Host Independent Microprocessor Development Systems

A new architecture makes it possible to use this family of emulators with workstations, mainframes, or personal computers. The cabling technology and chassis design inprove performance and usability.

by Arnold S. Berger

HP 64700 SERIES EMULATORS are a new family of products representing the latest stage in the evolution of the HP 64000 Microprocessor Development System, introduced in 1979. The original family of products, which is still going strong, was designed around a cluster of development stations sharing a local disc memory and printer. This cluster uses an optimized version of the HP-IB as its LAN core, and is a very efficient hardware and software integration environment for small engineering teams. In fact, many of today's concepts in distributed computing networks were first applied when the HP 64000 system was introduced.

As microprocessors became more complex, the development environment shifted to larger teams of software engineers working on small to medium-size mainframe computers, such as the Digital Equipment Corporation VAX™ and the HP 9000 Series 500. Hewlett-Packard responded to this trend with the introduction, in 1985, of the HP 64000 Hosted Development System. This system permits an existing HP 64000 cluster to be linked to the mainframe computer with the high-speed HP-IB file transfer capability, while the software cross-development tools remain resident on the host mainframe. Individual HP 64000 development stations can be operated under remote control via the terminals on the host machine.

However, the industry trends in software engineering were pointing towards more open software development environments. The large software development teams, now writing software for 32-bit microprocessors with address ranges measured in gigabytes, needed environments that could best be serviced by HP-UX (Hewlett-Packard's version of AT&T's UNIX® operating system), while the small teams were moving towards software development on HP Vectra-class computers.

Hewlett-Packard responded to the needs of the large teams with the introduction, in 1986, of the HP 64000-UX family of hardware and software-based products. Work was also in progress to develop a family of products that could satisfy the needs of the single engineer or small development team, without the proprietary technology of the original HP 64000 Logic Development System. We recognized that the IBM PC/AT and the Microsoft MS-DOS® operating system had become de facto computing standards for the engineering community and that a product offering was needed for these systems. In addition, we had the enviable position of being able to start with a clean slate, using the knowledge gained through the evolution of the original families of products.

Host Independence

During the investigation phase of this project, the entire design team went on the road to visit customers. Both our installed base of customers and customers who did not have any of the HP 64000 products were visited. In all, some 35 companies who design products containing embedded microprocessors were surveyed to determine the set of features that were considered most appropriate for a personal-computer-based microprocessor development system. The HP 64000-PC Personal Integration Environment, based on the HP 64700 Series emulators/analyzers, is the result of our effort to bring a microprocessor development system to the personal-computer-based engineering team.

Early in the HP 64700 product definition phase we realized that expanding the product definition to include the requirement of host independence would free the product from dependence on a particular computer or operating

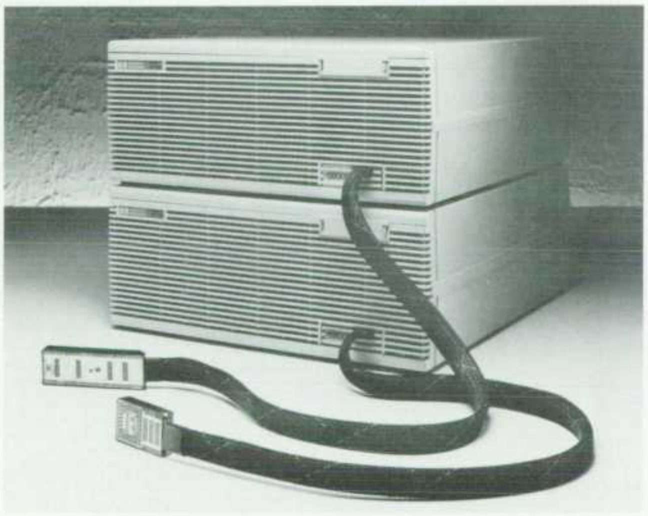


Fig. 1. *These HP 64742A Emulators for the MC68000 microprocessor are members of the HP 64700 Series. They are available with DIP or PGA cabling. HP 64700 emulators are host independent and can operate with workstations or personal computers.*

system and broaden the environment in which it could function effectively. At the lowest level, host independence implies a totally self-contained microprocessor emulation vehicle. The emulator contains its own local controller, which has sufficient on-board operating software to relieve the remote computer of all base-level control functions. All commands to the new HP 64700 Series emulators are constructed from simple ASCII character strings.

The emulator communicates with its controller over a standard-protocol, serial network. The world beyond its RS-232-D/RS-422 port is understood to be an 80-character-by-24-line "dumb" terminal. File transfers using industry standard formats are accepted. Thus, any host computer (or terminal) can control an HP 64700 Series product, making it useful for a variety of tasks, from code development through field service applications.

Host independence also means that the product can be used as easily with workstations such as the HP 9000 Series 350, with mainframe computers such as the DEC VAX, and with personal computers, such as the HP Vectra PC. Any cross-development software that produces an industry standard file format can create object files for use by HP 64700 systems.

The initial product offering consisted of four products: the HP 64742A/L 68000 Emulator, the HP 64753A/L Z80 Emulator, the HP 64764A/L 80186 Emulator, and the HP 64765A/L 80188 Emulator.

Follow-on products include emulators for the 8086/8088 microprocessor families and popular microcontrollers. Table I is a listing of microprocessors and microcontrollers supported at the time this article appears. Fig. 1 is a photo of the HP 64742A Emulator.

Table I
HP 64700 Series Emulators

Model	Processor	Type
64742	68000	General-Purpose 16-Bit
64745	68010	General-Purpose 16-Bit
64753	Z80	General-Purpose 8-Bit
64762	8086	General-Purpose 16-Bit
64763	8088	General-Purpose 16-Bit
64764	80186	General-Purpose 16-Bit
64765	80188	General-Purpose 16-Bit
64771	80C196	Microcontroller 16-Bit
64786	TMS 32020	Digital Signal Processor
64787	TMS 320C25	Digital Signal Processor
64764C	80C186	General-Purpose 16-Bit
64765C	80C188	General-Purpose 16-Bit
64731	V25	Microcontroller 16-Bit
64733	H16	General-Purpose 16-Bit
64735	647180X	Microcontroller 8-Bit

Emulation

Before going any further it might be instructive to review what an emulator is and how it operates. Fig. 2 is a block diagram of the major components of the emulation system. Every emulator contains a duplicate of the processor that is being emulated, which is called the target processor. The system being designed around the target processor is called the target system. The emulation processor resides on the run control board within the emulator and is connected to the processor socket in the target system via line driver buffers and a target system cable. The emulation processor, typically a high-speed version of the target system processor, forms the core of the run control board. Surrounding the emulation processor are basically three functional circuit blocks:

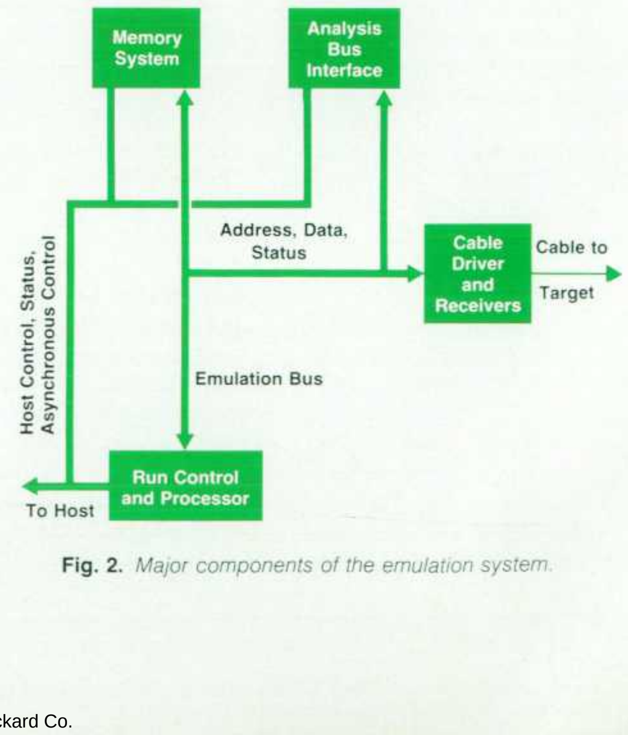
- Run control
- Memory and memory mapping
- Analysis bus interface.

The run control circuitry controls the operation of the emulation processor so that the user can always maintain control of the emulator. Since each microprocessor has a different internal architecture, the run control circuitry must be tailored to provide the interface to the host control system. Run control affects operations such as single-stepping, sharing interrupt signals with the target system, signaling the analyzer, keeping the host controller aware of the status of the emulation processor, and so on.

The emulator contains its own internal emulation memory, which can be used as a substitute for the memory, RAM, and ROM that will eventually be in the target system. This memory can be located anywhere in the address space of the emulation processor. Mapping of the memory and its designation as RAM, ROM, or guarded are handled by the mapping circuitry, which provides a translation function between the address being output by the emulation processor and the address being supplied to the emulation memory array.

The analysis bus interface provides the link between the signals traversing the address, data, and status buses of the emulation processor, the emulation bus analyzer, and the state/timing analyzer. The analyzers will be discussed later in this article.

Two other system blocks, the host controller and the analysis system, are not shown in Fig. 2. The host controller works with the run control system, memory subsystem, and analysis system to provide the coordination of all the complex tasks going on within the emulator. It interfaces with the user to convert commands to proper operational



Memory System
Analysis Bus Interface
Address, Data, Status
Cable Driver and Receivers
Cable to Target
Host Control, Status, Asynchronous Control
Emulation Bus
Run Control and Processor
To Host

Fig. 2. *Major components of the emulation system.*

sequences. The host controller also interfaces independently with the state/timing analyzer, emulation bus analyzer, and memory subsystems. This enables the host controller to communicate with emulation memory and read and write to it while the emulation processor continues to run the user's code. Likewise, the state/timing analyzer or the emulation bus analyzer can be set up or unloaded without stopping the emulator. This type of architecture is called dual-bus, referring to the two independent processor buses (emulation and host control) within the emulator.

Design Goals

As stated earlier, the entire R&D team undertook an extensive series of customer visits. The purpose of these visits was to focus on the features that were most important to users of microprocessor emulation systems. The two most important features were ease of use and transparency.

Every engineer wants products that are easy to use. The key for a design team is to make the product operate in a straightforward manner without compromising the functionality needed to solve complex measurement problems. Functionality, at least for development systems, has two aspects. First, the emulator must provide a comprehensive set of system analysis and processor control functions. Second, as important as the feature set is the transparency of the emulator. An emulator is supposed to behave as if the target microprocessor chip is plugged into the socket in the user's target system. When the performance of the emulator differs from how the target microprocessor would behave, then the transparency of that emulator is compromised.

The HP 64700 emulators have the same high degree of transparency as their predecessors. For the initial HP 64700 products, in-circuit operation is full-speed, with zero wait states in the target system or emulation memory. For the Z80, this is 10 MHz, and for the 80186/88 and the 68000, 12 MHz.

Special care was taken to match the timing and parametric performance of each emulator to those of the target processor. Operation from power-on or system reset is consistent with the target processor chip. Overall, care was taken to see that when differences do occur, as they must in some design environments, the HP 64700 Series performs in a manner consistent with a user's expectations.

To meet the ease of use and transparency design goals for the HP 64700 Series, it was necessary for the design team to address three critical areas: a new architecture for the emulation subsystem, a new chassis configuration, and new cabling technology.

System Architecture

Many possible architectures can be employed in emulator design. The HP 64700 Series uses a dual-bus architecture (Fig. 3), with a foreground or background monitor to permit the user to control the microprocessor emulation in the target system.

The dual-bus architecture has been the heart of the HP 64000 family of products since its introduction in 1979. What makes the architecture of the HP 64700 Series new is the addition of the hybrid monitor. The hybrid monitor, residing either in foreground or background, is a major departure from the original HP 64000 family design.

The monitor is the control program that the emulation microprocessor is running when it is not explicitly running the user's program. The monitor program is very similar to the small programs often supplied with single-board computers. Typically 2K to 3K bytes in size, the monitor handles reading and modifying memory, internal registers, I/O ports, and so on.

Single-stepping and software breakpoints are also handled by the monitor program. While the use of the monitor is easily understandable, its integration within the emulator plays an important role in the emulator's range of useful applications.

Complex applications typically require the use of a foreground monitor. This monitor is a block of code that runs in the same address space (foreground) as the user's own program. It is linked with the user's run-time code at assembly time so that when control of the emulation processor is switched from the user program to the monitor program, real-time system events can still be serviced. These events, such as multiple interrupts and watchdog timers, are very difficult to handle properly with other monitor implementations. The foreground monitor is often the architecture of choice for complex multitasking environments.

Background monitors are easier to use when trying to get a new design off the ground. The monitor is present in a background or shadow memory, and control is transferred to this monitor by remapping (called "jamming") the monitor instructions from the background address space directly to the emulation processor. This type of monitor

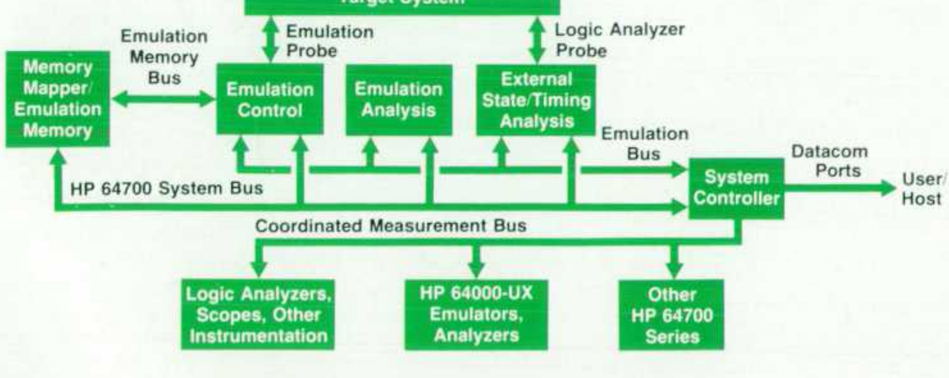


Fig. 3. *HP 64700 emulator system architecture.*

makes no assumptions about the user's environment. The emulation processor can always enter the monitor when instructed to do so. Problems arise from this architecture when it becomes necessary to keep a user's target system alive while the emulator is running in background. The new HP 64700 emulators, such as those for the 8018X and 68000 processors, solve this problem by offering the best of both worlds. Once the user answers a simple configuration question, either the emulation system will operate with the monitor in background, or by assembling and linking the monitor with the run-time code, the user will have the advantages of a foreground monitor. A more complete discussion of foreground and background monitors can be found in reference 1.

Cabling

In keeping with the twin goals of transparency and ease of use, the design of the cable to the target system became a major focus. The cable has to be small, light, flexible, and long, so the user can plug it into a target system where the microprocessor socket is not readily accessible. The signal that starts at the emulator and exits at the target system must look like the signal that would be present if the target processor chip itself were present. Since these cables experience a great deal of mechanical stress over the life of the product and its application in target systems, they have to be rugged.

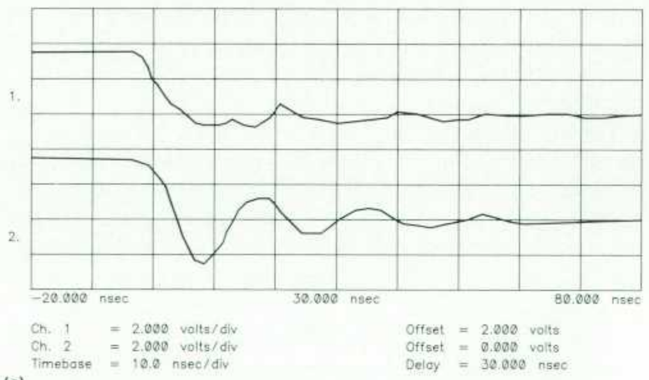
We approached this design task in two ways. A member of the team did an experimental and theoretical study of the electrical performance needed for the emulator to meet its design goal. The objective was to make the signal fidelity in the cable essentially independent of the length of the cable. Another team was working with an outside cable manufacturer to realize the mechanical features needed in the cable.

The length of the cable is important because in the majority of real-world target systems, the microprocessor socket is not easily accessible to the emulation cable, and it isn't always possible, either mechanically or electrically, for the user to have a convenient extender card with the processor socket easily accessible. Since the transit time through the cable is set by the speed of light, the length of the cable for each emulator is set by the maximum operating speed of the processor being emulated and its timing requirements.

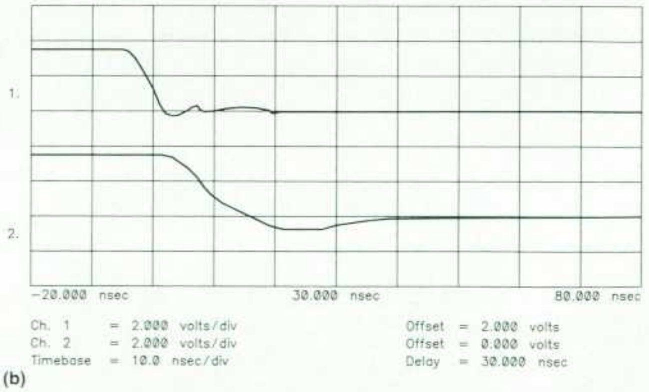
A cable design evolved that borrows heavily from existing oscilloscope technology. However, an oscilloscope typically has between two and eight signals to deal with. The Motorola 68000 microprocessor has 64 pins, of which roughly 60 must maintain high signal fidelity. The cable technology is considered proprietary, so further details cannot be given here. However, the results are easily demonstrated.

Fig. 4 illustrates the signal fidelity achievable with the new cable technology. Fig. 4a is the trace from a signal propagating along a standard ribbon-type cable, 12 inches long, of the type commonly used for connection to a target system. The upper trace is the source end and the lower trace is the exit end of the cable. It is clear that the lower trace exhibits significant undershoot and ringing. Signals such as this would probably lead to erroneous data and inconsistent performance, since correct operation would depend upon how much design margin was available in the user's target system. Fig. 4b is a 1.5-meter length of the controlled-impedance cable used in the HP 64700 Series. The signal fidelity is vastly superior, although some degree of rise time degradation does occur. Since each cable used within the HP 64700 Series is optimized for full-speed operation of the processor in the target system, the effects of cable propagation delay and rise time degradation can be minimized. Also, the clock signals to and from the processor are buffered at the target end of the cable with a small, surface-mounted line driver circuit. In processors such as the Z80, 8086, and 68000, this is not absolutely essential since these processors do not have their own internal oscillator circuitry. The 80186 processor does have an on-chip oscillator that would not function at all if it needed to drive the added capacitance of the cable. In this case, the buffer circuitry provides the oscillator function and drives a clean signal back to the emulator. With this design a Z80 emulator can operate at 10 MHz with no wait states with a one-meter-long cable. The shortest cable of the initial products is 18 inches long, and is designed to be used with an 80186/80188 processor operating at clock speeds up to 16 MHz.

Achieving these levels of performance results in a cable that is expensive to manufacture and test. This would translate to customer dissatisfaction with the product if this



(a)



(b)

Fig. 4. *(a) Signals at the source end (upper trace) and exit end (lower trace) of a standard ribbon-type cable 12 inches long. (b) Signals at the source end (upper trace) and exit end (lower trace) of a 1.5-meter length of the controlled-impedance cable used in the HP 64700 Series.*

performance led to a mechanically fragile cable design. Considerable effort also went into making the cable mechanically rugged and flexible. During the product's development, a special test fixture was designed to flex the cable through thousands of bending cycles, typical of how these cables are stressed in real environments.

Chassis

During our customer visits we found that one of our customers had suspended an HP 64000 Development System from the ceiling by steel cables to get it close to the target system processor, which was located in the topmost circuit card in a six-foot-high equipment bay. The HP 64000 is about the size of a large workstation computer and this was obviously not optimal for mechanical, safety, and convenience reasons. Thus, we saw the need for a chassis that could be easily brought close to the system under development.

The chassis for HP 64700 Series emulators is designed for minimum size. Holes are placed around the perimeter of the chassis to hold two handles, which can be used for standing or suspending the emulator closer to the target system (see Fig. 5). These handles, dubbed the "skyhooks" by a member of the design team, permit the emulator to adjust to any desired mechanical configuration.

Data Transfer

Since the HP 64700 Series uses an industry-standard serial interface structure, the speed at which files and information can be exchanged between the emulator and the host computer is a significant issue. The HP 64700 can transmit and receive files at burst rates up to 450 kilobaud using the synchronous RS-422 channel (imbedded in an otherwise standard RS-232-D port) with hardware handshaking. However, raw bandwidth is useless unless it can be coupled with an equally fast method of loading the files into the processor's memory space. This file transfer link has been optimized so that a realizable system transfer rate in the range of 100 to 150 kilobaud is obtained. This includes the overhead of file formats and file error correction software. Actual file transfer rate measurements have been made. A 256K-byte file was repeatedly downloaded to the emulation memory of each of the HP 64700 emulators. Table II lists the transfer times for three of the initial products.

Table II

HP 64700 Series Emulator 256K-Byte File Transfer Time

Emulator	Time (seconds)
HP 64753A (Z80)	14
HP 64764A (80186)	20
HP 64742A (68000)	26

However, it should noted that this is a special 8-bit binary transfer mode. Files in ASCII or hexadecimal format would load at approximately one half to two thirds of these rates. Buffered interface circuits are currently being designed for use with the HP 9000 Series 300 and HP Vectra Personal Computers (IBM PC/AT compatibles) so that the maximum download speed capabilities can be realized.

Operating Modes

An additional RS-232-D interface is also included with the emulators. This second port allows the emulator to be operated, together with a terminal, from a single drop of a mainframe computer, or in conjunction with a PROM programmer. The HP 64700 emulator can be placed in a pass-through mode so that the terminal can effectively communicate with the host computer. When the appropriate escape sequences are given, the emulator will intercept and execute those commands intended for it. The transparent mode also allows several emulators to be serially connected to each other and to a single host control port. After each emulator is assigned a unique serial address code, it can then be individually controlled by the host.

An emulator can be used in a stand-alone mode, that is, without target hardware. Used this way, it is much like a single-board computer. The stand-alone mode is a very efficient way to develop the embedded code that the final product will eventually be controlled by, even though it may be too early in the design cycle for any real target system hardware. Eventually, of course, it becomes necessary to begin turning on the target system hardware and integrating it with the various software modules that have been and still are being developed for it. At this point the emulator must be able to plug into the target system and substitute for the microprocessor that will be used in the final product.

Analysis

An important feature of the HP 64700 Series is the power of its internal emulation bus analysis and external state and timing analysis. The heart of the analysis system is the logic analyzer on a chip used in the HP 165X family of logic analyzers.[2] Internal analysis of the emulation processor's address, data, and status buses is a standard feature, and an additional 16 external channels of 25-MHz state analysis and 100-MHz timing analysis are available as an option. The external channels can be either tightly or loosely coupled to the internal channels, depending

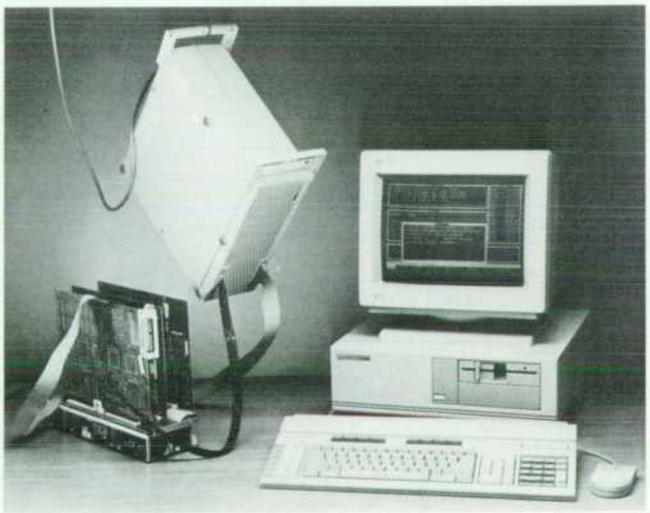


Fig. 5. *Handles attach to the HP 64700 emulator chassis for standing or suspending the emulator close to the target system.*

upon the requirements of the measurement.

The features and applications of the HP 64700 internal analyzer include:

- Eight levels of sequencing for complex program flow tracking
- Address, data, or status range resources
- Prestore queue for variable access tracking
- Time tagging for instruction execution measurements
- 1024-state-deep memory (512 states with time tagging)
- Powerful set of store qualification resources
- Selectable trigger resources
- Code coverage memory for reliability metrics.

Environments

An emulator is only one part of a microprocessor development system. Also required is a comprehensive software environment that will support the microprocessor of choice. The HP 64000-PC Personal Integration Environment provides the required environment. Since the hardware is host independent, any assembler or compiler that can output one of several industry-standard object file formats, such as Intel Hexadecimal File Format, Motorola Hexadecimal S-Record File Format, Extended Tektronix Hexadecimal File Format, and Hewlett-Packard Absolute File Format, can be used. However, to access the complete development system environment the user is encouraged to use software tools that also produce compatible symbol table information for use by HP-approved, high-level debuggers and HP-supported interfaces.

The supported interfaces, initially targeted for HP Vectra Personal Computers and HP 9000 Series 300 workstations, provide a window environment for code development and hardware/software integration. On the HP Vectra PC, the window environment is coupled with an inverted-tree command line structure. The inverted-tree structure is the interface most familar to users of the popular spreadsheet programs. This structure allows complex measurement sequences to be constructed from ever-descending levels of options until the final structure of the command is established. At each level a brief explanation line is provided for each command option. The user selects the option possibilities using the cursor keys or the first letter of the command option. This type of interface is often called noun-driven, since only noun descriptors are used to traverse the tree.

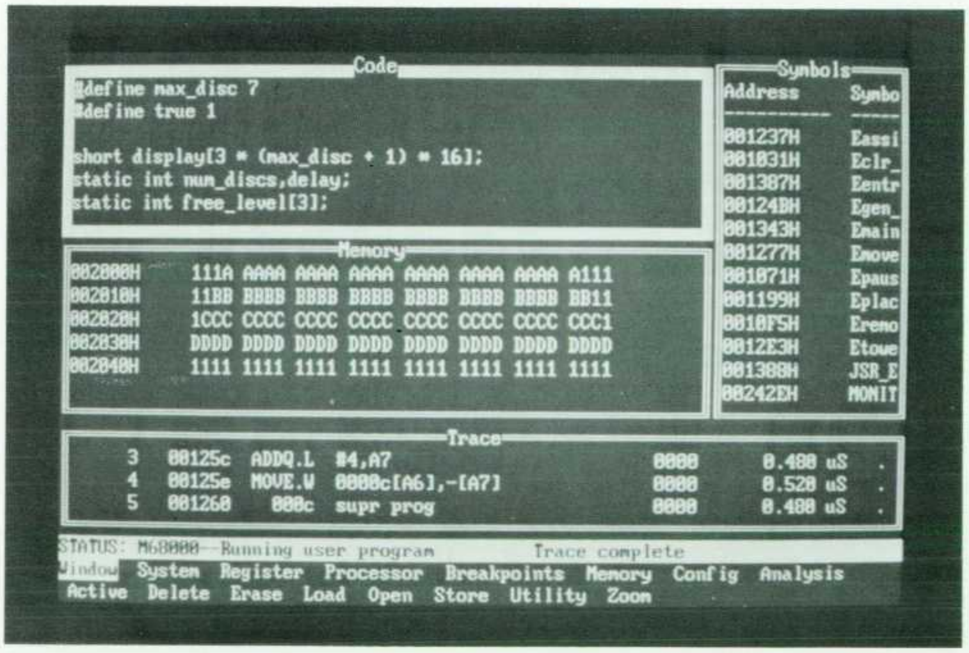
Windows can be set up and customized for various information displays. Register information, memory contents, high-level source code, assembly code, and analyzer trace information can coexist in different windows and be called up for full display. In addition, a terminal window can be invoked. This window allows direct access to the host independent, ASCII command set of the HP 64700 system. Fig. 6 shows this user interface for an MC68000 emulator (HP 64742A).

One of the most challenging chores for the user is the proper structuring of the triggering, tracing, and storing conditions required to build complex measurement structures. The window interface opens up to reveal a straightforward menu construct. Simply filling in the menu fields automatically builds the analyzer's measurement sequence.

On HP 9000 Series 300 Computers the interface is modeled after the original HP 64000 syntax-directed softkey structure. This consistency permits an HP 64700 Series emulator to operate from the same environment as a companion HP 64000-UX emulator.

Multiprocessor Emulation

Multiprocessor systems are becoming the norm, rather than the exception, in the design of products with embedded microprocessors. Industrial robots, telecommunications networks, and aircraft are just a few examples of such applications. The HP 64700 Series contains a special measurement and control structure called the coordinated measurement bus, or CMB. The CMB interface is a 9-pin D-type connector on the rear of the instrument. Up to 32 emulators



```
                          Code                                   Symbols
#define max_disc 7                                         Address      Symbo
#define true 1                                             ----------   -----
                                                           001237H      Eassi
short display[3 * (max_disc + 1) * 16];                    001031H      Eclr_
static int num_discs,delay;                                001387H      Eentr
static int free_level[3];                                  00124BH      Egen_
                                                           001343H      Emain
                          Memory                           001277H      Emove
002000H     111A AAAA AAAA AAAA AAAA AAAA AAAA A111        001071H      Epaus
002010H     11BB BBBB BBBB BBBB BBBB BBBB BBBB BB11        001199H      Eplac
002020H     1CCC CCCC CCCC CCCC CCCC CCCC CCCC CCC1        0010F5H      Ereno
002030H     DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD        0012E3H      Etowe
002040H     1111 1111 1111 1111 1111 1111 1111 1111        001388H      JSR_E
                                                           00242EH      MONIT
                                   Trace
     3   00125c  ADDQ.L  #4,A7                      0000      0.480 uS   .
     4   00125e  MOVE.W  0000c[A6],-[A7]            0000      0.520 uS   .
     5   001260    000c  supr prog                  0000      0.480 uS   .
STATUS: M68000--Running user program          Trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
Active  Delete  Erase  Load  Open  Store  Utility  Zoom
```

Fig. 6. *User interface for an HP 64742A Emulator, showing the window capability.*

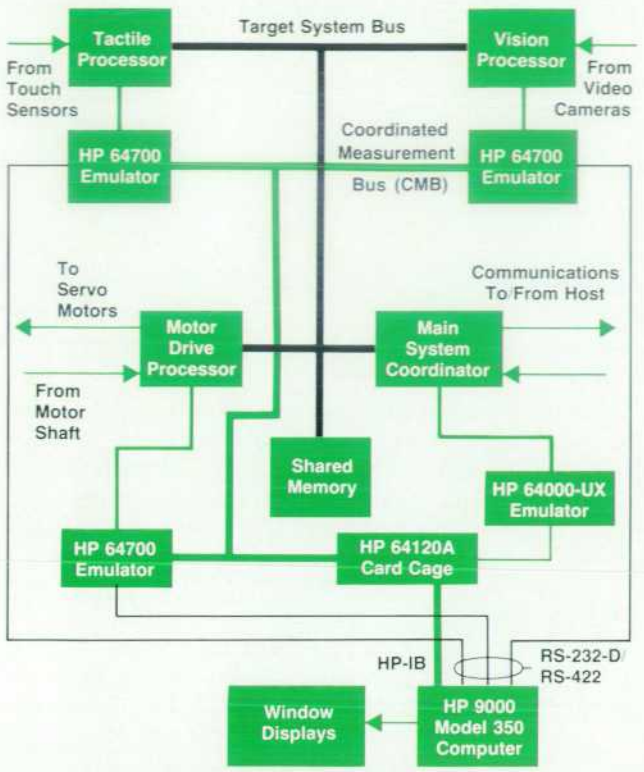


Fig. 7. *Emulation in a robotics application of a multiprocessor array.*

can be connected via this port, using inexpensive insulation-displacement ribbon cable, with a total separation distance of 50 meters. The CMB allows any grouping of emulators to participate in a coordinated measurement. This measurement can include synchronous starts and stops, internal or external analyzers coupled to multiple trigger sources, or triggering by activity occurring at another, remote emulator. Each emulator can be controlled by the same host computer or by independent hosts. Thus, for example, in a system with multiple processors accessing a common memory, fault conditions leading to an erroneous data transfer can be traced by triggering the analyzers of the two emulators at the detection of the fault, using the external state/timing analyzer to look at bus activity, and breaking the emulators to the background state to prevent any system or code corruption.

Fig. 7 is a simplified schematic diagram of one possible multiprocessor configuration, a complex industrial robotics application. The robot has communications links that allow it to share status and task programming information with other computers. Usually, satellite processors handle the peripheral chores, while a powerful central processor such as a 32-bit 68020 coordinates overall operation. In the example shown, each peripheral processor (such as an 80186) is replaced by a separate HP 64764A/L Emulator. These emulators are linked to each other via the CMB and to a 68020 emulator via the CMB/IMB link in the HP 64120A card cage. The entire system is controlled by an HP 9000 Model 350 workstation computer, and each emulator appears as a session on the high-resolution display. The HP 64120A card cage is linked to the Model 350 via the HP-IB (IEEE 488/IEC 625), and the HP 64700 emulators communicate via the RS-232-D/RS-422 serial link. With this configuration, some or all of the processors can be started or stopped simultaneously. Program flow events occurring in one part of the system can trigger analyzers somewhere else and perhaps shut down the system to prevent damage from, say, a robot arm out of control.

Acknowledgments

I would like to thank the members of the HP 64700 hardware team—John Hansen, Stan Bowlin, Don Logelin, Steve Peurifoy, Steve Kootstra, Jim Majewski, Bobby Self, Mike McCarthy, Bob Wardwell, and Andy Rodgers—for their creative efforts in making this product family a reality. Brad Harper and Bob Dockey helped to make the emulators manufacturable and, ultimately, sellable. Also, I want to thank Bill Fischer for helping to keep the sometimes tenuous hardware/software interface under control. Finally, I want to thank John Romano for providing an atmosphere that encouraged creative and outspoken engineers to do their best work.

References

1. A.S. Berger, "Foreground/Background Monitor Increases Development-System Transparency," *Personal Engineering and Instrumentation News,* Vol. 5, no. 2, 1988, p. 39.
2. *Electronic Products,* August 15, 1987.

Host Independent Emulator Software Architecture

Built into the firmware of the HP 64700 Series host independent emulators is an entire microprocessor development system.

by William A. Fischer, Jr.

FOR A MICROPROCESSOR EMULATOR, the system software is a significant portion of the overall design. HP 64700 Series emulators have, for the first time, an entire microprocessor development system built into firmware. This firmware lays the groundwork for future emulator design.

When a silicon manufacturer releases a new microprocessor, it is important that emulation support be available in a timely fashion. The present HP 64000-UX Advanced Integration Environment offers more than 40 different microprocessor emulators. A flexible software architecture was developed for the HP 64700 Series that also supports many different emulators. This article will discuss the HP 64700 software architecture. It will be shown how the software architecture improves the emulation development environment and how users benefit with easy-to-use, flexible emulation interfaces.

Software Architecture Overview

The HP 64700 software has a layered architecture. The layers consist of the processor specific drivers, the generic firmware, the terminal interface, the RS-232-D/RS-422 communication channel, the programmatic interface, and the host interface (see Fig. 1). The software layers have well-defined interfaces that permit easy communication between layers. These interfaces facilitated design and implementation without unnecessary concern for the detailed design of the other software layers.

The system design started with the terminal interface. The requirement for host independence (see article, page 45) necessitated development of an easy-to-use interface that connects to a simple ASCII terminal. The needs of the terminal interface user were some of the key system design concerns.

The terminal interface is supported by the generic and processor-specific firmware layers. The generic firmware layer is used by all emulation products. The processor-specific layer customizes the emulation product for a specific microprocessor. The terminal interface firmware is primarily generic, but commands for specific processors can be added.

The user interface residing on the host computer makes use of the terminal interface to perform its functions. Some terminal interface commands use an option mode to improve system performance by reducing the information passed over the RS-232-D/RS-422 communication channel.

The programmatic interface is the layer between the terminal interface and the host user interface. It resides on the host computer and consists of a library of C function calls that access the entire functionality of the emulation system. The programmatic interface is ported to multiple hosts and allows connection of user interfaces of different flavors including high-level software debuggers.

Terminal Interface

The terminal interface provides for communication between the emulation system and the outside world. Terminal interface commands are designed to be atomic, each performing only one function. Command names are one to five characters in length and form easily remembered mnemonic abbreviations of the emulation functions. Command options provide extra flexibility for the command functions. Command operands are used when additional data is required by the function.

For example, to display registers the simple command `reg` is used. With no operands, the command displays the entire register set. A single register can be displayed by following the command with an operand, the register name, as in `reg pc`. To modify a register the command might be used as follows: `reg pc=100h`.

The terminal interface command set consists of nearly 100 commands. Almost half of these commands are used to access the functionality of the emulation analyzer (see article, page 45). The other commands provide functionality

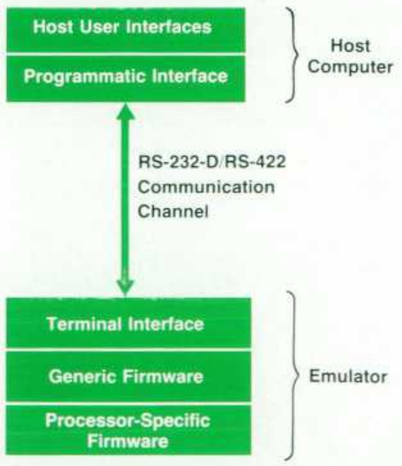


Fig. 1. *The HP 64700 emulator software has a layered architecture.*

for memory display, software breakpoints, emulation memory and target memory loading, stepping, running, and system functions.

To help the user with the sheer number of commands that must be remembered, the HP 64700 terminal interface contains an extensive help system. The help system provides detailed descriptions and uses for all commands. User access to the commands is simple and hierarchical so that users can work their way from generic help, through a listing of command groups, and finally to more thorough help for an individual command. See Fig. 2 for a display of help commands.

Terminal interface system commands are also available for displaying the time and date, controlling the communication port, and running performance verification tests. The HP 64700 Series also allows the user to define system macros, groupings of commands that can be executed by a user-defined name.

Firmware Structure

The firmware consists of generic core code and a set of processor-specific drivers and functions. This division allows easy partitioning of firmware tasks. The generic core code is universal. Only the processor-specific functions are developed for each new emulator. This minimizes the overall development time for a new microprocessor emulator.

The goals of the firmware design were to create maximum flexibility in the generic core code, total isolation of the processor-specific functions, and minimization of the processor-specific functions. Maximum flexibility allows the system to accommodate emulation of most conceivable types of processors. The total isolation of processor-specific

```
R>help

  help  - display help information

    help <group>          - print help for desired group
    help -s <group>       - print short help for desired group
    help <command>        - print help for desired command
    help                  - print this help screen

  --- VALID <group> NAMES ---
    gram     - system grammar
    proc     - processor specific grammar

    sys      - system commands
    emul     - emulation commands
    *        - all command groups


R>help emul

  emul - emulation commands
  -----------------------------------------------------------------------------
  b......break to monitor cov....coverage         mo.....modes
  bc.....break condition  cp.....copy memory      r......run user code
  bnct...bnct signal      dump...dump memory      reg....registers
  bp.....breakpoints      es.....emulation status rst....reset
  cf.....configuration    io.....input/output     rx.....run at CMB execute
  cim....copy target image load...load emul memory s......step
  cmb....CMB interaction  m......memory           ser....search memory
  cmbt...cmbt signal      map....memory mapper    x......emit CMB execute


R>help m

  m - display or modify processor memory space

    m <addr>                - display memory at address
    m -d<dtype> <addr>      - display memory at address with display option
    m <addr>..<addr>        - display memory in specified address range
    m -dm <addr>..<addr>    - display memory mnemonics in specified range
    m <addr>..              - display 128 byte block starting at address A
    m <addr>=<value>        - modify memory at address to <value>
    m -d<dtype> <addr>=<value>        - modify memory with display option
    m <addr>=<value>,<value>          - modify memory to data sequence
    m <addr>..<addr>=<value>,<value> - fill range with repeating sequence

  --- VALID <dtype> MODE OPTIONS ---
    w - display size is 2 byte(s)
    b - display size is 1 byte(s)
    l - display size is 4 byte(s)
    m - display processor mnemonics
```

Fig. 2. *A typical HP 64700 hierarchical help facility display.*

functions means that the firmware for a given emulator resides by itself in ROM on the processor-specific hardware board and that the functions are separately compiled. Most of the functionality of the firmware is in the generic code, reducing the quantity of processor-specific code needed for a new emulator.

The generic commands can be extended. An emulator may need a special command not developed in the original design. The processor-specific firmware can insert a new command into the generic command set or a host interface can download new commands via RS-232-D/RS-422. This makes the firmware extremely flexible for future but yet unknown emulation requirements.

Separating the generic core functionality from the processor-specific functionality means that the basic emulation functionality is handled by the generic core code, which makes calls to low-level drivers that are resident in the emulator ROM. The generic core code handles the parsing of commands and the basic emulator control. Calls are made to the processor-specific drivers whenever communication with the emulation hardware is needed to complete a function. For example, the user enters a command to display memory (m 0..20h). The generic core code parses the command and passes the ASCII string (0..20h) to an emulation function that converts the string to an actual address. The generic core code then passes the address to a processor-specific function that interacts with the emulation hardware to read memory and return the memory values. The generic core code will then display the value in the proper format.

The processor-specific functions are accessed through a table consisting of function calls. Other tables provide information about processor-specific registers, configuration items, the memory mapper, and error code information.

The data structures are also separated into generic and processor-specific pieces of information. Because of processor differences, each requires processor-specific data structures. Some of these processor differences include address, data, and registers. Functions that must differ for each emulator include emulation status, configuration, and miscellaneous functions.

Address

Physical address ranges differ widely with processor types. For example, the Zilog Z80 microprocessor has a 16-bit address range, the Intel 8086 has a 20-bit address range, and the Motorola MC68000 has a 24-bit address range. In addition, address mnemonics differ. The Intel family uses a segment:offset structure while the Motorola family uses a linear address space. Also, some processors like the MC68000 and the TMS 320C25 use the concept of multiple address spaces. For example, the MC68000 uses function codes to distinguish between user and supervisor space. These function codes become a logical extension to the address. See Fig. 3 for specific examples.

To handle the varying complexities of processor addresses, the concept of an address object was developed. An address object is a processor-specific structure that represents a location in the processor's address space. The address object structure contains all the pertinent information about that processor's address (see Fig. 4). The generic firmware never handles a specific address type but instead only passes pointers to these address objects. The lower-level processor-specific drivers interpret the address, translating ASCII string to address object, address object to ASCII string, address object to physical address, physical address to address object, or program counter to address object as appropriate. Adding an integer to an address object is another lower-level driver function.

Data

Data sizes also vary between processors. The MC68000 can access data lengths of 1, 2, or 4-byte words. The Intel 8086 can access data as 1 or 2 bytes but the data must be displayed in a swapped order compared with the MC68000. Since most processors access data as multiples of single bytes, data input and display are handled generically as an array of bytes. The ordering of the byte data for display and interpretation is left up to the processor-specific functions to specify for the particular processor being emulated.

It is also important to control the type of cycle that is used to access the data. The 8086, for example, can access data as bytes or words, and it may be necessary for peripheral devices to see the proper type of data access. The mode (mo) command is used to select the access mode for data. The options of this command are processor-specific and are handled in the drivers that access the data. These processor-specific access modes are defined in one of the processor-specific tables.

Similarly, it is important to display the data in the form that is most understandable to the user. Typical ways of displaying data are in bytes, words, or mnemonics. Again, the mode command is used to set the default for these processor-specific attributes. Alternatively, commands such as memory (m) and search memory (ser) permit the user to input an override option to change the display mode for that command and the default.

Registers

Every processor has its own set of registers. The register names vary, as do their lengths and functions. In addition, the registers may be read only, write only, or read-write.

```
        Processor             Address Example      Address Type

        Z80                    1000                linear
        MC68000                10000@sp            linear with function codes
        80186                  1000:23a5           segment:offset
        TMS320C25              1000@p              linear with function codes

note: @ indicates function code follows
      sp = supervisor program space for the MC68000
      p = program space for TMS320C25
```

Fig. 3. *Address representations for various processors.*

Some can reside in memory and others in I/O space.

Processor-specific tables are created for each emulator that fully define the valid register names, sizes, read/write status, and register grouping information.

Configuration

Each emulator requires a different set of configuration parameters. These configure the emulation hardware to provide the type of functionality that the user wants. Users need to decide if they want to use the target system clock or the internal emulation clock, or they might need to switch between a foreground monitor and a background monitor (see article, page 45). These configuration parameters are extremely important for normal functionality, and provide the flexibility for an emulator to work in a wide variety of target systems.

Each emulator establishes its own table of function calls for configuration parameters. The table contains the configuration name, the function to be called for that configuration, and the configuration help functions.

Status

An emulator must be able to control the transition to various states and determine the state in which it is currently residing. An emulator has three basic states: running user code, running in monitor, and static reset. The method each emulator uses to enter these various states can vary greatly from a simple toggle of a control bit located at a defined address in the controller's memory space to multiple writes to the hardware.

Determining the current status of an emulator may be as simple as reading a single hardware status bit or as complex as interrogating the monitor for detailed information. Processor-specific functions control the state transitions and read the current system state.

```
Address Object Length = 3
Word 1 = high 8 bits of address
Word 2 = low 16 bits of address
Word 3 = function code
```

Fig. 4. *Address object structure for the MC68000 processor.*

Miscellaneous

Each processor has its own instruction mnemonics. An inverse assembler is provided in firmware for each emulator. Also, functions are provided for loading the emulation monitor, the code that actually communicates with the target processor.

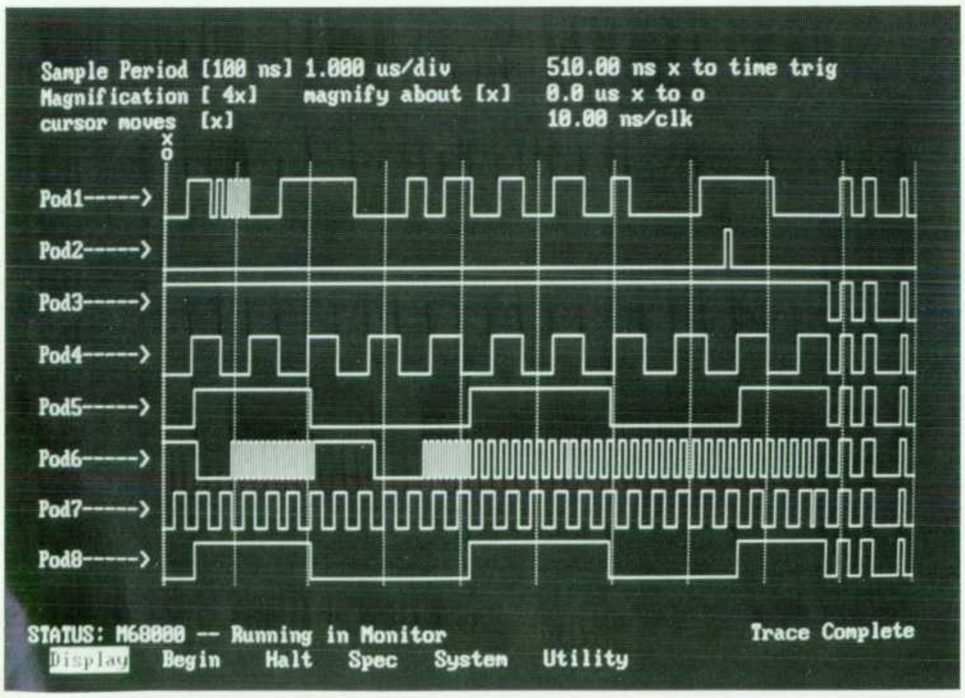
Host Interfaces

Host interfaces are programs resident on the host computer that provide the user's view of the HP 64700 Series emulators. Because emulation control firmware is resident in ROM on the emulator, it is reasonable and efficient to develop multiple host interfaces for each emulator.

Currently, host interfaces are supported for two different hosts: the HP 9000 Series 300 and the HP Vectra PC and IBM PC/AT compatibles. Two different interfaces are supported on the Series 300: an emulator interface using the same syntax-directed softkey design as the original HP 64000 and a new high-level software debugger. The interface on the HP Vectra PC provides a friendly emulation interface with a flexible windowing display system (see Fig. 6 on page 50). A timing analyzer interface is provided on the HP Vectra PC using a graphical display (see Fig. 5). All host interfaces provide a syntax-directed command structure, displays of emulator output, and an integrated symbolic package.

Programmatic Interface

The programmatic interface is the starting point for all host software. The programmatic interface is a library of C


```
Sample Period [100 ns] 1.000 us/div          510.00 ns x to time trig
Magnification [ 4x]    magnify about [x]     0.0 us x to o
cursor moves  [x]                            10.00 ns/clk
```

```
STATUS: M68000 -- Running in Monitor                 Trace Complete
Display  Begin   Halt   Spec   System   Utility
```

Fig. 5. *HP 64700 timing analyzer interface for the HP Vectra Personal Computer.*

function calls. A host interface program gains access to the emulation system via these calls.

The programmatic interface provides functionality for memory and I/O accesses, run control, register display and modification, software breakpoints, the coordinated measurement bus, and error message control. The programmatic interface hides the emulator's functional idiosyncrasies from the host interfaces. Thus, the designers of host interfaces do not need to understand the complete emulator behavior, greatly simplifying their task.

The programmatic interface also handles command and error message synchronization. The HP 64700 emulators will only respond with information such as data, an error message, or a status message in response to an entered terminal command. Updates of information are also returned after a null command (a simple carriage return). The only time the user receives information without requesting it is on power-up, when the copyright message and emulator hardware configuration and firmware revisions are sent. This methodology greatly simplifies the handling of information returned from the emulator.

The information returned from the HP 64700 Series can be either synchronous or asynchronous. A memory command, for example, returns the contents of the requested memory locations. If an invalid address is entered, an error message is generated and displayed. Both of these information types are synchronous with the last command entered.

A software breakpoint command (bp) establishes a condition that could result in the generation of an asynchronous message. The software breakpoint command replaces an instruction in the target memory space with the processor's software trap instruction. When that trap instruction is executed in the user's program, the control of the target processor is transferred to the emulation monitor program. The emulation control firmware, executing on the control processor, determines that a software breakpoint occurred, and a message is displayed indicating this. The software breakpoint will most likely occur long after the original software breakpoint command was entered. Because of this, the software breakpoint message will be returned with data from another command. This type of message is termed asynchronous.

The status and error commands are separated into synchronous and asynchronous buffers by the programmatic interface. Host interfaces can access these two buffers for display and error evaluation purposes.

The programmatic interface, like the firmware, uses the concept of an address object. The processor-specific dependencies are manipulated by a library that together with an expression evaluator and a symbol package evaluates all expressions of address objects. This permits the programmatic software to handle the expression in a generic fashion.

The programmatic interface also provides substantial control of the HP 64700 emulation analyzer. The terminal interface handles analyzer control with a large set of elemental commands. Simple analyzer control is easy, but setting up a complex analysis sequence requires the user to allocate all the analyzer resources manually. This requires an intimate understanding of the analyzer hardware and its programming model. The programmatic interface uses an expression tree with a validation function to determine if the analyzer contains enough resources to perform the measurement.

The programmatic interface also includes the communication port drivers. For MS-DOS these port drivers control the standard RS-232-D/RS-422 port on the HP Vectra PC and provide high-speed communication at up to 460 kilobaud using a new buffered interface card (HP 64037A). A communication port configuration file is read to associate a logical emulator name with a physical communication port on the host computer.

The PC Interface

The host user interface for PCs provides the user with a friendly windowing environment. Commands are entered by single keystrokes or command selection using the cursor keys. The PC interface takes the commands from the user and builds a set of data structures. These data structures are passed to the programmatic interface. The programmatic interface validates the data structures and generates the appropriate terminal interface commands. The commands are passed to the port drivers which send them to the HP 64700 emulator via an RS-232-D/RS-422 communication channel.

The data generated by the HP 64700 Series is received via the RS-232-D/RS-422 drivers. It is then processed by the programmatic interface and the data is put into data structures. The PC interface translates the data structures into ASCII for display in one of the windows.

Summary

The layered software interface for the HP 64700 emulators has succeeded in reducing the development time for new emulation products. The software contains well-defined interfaces that simplify software communication between layers. The terminal interface layer provides a host independent interface that is also used by all the host interfaces. Incorporating the entire emulation functionality in firmware has permitted easy development of more than one host interface, providing an emulation system that can be used as a hardware/software integration tool, a high-level software debugger, or a timing analyzer.

Acknowledgments

Much thanks goes to the software architects who devoted so many hours to the development of this system: Eric Kuzara, Jeff Downs, Cheryl Brown, Beth Vail Jones, Mike Gardner, Harold Shaw, Steve Warntjes, Rick Nygaard, Manfred Arndt, and Todd Bailey. A special thanks is given to John Romano who successfully guided us through this project.