a 16-bit segment followed by a colon, followed by a 16-bit offset. The other form, physical, is the folded logical address from which a 20-bit address is derived. To enter a physical address, simply enter a single number of up to 20 bits.

Command File Execution and Control

A command file can be invoked within emulation by typing in the desired file name. Any valid emulation command can be followed by a semicolon, which will act as an end-of-line character. This becomes the comment-field delimiter, a very useful feature for command files used in testing situations. A command file for use in emulation is

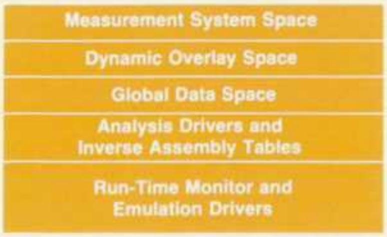

Measurement System Space
Dynamic Overlay Space
Global Data Space
Analysis Drivers and Inverse Assembly Tables
Run-Time Monitor and Emulation Drivers

Fig. 2. *A rough outline of the run-time use of the 64000 system memory.*

Table I
64000 Emulation Wait Commands

Command	Waiting Condition Before Processing Next Command
wait	Any keystroke
wait X	Any keystroke, or X number of seconds
wait measurement_complete	Any keystroke, or measurement to come complete

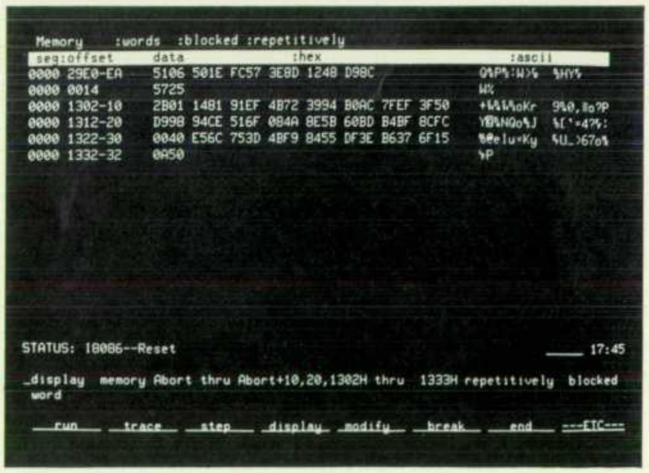
Notes:
1. When operating in remote mode, the "wait for any keystroke" condition is not enabled.
2. While under a wait condition, striking the reset key once will satisfy the "wait for any keystroke" condition and stop execution of a command file.

logged (created) by the user's completing the following steps:
1. From the system monitor level, issuing the command log_commands to ⟨new command file name⟩
2. Entering emulation
3. Going through all the commands desired in the command file
4. Ending emulation, returning to the system monitor level and issuing the command log_commands off
5. Editing the command file just created and removing the commands that led to the entry and exit of emulation
6. Evoking the new command file from the emulation monitor.

Command delays now allow the user more flexible use of command files (although these commands are also available outside of command files). They allow the user to give the emulation system and target processor time to complete some condition or reach a given state before bringing in the next command. The user may issue these wait commands (see Table I) during a session to create a new command file.

Memory Interface

Memory-related functions include loading and storing



```
Memory     :words  :blocked :repetitively
 seg:offset    data              :hex                              :ascii
0000 29E0-EA   5106 501E FC57 3E8D 1248 D98C                      Q%P%:W>%  %HY%
0000 0014      5725                                               W%
0000 1302-10   2B01 1481 91EF 4B72 3994 B0AC 7FEF 3F50            +%%%oKr  9%0,%o?P
0000 1312-20   D998 94CE 516F 084A 8E5B 60BD B4BF 8CFC            Y%%NQo%J  %['=4?%:
0000 1322-30   0040 E56C 753D 4BF9 8455 DF3E B637 6F15            %@elu=Ky  %U_>67o%
0000 1332-32   0A50                                               %P

STATUS: 18086--Reset                                                   _____ 17:45

_display  memory Abort thru Abort+10,20,1302H thru  1333H repetitively  blocked
 word

 __run__  _trace_  _step__  _display_ _modify_ _break_  __end__  ---ETC---
```

Fig. 3. *Typical memory display.*

© Copr. 1949-1998 Hewlett-Packard Co.

absolute memory image files, displaying and listing memory in human readable format (see Fig. 3), and modifying memory. The load memory command accepts one named absolute file which was produced by the 64000 linker or equivalent. The load file may contain an optional transfer address and can have any number of contiguous or discontiguous segments. The name of the last file loaded is recorded so the symbolic interface software can access the global and local symbols associated with it. The presence of the emulation monitor program symbols is checked and addresses recorded if symbols are present. All target system memory (mapped to emulation or user resources) is accessed by the emulation system software in variable block sizes from one to 250 bytes. This feature speeds up the load memory command, allowing it to handle large files quickly.

The display/list interface now allows complex list requests and remembers the last display format and list entered. The user can enter up to 16 memory list entries, each consisting of a single memory location or a memory range, when displaying or listing memory in either blocked or absolute format and in either bytes or words. This lets the user view or list an arbitrary set of locations or ranges anywhere in memory. If the list specifies more than one screenful of data, the roll or page keys can be pressed to scroll the desired segments of the list specification into view. The repetitive display option periodically updates the memory locations currently shown on the screen.

The mnemonic memory display/list format accepts either starting address or address range. Also, rolling and paging are more sophisticated. The **ROLL UP** and **NEXT PAGE** keys add to the current display starting at the next available address at the bottom of the display. An internal stack keeps track of several pages so that the **ROLL DOWN** and **PREV**ious **PAGE** keys respond quickly and logically. When a **ROLL DOWN** or **PREV PAGE** key is pressed and the data is available, an algorithm is used to build back from the first address displayed. There are cases where ambiguous results occur, so the ↑ and ↓ keys rebuild the screen by adjusting the current first display address up or down by one byte and then doing the inverse assembly beginning at that new address. Inverse assembly for memory, register, and trace display/list is handled by a table-driven inverse assembler.

The modify memory command accepts byte or word mode, a single target location or a target range of locations, and a single hexadecimal value or a list of hexadecimal values. The modification value(s) are interpreted as specified, either as bytes or words (actual memory accesses are on a byte basis). For example,

```
modify memory 1000 to 05
modify memory word 2000 thru 3FFF to 0FFFF
modify memory byte START to 0EA, 0EB, 0EC
modify memory 12F0 thru 18FF to 0, 1, 2, 3, 4
```

A new capability of the display/modify memory feature is the use of real numbers. Both short (32-bit) and long (64-bit) real numbers are supported using the IEEE standard floating-point format. These formats are also used by the 64000's compilers and assemblers. Short real numbers are displayed with six significant digits, and long real numbers are displayed with fifteen significant digits. All memory display options, such as multiple addresses or address ranges, are available in real mode, as are all such options for the modify memory command. The display format is (address) sd.dddddEsdd for short real numbers and (address) sd.ddddddddddddddEsddd for long real numbers where s is the sign of the number or its exponent (displayed only if negative) and d is a digit of the number or its exponent. Three special symbols may appear in the real number display: NaN=not a number, +INF=positive infinity, and −INF=negative infinity.

I/O Interface

A new facility offered in the latest emulation release is the ability to interrogate I/O address space for those microprocessors with that feature. The I/O interface handles up to 16 bits of I/O address.

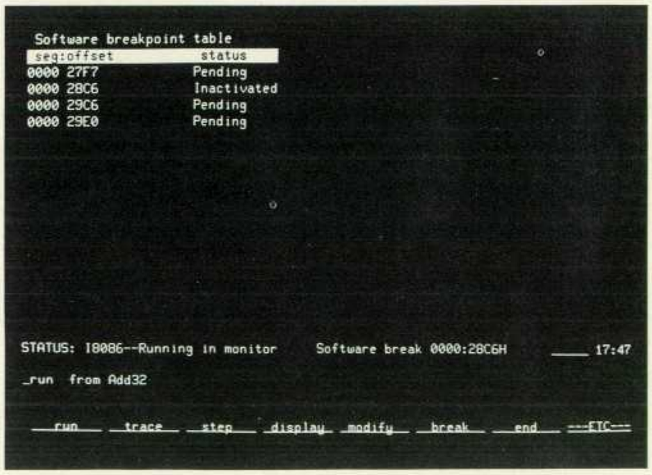
The display/list io_port command works like its display/list memory counterpart except that when the word or bytes mode is specified, accesses are made only in that mode. This lets the user control the accesses made to I/O port addresses. The display/list interface accepts a display list of up to 16 entries, each entry being a single address or an address range. A continuous option does a repetitive update of all locations on the display. Rolling and paging work in the same manner as with the display memory interface. Format options include absolute (one entry per display line) or blocked (eight entries per display line).

The modify io_port command is like its counterpart, modify memory, but again the mode (word or byte) forces all accesses to be done by whichever mode is specified.

Software Breakpoint

The emulation system can perform an effective break on execution by using the modify software_breakpoint set ⟨ADDR⟩[⟨ADDR⟩...] command where ⟨ADDR⟩ is the beginning of any valid instruction. A valid address may take the form of a number, an expression, or a symbol. The display and/or list of the software breakpoints (Fig. 4) allows the user to view the entered breakpoints and their current status. As many as 16 breakpoints are maintained in a table stored in the emulation command file, which is kept intact at the emulation session and is available when the session is resumed.

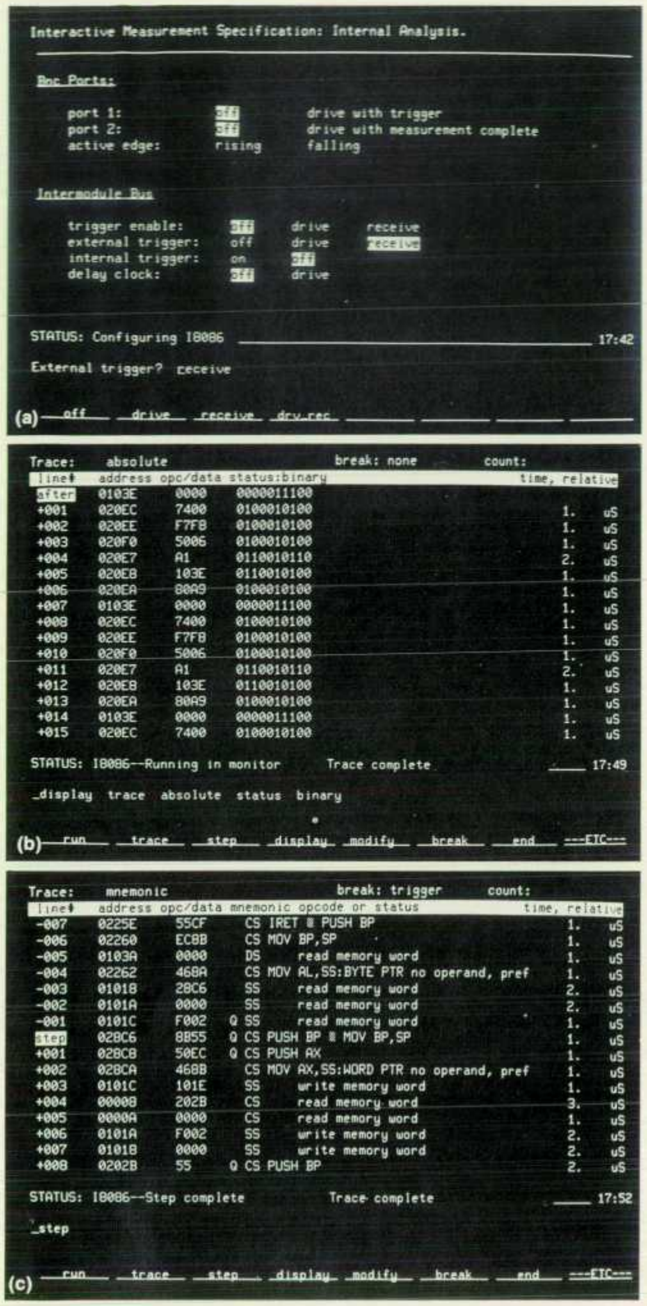
Setting breakpoints, combined with a trace before SWBK_ENTRY command, provides a convenient tool for



```
Software breakpoint table
seg:offset            status
0000 27F7             Pending
0000 28C6             Inactivated
0000 29C6             Pending
0000 29E0             Pending

STATUS: I8086--Running in monitor     Software break 0000:28C6H        ____ 17:47

_run  from Add32
```

Fig. 4. *Software breakpoint table display.*

© Copr. 1949-1998 Hewlett-Packard Co.



```
Interactive Measurement Specification: Internal Analysis.

Bnc Ports:

    port 1:            off       drive with trigger
    port 2:            off       drive with measurement complete
    active edge:       rising    falling

Intermodule Bus

    trigger enable:      off    drive    receive
    external trigger:    off    drive    receive
    internal trigger:    on     off
    delay clock:         off    drive

STATUS: Configuring I8086 ________________________________________ 17:42

External trigger?  receive

(a)___off___ __drive__ _receive_ _drv_rec_ _________ _________ _________ _________
```

```
Trace:    absolute                           break: none        count:
 line#   address opc/data status:binary                          time, relative
after    0103E     0000     0000011100
+001     020EC     7400     0100010100                               1.    uS
+002     020EE     F7FB     0100010100                               1.    uS
+003     020F0     5006     0100010100                               1.    uS
+004     020E7     A1       0110010110                               2.    uS
+005     020EB     103E     0110010100                               1.    uS
+006     020EA     80A9     0100010100                               1.    uS
+007     0103E     0000     0000011100                               1.    uS
+008     020EC     7400     0100010100                               1.    uS
+009     020EE     F7FB     0100010100                               1.    uS
+010     020F0     5006     0100010100                               1.    uS
+011     020E7     A1       0110010110                               2.    uS
+012     020EB     103E     0110010100                               1.    uS
+013     020EA     80A9     0100010100                               1.    uS
+014     0103E     0000     0000011100                               1.    uS
+015     020EC     7400     0100010100                               1.    uS

STATUS: I8086--Running in monitor      Trace complete           _____ 17:49

_display  trace  absolute  status  binary

(b)___run___ __trace__ __step___ _display_ _modify__ __break__ ___end___ ---ETC---
```

```
Trace:    mnemonic                           break: trigger     count:
 line#   address opc/data mnemonic opcode or status                time, relative
-007     0225E     55CF       CS IRET ▮ PUSH BP                      1.    uS
-006     02260     EC8B       CS MOV BP,SP                           1.    uS
-005     0103A     0000       DS      read memory word               1.    uS
-004     02262     468A       CS MOV AL,SS:BYTE PTR no operand, pref 1.    uS
-003     01018     28C6       SS      read memory word               2.    uS
-002     0101A     0000       SS      read memory word               2.    uS
-001     0101C     F002     Q SS      read memory word               1.    uS
step     028C6     8B55     Q CS PUSH BP ▮ MOV BP,SP                 1.    uS
+001     028C8     50EC     Q CS PUSH AX                             1.    uS
+002     028CA     468B       CS MOV AX,SS:WORD PTR no operand, pref 1.    uS
+003     0101C     101E       SS      write memory word              1.    uS
+004     00008     202B       CS      read memory word               3.    uS
+005     0000A     0000       CS      read memory word               1.    uS
+006     0101A     F002       SS      write memory word              2.    uS
+007     0101B     0000       SS      write memory word              2.    uS
+008     0202B     55       Q CS PUSH BP                             2.    uS

STATUS: I8086--Step complete             Trace complete         _____ 17:52

_step

(c) ___run___ __trace__ __step___ _display_ _modify__ __break__ ___end___ ---ETC---
```

Fig. 5. *(a) Display of interactive measurement specification, receiving external trigger with internal trigger off. (b) Trace display of 8086 showing 20 address bits and 10 status bits (in binary). (c) Display of a step showing bus activity that would not be observed in a register display.*

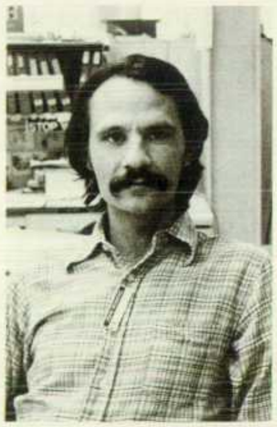
software analysis. SWBK_ENTRY is a global symbol defined in the monitor program. The data found at the breakpoint address is saved in the breakpoint table. A special code (byte or word, depending on the microprocessor) replaces the data in a program, and when the program execution reaches the special code, a break into the emulation monitor program occurs. The processor is then running in the monitor program, and a message is displayed showing the current program counter address. Any displayed breakpoints reflect a change in status from pending to inactivated (Fig. 4). The data from the breakpoint table is restored to the program and the program counter reflects the breakpoint address. Execution from the breakpoint in the program now can be continued by issuing a run or step command.

A step command (to get the processor past the breakpoint address) followed by a modify software_breakpoint set ⟨ADDR⟩ command reactivates that breakpoint. A modify software_breakpoint set command reactivates all inactive breakpoints. If the special code is executed and the address of execution does not correspond to an entry in the table, the message Undef. software break trap is displayed.

Analysis

Analysis enhancements include the ability to participate in coordinated analysis via the measurement system, support for a 48-channel analysis board, closer correspondence between real-time hardware capability and command options, and extended ability to save traces.

The most significant change is the ability to coordinate measurements with other analysis modules (timing, state, or other emulation analysis). There is now synchronous initiation of the participating modules, including starting of analysis and running of target microprocessors. Measurements can be specified that use emulation analysis not only to enable its internal trigger via an external signal, but also to trigger another module or receive an external trigger. This allows tracing of communication between coprocessors. It also provides greater depth via sequential triggering and simultaneous timing and state analysis or the use of the



John P. Romano

Born in Denver, Colorado, John Romano attended the University of Colorado where he earned a BS degree in education in 1971, a BA degree in mathematics in 1972, and an MA degree in education in 1976. After working as a land surveyor and an elementary school teacher, he joined HP in 1980. John worked on the software for the 16-bit emulators and is a project manager for the next generation of 64000 software. He lives in Colorado Springs, Colorado with his two teenage children, a son and a daughter. In his spare time, he enjoys running, bicycling, attending concerts and movies, and reading.

David B. Richey

After working for three years as an ATE and custom IC design engineer for a major semiconductor manufacturer, Dave Richey joined HP in 1979. He worked on the 68000 emulator design, was the project manager for the 16-bit emulators, and now is the project manager for the 1630A/D Analyzers. Dave attended Ohio State University, receiving a BSEE degree in 1976. Born in Fort Wayne, Indiana, he now lives in Colorado Springs, Colorado. He is married, has a son, and enjoys photography, fishing, and tinkering with computers.

© Copr. 1949-1998 Hewlett-Packard Co.

enable function for complex triggering, using the resources of two or more analyzers.

These functions are selected within emulation configuration during the interactive measurement specification segment (Fig. 5a). Then the specified function occurs whenever a measurement is executed, until the interaction specification is modified. Two new commands allow specification of traces or runs without execution and subsequent execution without repeated specification.

The new 64302A 48-channel Emulation Analysis Board and the earlier 40-channel board are supported by the new software. The eight additional channels can be allocated as needed by the particular microprocessor to either address (up to a maximum of 24 bits) or data (up to a maximum of 16 bits), or both (Fig. 5b). A related feature is the ability to specify the number of significant address bits and have the emulation analysis automatically ignore higher address bits. New display options include binary (Fig. 5b) and mnemonic status displays.

Other changes include an improved trace command syntax that corresponds exactly to the hardware's real-time triggering and storage capability. The combined resources of emulation analysis, software breakpoints, and command files allow application-specific nonreal-time analysis and enhance the system's general flexibility. Trace specifications and trace data can be stored in a trace file and loaded again later to reexamine the data or to reuse the specification. Also, entire traces can be listed to a file with a single command, rather than a series of partial listings.

The step and run until commands are implemented using emulation analysis to break into the monitor. As a result, when using these features, a trace can be displayed (Fig. 5c) that shows the bus activity occurring during the current instruction for the step command and up to and including the final instruction for the run until command.

In trace specifications there is now a set of symbolic keywords to be used in place of the normal hexadecimal or binary "don't care" specification for naming status values, and a capability to build status expressions using the operator and with status keywords (or "don't care" numbers).

Acknowledgments

Credit for the hardware design goes to Bryan Call for the Z8001 and Z8002 emulator pods, Kevin Hall for the 64155A wide-address memory controller, Dennis Heitzman for the 8086 and 8088 emulator pods, Bob Morrison for the 64271A emulation control board, Dan Schmelzer for the 64302A 48-channel internal analyzer, and Phyllis Smith for the 64161A, 64162A and 64163A static RAM boards.

Thanks go to Steve Graham for his description of analysis changes and real-number support.

Credit for software design also goes to Brian Kerr, team leader, Steve Graham, and Gordon Greenley.

Reference

1. J.B. Donnelly, G.A. Greenley, and M.E. Mutterspaugh, "Emulators for Microprocessor System Development," Hewlett-Packard Journal, Vol. 31, no. 10, October 1980.

High-Level Language Compilers for Developing Microprocessor Systems

by Martin W. Smith and Joel D. Tesler

THE TERM "UNIVERSAL," when applied to a microprocessor development system, should apply to the software provided with the system as well as to the hardware. That is, if a compiler for high-level language X is supported by the development system, and if the system provides hardware support for microprocessors A, B and C, then the compiler for language X should be able to generate code for microprocessors A, B and C. Conversely, if the development system provides hardware support for microprocessor A, then any high-level language compiler supported by the system should be able to generate code for A.

These requirements suggest a structure for the high-level language compilers supported by the 64000 Logic Development System. This structure, shown in Fig. 1, is similar to a restaurant menu. It shows that compilers for languages X and Y (Pascal and C in the diagram), are really just different entry points into a high-level language system.

The parts of the system are represented by the items on the menu. Thus, just as a customer desiring a good meal can pick from the menu "one appetizer, one main course, and one dessert," so can a user of a 64000 Logic Development System select a language from column A and a microprocessor from columns B and C to get relocatable code for that processor in column D.

Using this type of structure has some important benefits for the user. First, if a new microprocessor begins to win acceptance in the marketplace, Pascal and C compilers for that processor can be created by providing one code generator and one set of tables (columns B and C) to the compiler system. This reduces the labor normally needed to build an entire compiler by a factor of three to six. Second, if a new language begins to see wide use in the microprocessor environment, a compiler for that language can be brought up on the 64000 System by providing a pass 1 for

© Copr. 1949-1998 Hewlett-Packard Co.

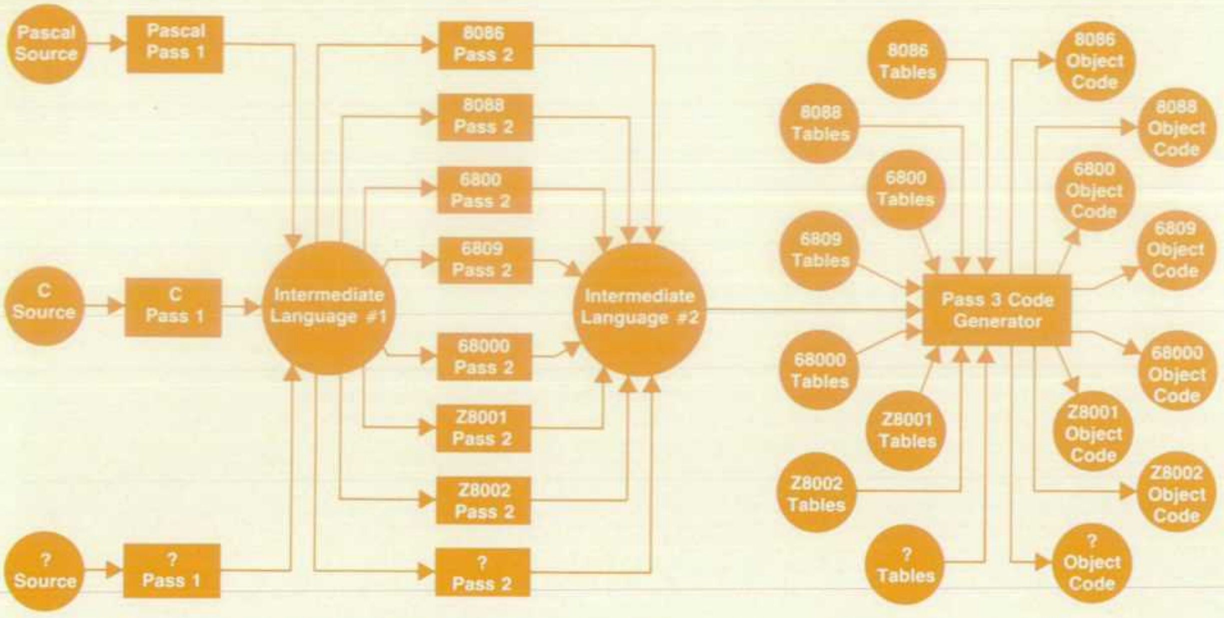


Fig. 1. *The compiler structure in the 64000 Logic Development System.*

the language (column A). Again this takes much less labor compared to that needed to produce an entire compiler. Third, the user can look forward to better service for detecting and correcting compiler bugs. When a user of the 8086 Pascal compiler reports a software bug, and that bug turns out to be caused by a problem in Pascal pass 1, then when the bug is corrected in the Pascal pass 1, it is corrected for all the supported microprocessors in column B. And if the bug turns out to be located in 8086 pass 2, then once corrected, it is corrected for all of the supported high-level languages.

Compiler Languages

At first glance, the languages C and Pascal may appear similar. Both are structured languages and contain similar looping constructs. Both have similar data types containing integers, reals, pointers, arrays, records or structures, and with the addition of the enum data type to C, scalars. Given the similarities between the two languages, the question may arise as to the reasons for supporting both languages on the 64000 System.

There is a basic difference in philosophy between C and Pascal. C is generally more concise (and therefore at first glance more cryptic) than Pascal. C gives the programmer as much freedom as possible, and imposes few restrictions on the user. By contrast, Pascal protects the programmer from certain types of errors, resulting in limited freedom.

An example of this difference is illustrated by procedure calls. In Pascal procedures, the number and types of parameters must be declared explicitly. At every call, the arguments are checked, and if there is any incompatibility in either type or number, an error message is given. In C, no check is made. Note that in C it is much easier to make an error in parameter passing. However, variable numbers and types of parameters can be passed in C, which is impossible to do in Pascal. Of course, both the sending and receiving routines must conform to the same parameter passing convention, since there is no way of verifying the number and type of parameters that were passed to it. Failure to do this results in unpredictable behavior by the program.

Another example of this difference in philosophy is apparent in the use of pointers and addresses. In Pascal, all pointers point into the heap, an area of memory specifically allocated for dynamic memory (i.e., NEW, DISPOSE, etc...). This tends to prevent writing over random memory not intended for that purpose. In C, there is a specific operator for taking an address. Additionally, pointers and integers are assignment compatible. Therefore a C pointer may contain anything, a potentially dangerous but powerful tool.

In Pascal, an integer and a set are two different constructs. A set can contain any number of elements (subject to the limitations of the compiler) and the representation is not specified (at least not in a manner that is transportable between one compiler and another). In C, logical operations can be done on any integer. This is useful for doing bit masking on integers.

Pascal/64000 has certain extensions that allow the user accesses to certain types of functions possible in C. For example, type changing enables the user to bypass some of the type checking normally done by Pascal. It can also be used to convert between integers and sets, thus allowing masking of integers. Another extension is the ADDR function, which gives the ability to take addresses. This allows Pascal/64000 pointers to point anywhere in memory, not just the heap. Other features of C, such as variable parameter passing, are not available as extensions to Pascal/64000.

The following example illustrates the difference between the two languages. The C statement:

```
c += a[ *w++ = getch( )]
```

first calls a function getch, which returns a character. This character is then assigned to the location pointed at by w, after which w is then set to point to the next character (presumably w points into a buffer). The character is then looked up in an array a, and its value is added to c. To write the same statement in Pascal, the following would be necessary:

© Copr. 1949-1998 Hewlett-Packard Co.

```
    buffer[w] := getch;
    w := w+1;
    c := c + a[ buffer[w] ];
```

The only difference is that here w is an integer instead of a pointer, and the buffer must be specified explicitly.

The C statement is more concise, but sacrifices readability for the person who is not well versed in C. Note however, that the program could have been written in C to look exactly like the Pascal program (except for minor syntax changes, i,e., = instead of := and ( ) after getch).

Table I summarizes some of the major differences between Pascal and C. Those items available as extensions in Pascal/64000 are indicated.

The complicated arithmetic expressions referred to in Table I for C include such things as autoincrement and decrement, conditional assignment, exclusive OR, and logical shift (logical shift is also available in Pascal/64000). On the other hand, Pascal/64000 has the nonstandard feature rotate.

Although parameter passing differs between Pascal and C, and thus may not be automatically compatible between the two languages (depending on the specific target processor compiled for), a special compiler directive is available in the C compiler to guarantee compatibility with Pascal for a given procedure. While the use of this option places certain restrictions on the user, such as the inability to have a variable number of parameters, it does allow both languages to be linked together. Since assembly language can also be linked in, it is not at all difficult for a developed system to be made up of modules in all three languages. For example, assembly language can be used to access specific microprocessor I/O instructions and for other code not possible in a high-level language. C might be used to write device drivers, and Pascal can be used for applications of those drivers.

Table I
C and Pascal Differences

C	Pascal
Weak type checking	Strong type checking
Variable parameters	Fixed parameters
Logical operations on integers	Sets
Boolean operations on integers	Boolean variables
No nested procedures	Nested procedures
Address function	Available in Pascal/64000
Type casting	Type changing in Pascal/64000
Different-sized integers	Available in Pascal/64000
Unsigned arithmetic	Available in Pascal/64000
Complex pointer operations	None
Complicated arithmetic expressions (see text)	Simpler arithmetic expressions (see text)
Little programmer protection	Strong programmer protection

Joel D. Tesler
Joel Tesler joined HP in 1980 as a development engineer. He designed the C compiler pass 1 and the Z8000 code generator for the 64000 System. Joel was born in Los Angeles, California and has a BS degree in electrical engineering and computer science awarded by the University of California at Davis in 1980. He lives in Colorado Springs, Colorado, likes bicycling, and is an avid punster.

Martin W. Smith
Marty Smith was born in Tacoma, Washington and attended the University of Puget Sound, earning a BS degree in mathematics in 1972. He also studied at the University of Washington, receiving an MS degree in computer science in 1978. Marty served in the U.S. Air Force as a first lieutenant and worked as a senior software engineer before joining HP in 1978 as a design engineer. He lives in Colorado Springs, Colorado and enjoys swimming and playing slow-pitch softball.

Hewlett-Packard Company, 3000 Hanover Street, Palo Alto, California 94304

Bulk Rate
U.S. Postage
Paid
Hewlett-Packard
Company

HEWLETT-PACKARD JOURNAL
MARCH 1983 Volume 34 • Number 3
Technical Information from the Laboratories of Hewlett-Packard Company
Hewlett-Packard Company, 3000 Hanover Street
Palo Alto, California 94304 U.S.A.
Hewlett-Packard Central Mailing Department
Van Heuven Goedhartlaan 121
1181 KK Amstelveen, The Netherlands
Yokogawa-Hewlett-Packard Ltd., Suginami-Ku Tokyo 168 Japan
Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive, Mississauga, Ontario L4V 1M8 Canada

0200020707&&&BLAC&CA00
MR C A BLACKBURN
JOHN HOPKINS UNIV
APPLIED PHYSICS LAB
JOHNS HOPKINS RD
LAUREL MD 20707

CHANGE OF ADDRESS: To change your address or delete your name from our mailing list please send us your old address label. Send changes to Hewlett-Packard Journal, 3000 Hanover Street, Palo Alto, California 94304 U.S.A. Allow 60 days.

5953-8509

© Copr. 1949-1998 Hewlett-Packard Co.