HEWLETT-PACKARD JOURNAL

October 1991

HEWLETT PACKARD

HEWLETT-PACKARD
JOURNAL

October 1991

Articles

Departments

In this Issue

While cost containment is certainly the most publicly visible concern in medical care today, hospitals are under just as much pressure to improve the quality of patient care. Since they can invest in new equipment only every seven to fifteen years, hospitals want that equipment to be easy to upgrade and adapt to new technologies. In designing HP's fourth-generation patient monitoring system, the engineers at HP's Medical Products Group had to deal with these and other issues influencing their hospital customers' investment decisions. The design of the new system, which they call the Component Monitoring System, emphasizes modularity, flexibility, and ease of use while addressing the increasing need to measure new patient parameters and to process this information using powerful algorithms and data management tools. The article on page 6 introduces the system and its overall architecture, while details of the hardware and software architectures can be found in the articles on pages 10, 13, and 19. The necessary functions of data acquisition, parameter signal processing, display, and system connections are implemented as hardware and software building blocks. Application software modules, such as the blood pressure measurement software, can be arbitrarily assigned to any CPU in a loosely coupled multiprocessor system. The applications think that they are running on a nonexistent virtual processor. The architecture makes it easy to configure the system for both current and future needs in the operating room, intensive and cardiac care units, and other hospital areas, and contributes to a greatly simplified, low-cost production and test process. Patient parameters such as blood pressure, the electrocardiogram (ECG), blood gases, temperature, and others are measured by state-of-the-art modules that can be located close to the patient while the signal processor and displays can be in another room, if necessary. The ECG, blood pressure, and recorder modules are described in the articles on pages 21, 25, and 26. The system not only provides the clinician with the raw data measured by these modules, but also processes it, as explained in the article on page 40, to obtain many meaningful indicators of physiological functions, such as ventricular ejection and systemic vascular resistance. Ease of use is delivered by a thoroughly tested user interface design (see page 29), which can be localized easily for most languages (page 37). Mechanical design, software testing, and production and final test of the Component Monitoring System are the subjects of the articles on pages 44, 49, and 52.

The personal computer, or just PC, based on the Industry Standard Architecture (ISA) pioneered by the IBM PC, has gained steadily in processing power as each new generation of Intel microprocessors was introduced. The HP Vectra 486 PC uses not only the latest-generation microprocessor, the Intel486, but also the new Extended Industry Standard Architecture (EISA). The Intel486 integrates the CPU, a cache memory, and a math coprocessor onto one chip running at 25 or 33 megahertz. The EISA takes the 16-bit ISA bus to 32 bits while maintaining compatibility with all ISA I/O cards. The design of the HP Vectra 486 shows that designers can still contribute creatively within the constraints of industry standards. Among the design contributions are an architecture that incorporates all of the new technical features of the EISA and adapts easily to faster versions of the Intel486, a bus connector that accommodates both EISA and ISA cards, a burst-mode memory controller, a high-performance hard disk subsystem, and enhancements to the Vectra's basic I/O system (BIOS) to take advantage of all of the new features. An overview of the Vectra 486 design appears on page 69, and the articles on pages 73, 78, and 83 discuss the connector, memory controller, and BIOS designs. Performance analysis of many of the design concepts was done using a specialized hardware and software toolset, allowing the designers to make critical design trade-offs before committing to hardware (see page 92).

How much does a software defect cost in terms of unnecessary expense and lost profit? Why is it important to know? As Jack Ward of HP's Waltham Division explains in the article on page 55, if you're trying to justify the cost of a new software development tool, it helps to know what the tool will save by reducing software defects. He proposes an algorithm for computing the cost of software defects, applies it to a five-year database of software product releases, and shows that defect prevention and early removal can save a lot of money.

Code inspections are now standard procedure in many software development organizations. Are they effective? The article on page 58 describes the results of one HP division's effort to collect data to find out. There are both positive and negative findings, but the conclusion is that formal inspections are beneficial, while the value of informal inspections is still open to question.

R.P. Dolan
Editor

What's Ahead

The December issue will feature HP Sockets, a software tool for integrating applications in a network environment. An article from HP Laboratories in Bristol, England will introduce HP's formal specification language, HP-SL, and four articles will present examples of the use of HP-SL in software development. Another article will describe the HP Network Monitoring System for telecommunications networks using the 2-Mbit/s primary rate interface and the CCITT R2 or #7 signaling system. The 1991 index will also be included.

The Hewlett-Packard Journal is published bimonthly by the Hewlett-Packard Company to recognize technical contributions made by Hewlett-Packard (HP) personnel. While the information found in this publication is believed to be accurate, the Hewlett-Packard Company disclaims all warranties of merchantability and fitness for a particular purpose and all obligations and liabilities for damages, including but not limited to indirect, special, or consequential damages, attorney's and expert's fees, and court costs, arising out of or in connection with this publication.

Subscriptions: The Hewlett-Packard Journal is distributed free of charge to HP research, design and manufacturing engineering personnel, as well as to qualified non-HP individuals, libraries, and educational institutions. Please address subscription or change of address requests on printed letterhead (or include a business card) to the HP address on the back cover that is closest to you. When submitting a change of address, please include your zip or postal code and a copy of your old label.

Submissions: Although articles in the Hewlett-Packard Journal are primarily authored by HP employees, articles from non-HP authors dealing with HP-related research or solutions to technical problems made possible by using HP equipment are also considered for publication. Please contact the Editor before submitting such articles. Also, the Hewlett-Packard Journal encourages technical discussions of the topics presented in recent articles and may publish letters expected to be of interest to readers. Letters should be brief, and are subject to editing by HP.

Please address inquiries, submissions, and requests to: Editor, Hewlett-Packard Journal, 3200 Hillview Avenue, Palo Alto, CA 94304 U.S.A.

Introduction to the HP Component Monitoring System

This fourth-generation patient monitoring system offers a set of hardware and software building blocks from which functional modules are assembled to tailor the system to the application and the patient.

by Christoph Westerteicher

Over the past twenty years HP has been a supplier of patient monitoring equipment to the healthcare industry. Patient monitors are observational and diagnostic tools that monitor physiological parameters of critically ill patients. Typical parameters include the electrocardiogram (ECG), blood pressure measured both invasively and noninvasively, pulse oximeter (SaO_2), and respiratory gases, among others. The catalog of parameters is still growing based on the need for better patient care and the technical feasibility of new measurement techniques.

Patient monitors are used in a variety of departments within hospitals. These include operating rooms, intensive care units, cardiac care units, in-hospital and out-of-hospital transportation, and special function areas such as lithotripsy and x-ray. A patient monitoring system must be versatile and applicable to most of these areas. This means that it must support a wide range of configurations and allow quick adaptation to the patient-specific level of care. For a normal appendectomy, monitoring the ECG, noninvasive blood pressure, SaO_2, and one temperature will suffice. At the other extreme, during a cardiovascular operation as many as eight different physiological parameters will be measured.

The HP Component Monitoring System is designed to meet these requirements. This article outlines the high-level project goals and the approaches taken to meet them. It also describes the overall hardware and software architecture of the HP Component Monitoring System. Subsequent articles in this issue highlight the technical contributions of the Component Monitoring System project in more detail.

Design Goals

The HP Component Monitoring System is the fourth generation of patient monitors to be designed and built by the HP Medical Products Group. Based on our experience, current customer needs, and expected future trends in the medical field, two objectives were viewed as areas in which HP could make a major contribution. One is the area of modularity and flexibility and the other is ease of use.

Modularity and Flexibility. The monitor is composed of the following functional modules:

- Data acquisition
- Parameter signal processing
- Monitor control and data input
- Display
- System connects.

Each of these functional modules is implemented in a set of hardware and software building blocks, which as a whole form the Component Monitoring System depicted in Fig. 1. Separating the monitor into its generic elements provides many advantages. First, the monitor can easily be configured to best meet the application needs of the individual customer. Parameters can quickly be combined according to the required level of care and changed when necessary. Second, adding functionality to a monitor is as simple as inserting the appropriate hardware into an existing unit and updating the software if necessary.

A third advantage is that the Component Monitoring System can be kept abreast of new technological trends by

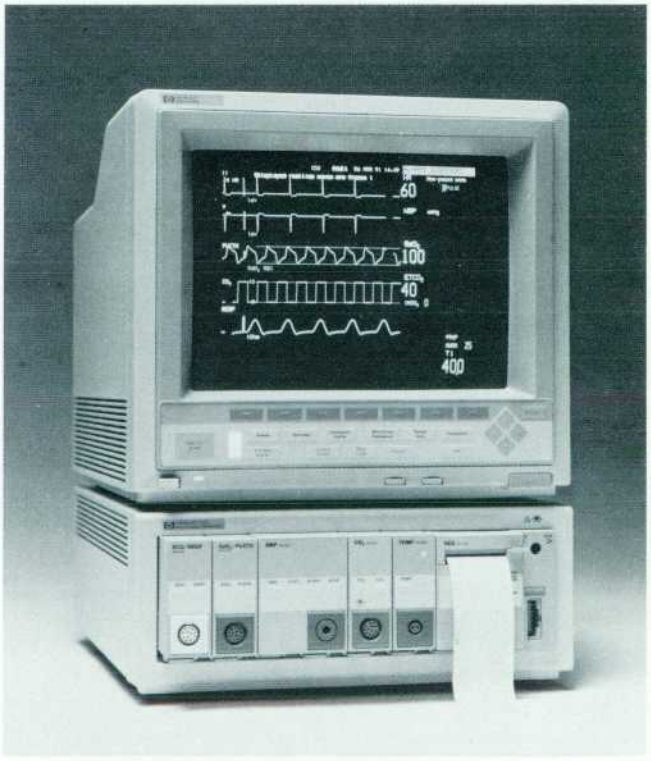


Fig. 1. In the HP Component Monitoring System, a fourth-generation patient monitor, each functional module is implemented in a set of hardware and software building blocks.

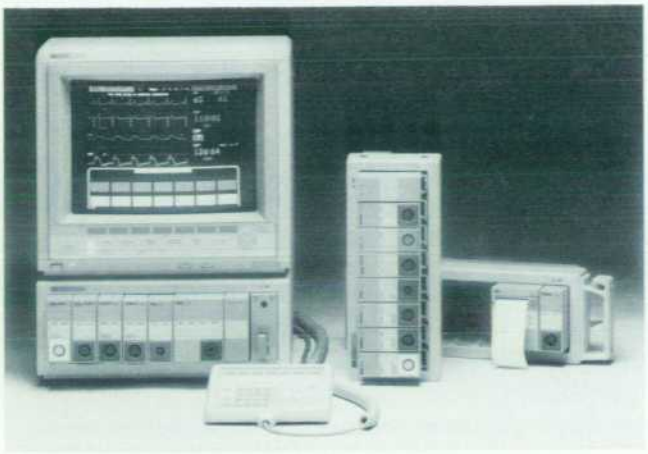


Fig. 2. The Component Monitoring System architecture has three segments: the module rack and parameter modules, the computer module, and the displays. The module rack can be either an integral part of the computer module or totally detached. A remote keypad is optional.

enhancing or redesigning the appropriate functional element. Implementation will only affect one building block, and will be fully backward compatible with existing systems.

Finally, production has been dramatically simplified. Customization of each monitor is the last integration step in production. Thus, all components can be assembled and tested without knowing the specific configuration in which they will be used.

Flexibility is enhanced by designing the monitor components so that their physical location can be optimized to address ergonomic considerations and by allowing the user to program the monitor's default settings and standard configuration. This means that the monitor can be adapted to a wide range of current and future clinical applications.

Ease of Use. Ease of use is of particular importance for patient monitors in operating rooms and critical care units, where clinicians use patient monitors to make informed decisions about potentially life-threatening situations. In the past, clinicians have had to strike a compromise between the desired functionality of a patient monitor and its ease of use. Our goal was to make this very sophisticated piece of equipment truly intuitive for doctors and nurses to use. Other areas that we focused on, and that played an important role during the development phase were:

- Implementation of methods to meet HP quality goals.
- Minimization of production costs and support for a linear cost profile. This means that functionality can be segmented down to its generic building blocks. Should a particular feature be needed, the customer pays for it and nothing more.
- Standardization, ranging from uniform design tools and software development environments all the way to minimizing the number of different electrical components used in the Component Monitoring System as well as the number of mechanical parts needed to assemble the unit.

System Architecture

From an architectural standpoint the Component Monitoring System can be divided into three segments (see Fig. 2):

- Module rack with parameter modules
- Computer module
- Displays.

The module rack and parameter modules represent the interface to the patient. Each parameter module is dedicated to the measurement of one or more physiological signals, and is housed in a separate enclosure. Within the parameter modules, the transducer signals are electrically isolated from ground potential, amplified, sampled, and converted from an analog to a digital format. The digital parameter values together with the status of each module are polled at fixed intervals and sent to the computer module for further interpretation.

Up to eight single-width modules fit into one module rack. The module rack can be either an integral part of the computer module or totally detached, in which case it would be called a satellite module rack. The satellite module rack is connected to the computer module by an umbilical-cord-like cable, which carries both the digital signals and a 60V dc power line for the parameter modules. One computer module can support as many as four satellite module racks. This concept allows the user to position the parameter modules as close as possible to the patient, where the signal is measured. The transducer cables can thus be kept short, minimizing the amount of wiring as well as the tendency for it to become tangled or draped over the patient.

Computer Module

The computer module is the main processing unit. It consists of a cardcage that can house up to 23 function cards and one dc-to-dc converter (Fig. 3). Function cards currently available include CPUs, memory cards, interface cards, display controllers, and a utility card. For the first release, a total of 11 function cards were designed. The interconnection within the cardcage takes place via the central plane, a motherboard located in the middle of the chassis with press-fit connectors mounted on both sides

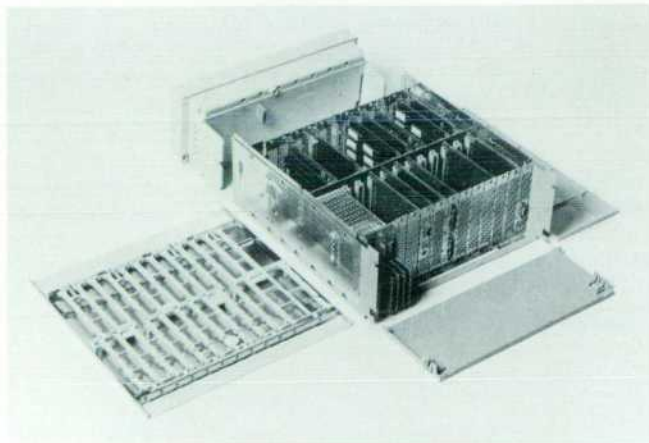


Fig. 3. The computer module houses up to 23 function cards.

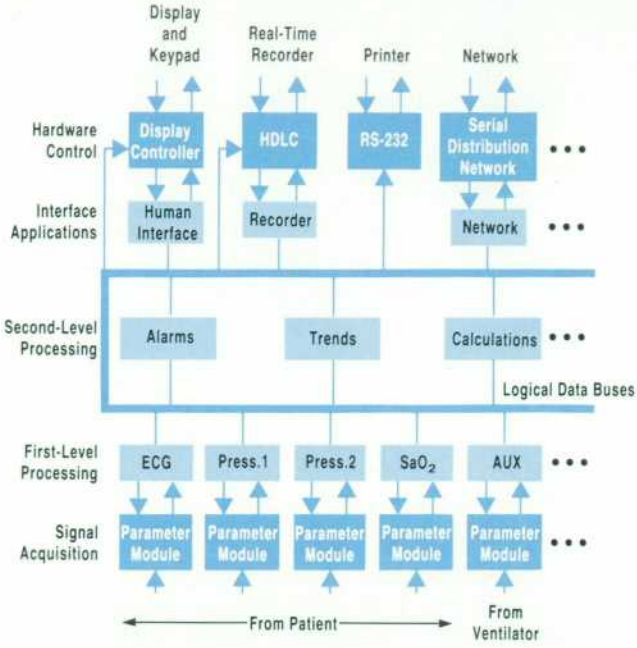


Fig. 4. Functional block diagram of a Component Monitoring System patient monitor. The shaded boxes are software functions on the CPU card processors. The solid boxes are specific hardware and firmware functions.

of a printed circuit board. Data exchange between the function cards takes place on the message passing bus. This bus is routed to all 23 slots on the central plane, allowing a high degree of freedom as to where a function card can be inserted. The message passing bus is the backbone of the Component Monitoring System. Many of the goals listed above only became possible with the help of this communication concept.

The basic function of the message passing bus is that of a broadcast system. Each message sent on the bus consists of a header, which describes the content, and the actual data. A source (e.g., a CPU card) will obtain control of the message passing bus and transmit its information. The data is not transmitted in a point-to-point fashion from one source to one receiver. Instead, message passing bus data is transmitted without any specific destination, and it is up to the function cards to watch for the information needed by their applications. As soon as a card detects a match between a header it is looking for and the header of the message on the bus, it automatically pulls this data into an internal stack.

The activities of bus arbitration, transmission, header matching, and data reception are controlled by the message passing bus chip. One of these interface chips is located on each function card that actively takes part in the communication process. The chip was designed specifically for the Component Monitoring System. It also was the first HP production ASIC (application-specific IC) to be designed using a silicon compiler tool.

A more detailed description of the message passing bus concept and the design of the interface chip can be found in the article on page 10, which covers the Component Monitoring System hardware architecture.

Displays

The customer can choose either a monochrome or color high-resolution display. Multiple independent displays can be used to present different sets of information to specific user groups. For example, the surgeon needs a different presentation of patient information than the anesthesiologist during surgery.

Physically separating the display from the computer module gives the user a choice of screen sizes and the possibility of mounting the computer module at a remote location when space next to the patient is at a premium.

The user interacts with the monitor through a combination of hardkeys and softkeys on the display keypad or through a remote keypad which functionally duplicates the keys on the display bezel.

Software Modularity

The concept of a modular system also applies to the software architecture (see Fig. 4). Application-specific modules represent the basic building blocks out of which the total solution can be assembled. The ECG application, for example, including the signal interpretation, alarm handling, and control interaction, is all encapsulated in one module. To the surrounding environment these application software modules are totally self-contained packages, and only exchange information with one another via the message passing bus. By virtue of this concept, it is possible to link each module as an independent entity with any of the other modules and assign it to one of the Component Monitoring System CPU cards.

A more detailed description of the software architecture can be found in the article on page 13.

All of the Component Monitoring System software is stored on EPROM function cards. These cards are physically located next to a CPU, and the applications running on that CPU execute directly from the adjacent memory card. All other CPU cards in the monitor get their application software downloaded into the on-board RAM during boot time. The advantage of this solution is that installing software is as easy as inserting one EPROM card.

Summary

The Component Monitoring System has proved that the concept of component modularity can be extended far beyond the mere ability to mix and match parameter modules. Modularity in this system means that the customer can tailor the patient monitor to best fit the application all the way from the parameters that need to be registered to the displays and interfaces the system should incorporate. The Component Monitoring System can also grow with the user's needs over time, and thus secure the hospital's assets for many years.

The success of the modularity concept is reflected in the fact that some of the hardware and software elements have found their way into other medical devices manufactured by HP. Overall, the Component Monitoring System architecture has proven it can function as a monitoring platform for years to come.

(continued on page 10)

Medical Expectations of Today's Patient Monitors

HP has been a supplier of patient monitors since the mid-1960s. The HP 7830, HP 782xx, HP 7835x, and HP 7853x monitors have measured vital signs like ECG, blood pressure, body temperature, carbon dioxide, inspired oxygen, and others.

Over the years, customer demands and measurement technology have both developed. Simple ECG monitors have gradually become complex monitoring solutions, acquiring, processing, displaying, and storing many parameters. These modern monitors need to communicate over dedicated LANs, both with one another and with central stations. For this purpose, several years ago, HP introduced a serial distribution network (SDN), which makes it possible to transmit a multitude of parameters, high-resolution waveforms, and other information to as many as 32 participants in a synchronous way. These features are not easily achievable with modern asynchronous LANs.

Medical Expectations

Most of today's patient monitors do not satisfactorily fulfill many of the current and future expectations for these systems. The increasing need to measure new parameters and to postprocess this information using powerful signal processing algorithms and data management tools require a different approach. At the same time, our customers are under tremendous cost containment pressure, allowing them to invest in new equipment only in cycles of seven to fifteen years. Having these and other customer needs in mind, we launched a development program for a new patient monitoring system. The goals of our project were to:

- Develop a solution that can be easily adapted to our worldwide customers' medical as well as financial needs.
- Develop a user interface that allows the user to control the system easily through different means including softkeys, touch, and remote keyboards (see article, page 29).
- Develop a solution for all major languages, including Asian languages.
- Develop a solution that is backwards compatible with the existing LAN (SDN) and future LAN implementations.
- Develop a solution corresponding to our customers' space restrictions. This often means acquiring measurement data close to the patient to avoid the so-called "spaghetti syndrome"—too many cables around the patient—and processing and displaying the measurement data farther away from the patient.
- Develop a solution that allows additional CPUs to be added to the system according to the signal processing needs, and that provides independent displays that can be addressed directly.

Conclusion

The HP Component Monitoring System is our solution to these needs. Fig. 1 shows the bedside monitoring concept of this system. Expansion is possible both horizontally and vertically in this matrix. The horizontal axis shows the various functional needs, while the vertical axis shows various ways in which these needs are met in specific applications. The last row of the matrix shows examples of possible future capabilities that can be added easily to the Component Monitoring System if and when they become available.

The HP Component Monitoring System is an ultraflexible patient monitoring system that can be adapted to almost all monitoring needs that arise in hospitals worldwide. Its built-in modularity and an HP proprietary message passing bus make it independent of technology changes in microprocessors, LANs, or displays. This is expected to result in a very long product lifetime, thereby protecting our customers' investments. By offering different configurations of the Component Monitoring System, we provide a wide price/performance range, and through a flexible upgrade strategy, we ensure high adaptability to our customers' future needs.

The Component Monitoring System is a truly international development that involved engineers in the U.S.A. and in Europe. Today, it is being manufactured at two HP sites, one in Europe and one in the U.S.A.

Such a significant technology step can never be taken without encountering challenges. Thanks to the engagement and dedication of all of the team members, these challenges were met successfully.

Frank Rochlitzer
General Manager
Surgical and Neonatal Care Business Unit

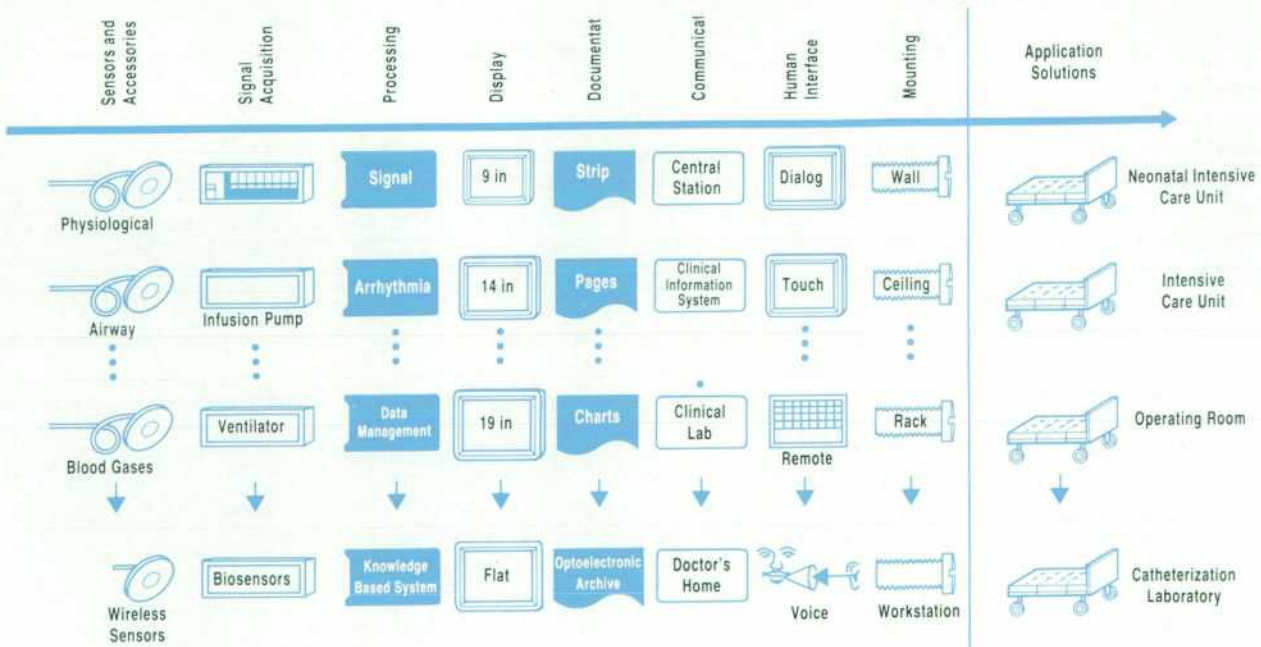


Fig. 1. The bedside monitoring concept of the HP Component Monitoring System.

Acknowledgments

The design of the Component Monitoring System has been a team effort involving at times up to 50 engineers located at the Waltham and Böblingen Medical Divisions. All of these individuals and many others deserve recognition for their hard work to make this product a success. I would also like to thank all who contributed to this series of articles on the Component Monitoring System. Besides the authors of each chapter, and the numerous unnamed supporters, one person deserves particular recognition: Heike Schreiber. Her continued enthusiasm and devotion helped to make this issue of the HP Journal possible.

Component Monitoring System Hardware Architecture

Up to 23 function cards residing in a computer module communicate over a message passing bus. The computer module, the display, and the parameter modules that measure vital signs can be in separate locations as needed by the application.

by Christoph Westerteicher and Werner E. Heim

The prime objective in the development of the HP Component Monitoring System was to build a patient monitor that would adapt optimally to the majority of clinical applications, now and in the foreseeable future. To the R&D team, this meant modularity, but not just in the sense of being able to mix and match parameters. The goal of this project was to carry the idea of configurability a quantum leap into the future.

Major Parts

The Component Monitoring System can be segmented into three parts (Fig. 1):

- The rack and parameter modules
- The computer module
- The display.

This segmentation is not just a theoretical way of looking at the Component Monitoring System. The system can actually be separated into these components. It is therefore possible to place the parameter modules close to the patient and position the display within sight of the anesthesiologist, while the computer module can be totally removed from the vicinity of the patient.

Computer Module

The computer module incorporates all the processing power, the interfaces to other devices and networks, the display controllers, and drivers for human interface equipment.

These functional elements have been broken into their generic components, and then designed and implemented as individual function cards. The processing power, for example, is provided by an application dependent number of CPU cards, working together as a loosely coupled multiprocessor system. Based upon a 16/32-bit microprocessor, each CPU card is an independent subsystem, includ-

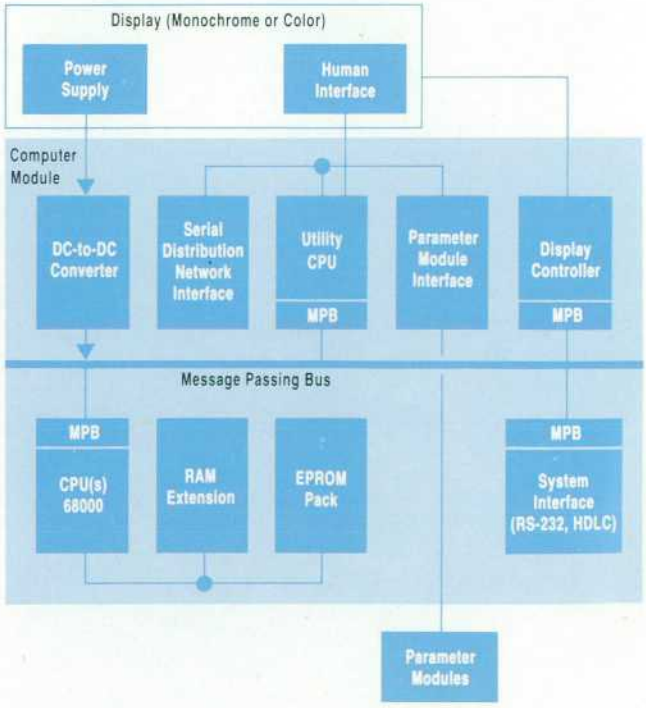


Fig. 1. Block diagram of the Component Monitoring System hardware architecture.

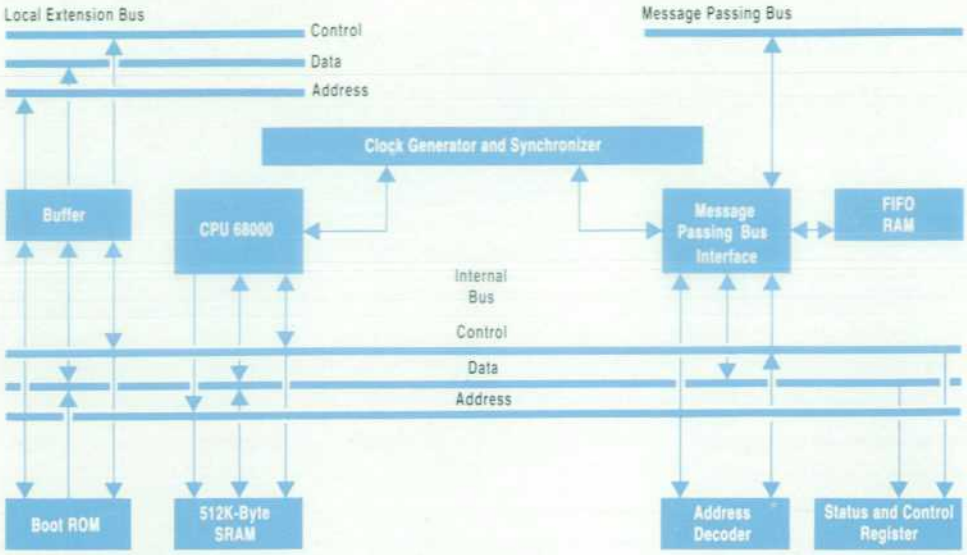


Fig. 2. CPU card block diagram.

ing a large amount of static memory, boot EPROM, and an interface chip for the computer module's internal bus, the message passing bus (Fig. 2). The ability to work as a self-contained, independent entity was the smallest common denominator we wanted to apply to our computer module building blocks. Because of this concept, processing power, interface cards, or display controllers can be added depending upon the customer's application. New function cards can be added by plugging them into the computer module without interfering with the existing configuration.

Local resources of a function card, like static memory or EPROM, can be extended by adding a battery-buffered static RAM card and an EPROM card. They connect to a local extension bus, which is routed on the same connector as the message passing bus, thus allowing an identical design for all slots on the backplane.

At first release of the Component Monitoring System, there are 11 function cards. The spectrum includes the above-mentioned processor and memory cards, interface cards to RS-232 devices and HP's medical signal distribution network (SDN), high-resolution monochrome and color display controllers, and other cards.

Each function had to fit onto the standard function card. To make this possible, several application-specific integrated circuits (ASICs) provide high performance in a minimum of card space. Surface mount technology allows components in very small packages to be mounted close together directly on the surface of a function card. The benefits of these new technologies are highly automated production processes, reduced part count, and increased reliability.

Message Passing Bus

The message passing bus represents the backbone of the Component Monitoring System. It is by virtue of this solution that it was feasible to implement modularity in such an extensive fashion.

The message passing bus is based on a message broadcasting system, in which one bus participant transmits information without having to specify the address of the receiving device. Instead, every message is classified by a header, indicating the content of the message. A bus participant interested in a specific class of messages writes the header of the information and a priority into the signature RAM of its message passing bus interface. When the header of the message on the bus and the header in the signature RAM match, the receiving card's message passing bus interface automatically loads that message into its FIFO buffer. Depending on the priority assigned, the incoming information is pushed into either the high-priority or the low-priority FIFO, thereby preprocessing data for the CPU. Comparing headers and moving information into and out of the FIFOs is controlled by the message passing bus interface chip with no interaction from the data processing device. Fig. 3 is a block diagram of this chip.

The major advantages of this concept are threefold. First, messages only need to be sent once, regardless of the number of bus participants interested in the information. This guarantees that bus bandwidth is not used merely to

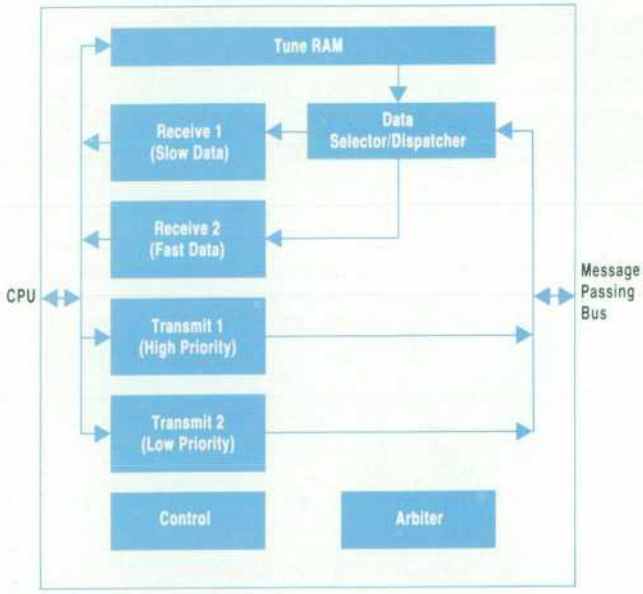


Fig. 3. Message passing bus interface chip block diagram.

duplicate messages going to multiple receivers. Second, the absolute bus bandwidth is defined by the speed of the message passing bus interface chip. The chip controls the transmit and receive FIFOs and compares headers without support from the data processing device. In essence, the message passing bus chip is a buffer between a high-speed bus and data processing logic working at varying speeds, which should not be interrupted by every bus activity if it is to function effectively.

The third major advantage of the message passing concept is that new system cards can be added to a monitor, and as long as they know the algorithm for allocating headers, they can actively participate on the message passing bus. The bus has a decentralized arbitration algorithm, which determines how each participant accesses the bus. Each interface chip incorporates arbitration circuitry based on a round-robin-like system, assigning the bus on a rotating priority basis. If the current bus master is level n, priority n−1 will be given to the next interface requesting the bus, and so on sequentially. This guarantees that the bus is shared equally among all of the function cards and is dynamically distributed every time a message is sent.

The message passing bus chip was designed as an ASIC, using a silicon compiler tool to develop and simulate the circuit's functionality.

Central Plane and Power Concept

In the center of the computer module chassis is a 23-connector motherboard with press-fit sockets mounted on both the front and the rear. The message passing bus is routed to all 23 slots on this central plane (Fig. 4).

Since all of the function cards are mechanically identical in size, any card can be inserted into any slot of the central plane, thus making possible a wide range of configurations. The only exemption is the dc-to-dc converter, which always is located in the same slot. This one card provides the power for the computer module, and is sourced with 60V directly from the Component Monitoring System display (Fig. 5).

This somewhat uncommon power architecture was necessary to comply with the stringent leakage current limits imposed on medical equipment. If the Component Monitoring System were to incorporate separate power supplies in the display and the computer module, the leakage currents of both to ground would be added together, making it very difficult to reduce this value to below 100 μA.

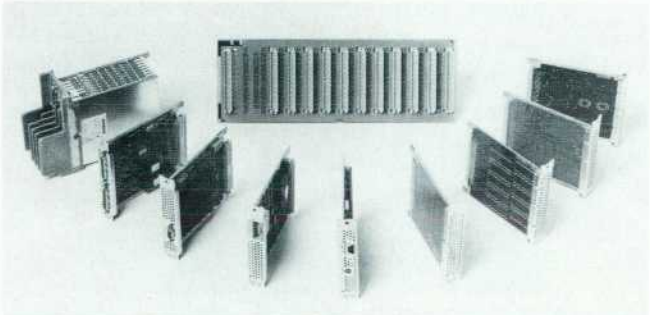


Fig. 4. Computer module central plane and function cards.

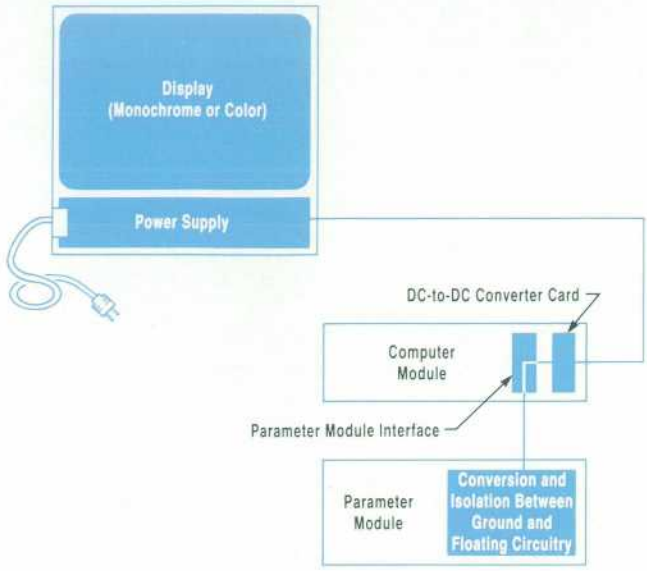


Fig. 5. Power concept. The single power supply is housed in the display.

The second reason for taking the power supply out of the computer module was the need to reduce the amount of heat dissipated in this small box to a minimum, so as not to jeopardize reliability. We therefore decided to have only one power supply for the entire Component Monitoring System, and to house this in the display.

The Display

Customers can choose either a monochrome or color 14-inch display (Fig. 6).These are high-resolution, noninterlacing, high-contrast displays designed and built specifically for medical applications.

To provide outstanding waveform quality, the displays and the display controllers have a very high horizontal resolution of 2048 pixels. At this pixel spacing the human eye can no longer resolve the individual dots, so the curves appear very smooth.

The displays also incorporate the Component Monitoring System control panel, located beneath the screen. This control panel is the main means of interacting with the monitor. It consists of a membrane keyboard, LED indica-

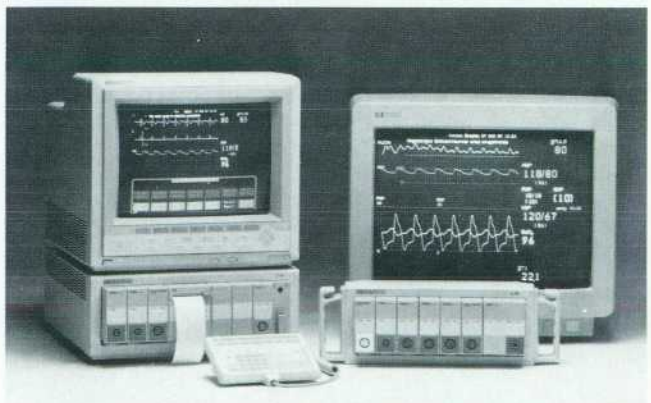


Fig. 6. The display also incorporates the control panel.

tors, and the human interface board, which is an HP-HIL device looped through to the computer module.

Summary

The hardware architecture has proven to be one of the steps on the ladder to success of the Component Monitoring System. With the advent of this new monitor, production has been automated and streamlined to an extent unheard of for such a complex device as a patient monitor. The parts standardization effort has resulted in a mere 300 different items for the entire system. Our customers can now have a state-of-the-art monitoring system that they can configure to their specific needs, and at the same time be assured that their system has been designed to stay abreast with technological or application changes for many years to come.

Acknowledgments

For their outstanding results, for their endurance, and for being a most enjoyable team to work with, our thanks go to all members of the Component Monitoring System hardware group.

Component Monitoring System Software Architecture

A modular design leads to a complex but easily manageable system that ensures economical resource utilization.

by Martin Reiche

HP Component Monitoring System patient modules can be mixed and matched to suit the application. A module is added simply by plugging it into any free slot in the module rack. Wouldn't it be convenient to handle all functions implemented as software the same way? Just find a free resource on any CPU card and assign the required set of software building blocks to it. Use only as many CPUs as necessary. This article will show that this approach is not only viable, but also appropriate in terms of both development economics and resource utilization.

The basic idea of having building blocks with standardized interfaces that can be arbitrarily combined has proven its power in many projects. The Component Monitoring System patient signal acquisition system, computer module, and message passing bus concept reflect this idea well.

This approach should also be promising for software. However, a problem with software is its complexity, both internally and in terms of interaction with external entities. Component Monitoring System software modules show significantly different profiles in resource requirements, must share a multiprocessor real-time system in varying configurations without conflicts, have to act and communicate in a meaningful way with regard to the current configuration, and are implemented by different people in different places at different times. This makes standardization difficult.

We will show how these problems were overcome, both from an architectural point of view and from a development environment perspective. As we proceed, we will encounter a continuously recurring question in different contexts: How can we provide the needed creative freedom for each individual, and at the same time manage their cooperation and integration into a coherent total solution?

Layered Software

As can be seen in Fig. 1, the Component Monitoring System's functionality can be represented in a layered scheme. Similar to existing computer systems, the hierarchy has four levels:

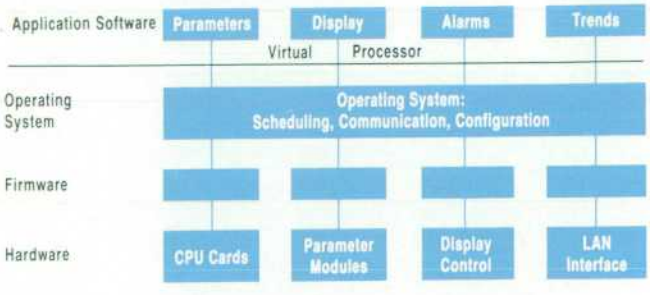


Fig. 1. The basic task allocation scheme of the Component Monitoring System uses a layered software structure. The lower-layer activities are transparent to the application software modules, which are designed to run on a virtual processor.

Component Monitoring System Software

Total amount of source code:	315 KNCSS
Number of software modules developed	30
Number of module instances in an instrument	43
Number of message passing bus headers allocated by the operating system	880
Average data flow on the message passing bus	50 kbytes/s

- The CPU cards represent the basis for all data processing. They communicate over the message passing bus. All interfaces to external devices are found here.
- Firmware located on these cards provides services to the higher layers of the model. The firmware implements complex application independent functions by converting commands and protocols into hardware related signals. Patient parameter modules play a special role: controlled by firmware, their analog and digital hardware converts the incoming patient signals into digital information accepted by the computer module.
- The operating system establishes the data paths between the application software modules on the one hand and the firmware on the other. It controls the execution of the application programs, moves messages back and forth, and continuously supervises the correct execution of all functions.
- It is up to the applications to provide the signal interpretation, computation, alarm generation, and similar functions and to support user interaction by drawing windows and menus on the screen. How functions are implemented in the lower layers is hidden from the application software modules.

Modules and Messages

All function cards are designed so that they can be arbitrarily combined over the central plane. They can transmit and receive messages and perform their functions regardless of the slots in which they reside.

In a straightforward extension of this principle, the Component Monitoring System software architecture allows for arbitrary distribution of software to the various processor cards (see Fig. 2). These self-contained application software modules are the building blocks of the modular system. Each module represents a large functional area—for example, the signal processing for the blood pressure measurement with its affiliated aspects of alarming, control interaction, transducer calibration, and so on.

To achieve this modularity, the current configuration is made transparent to the modules. They will execute on any CPU card, and their sharing of a CPU card with other modules will not interfere with their operation. The only way they can communicate is to transmit and receive messages. As with the message passing bus, the origins and destinations of these messages are hidden from the application.

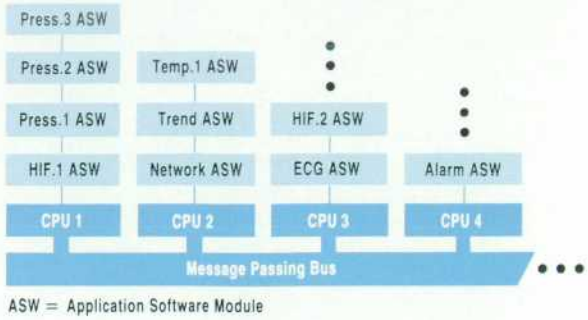


Fig. 2. Application software modules can be assigned arbitrarily to the available CPU cards to achieve the most economical resource utilization.

This principle guarantees maximum flexibility in reaching the required functionality with a given hardware set. It also reduces development risks, since at the beginning of a complex project, neither the sizes of the modules nor the resulting processing requirements can be accurately estimated.

Inside a Module

From a programmer's point of view, a module is composed of a number of related C-language source files (see Fig. 3). There are different types of files. For example, PROG files contain executable code and variable definitions, and TEXT files consist only of character codes.

Following a tailored syntax, an ASCII file called a module table provides comprehensive information about that module. Identification, communication behavior, execution, and resource requirements are specified in this table. This file is converted to C source code by means of an HP-developed compiler called mtc. In this way, a generic module

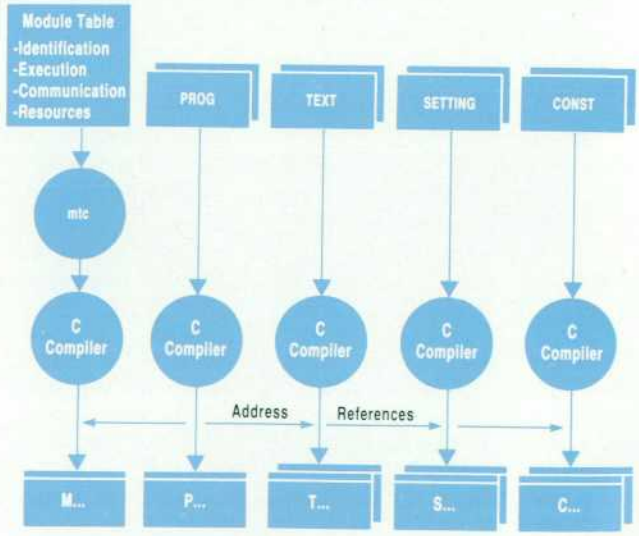


Fig. 3. An application software module consists of a module table and a set of C source files. A proprietary compiler, mtc, converts the module table to C source code.

structure is defined, along with a formal description of interfaces among the modules and the rest of the system. These standardization rules are followed by every software designer, specify the guidelines for all interaction, and lay the foundation for a fully automated code generation process. Thus, machine-processed specification *enforces* standardization.

All C code is then compiled and linked, yielding a number of object files, which are loaded unchanged into an EPROM card to be plugged into the computer module.

Virtual Processor

The main characteristic of this software concept is that the current configuration is invisible to all of the applications. They do not know which modules are assigned to which processor, how many processors are available, which interface cards are present, or where messages come from and where they go. Therefore, software developers must not make any assumptions as to where their applications will be executed. The only way modules are allowed to communicate with each other is by exchanging messages.

These are prerequisites for a truly modular system in which applications can be mixed and matched according to a predefined functionality. With this in mind, it is appropriate to define an abstract programming model that we call a *virtual processor* (see Fig. 1). This is a collection of artificial resources such as application priorities or data links. The virtual processor supplies the application programmer with all of the construction elements necessary to implement functions effectively and straightforwardly. The programmer can write functions that process

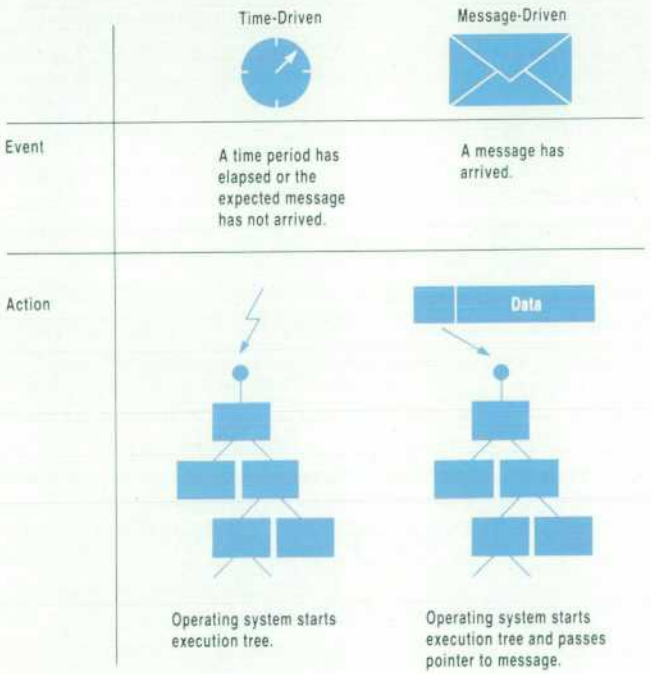


Fig. 4. Execution trees can be started after a certain amount of time has elapsed or when a certain event occurs.

Component Monitoring System Software Development Environment

All development was performed on HP 9000 workstations running under the HP-UX operating system and connected by a local area network. This includes the operating system, the development tools, and all applications and documentation. Each workstation had mass storage and emulation facilities (HP 64000) and could be tailored to specific needs.

Starting out with only a few developers on a single HP 9000 Series 500, we ended up with about ten HP 9000 Series 300 systems being used by 20 software engineers toward the end of the project.

Thanks to the power and flexibility of these HP-UX systems, the continuous evolution of the development environment proceeded very smoothly with respect to both its extent and its comprehensiveness. For example, all generic data (e.g., symbol tables for message specification or tools for process automation) was automatically updated on all machines. As a consequence, at any point in time, identical processes were used project-wide—a prerequisite for smooth integration of the components.

Since all application modules have the same form, it was possible to implement the standard generation processes only once. Besides the definition and evaluation of the module table, a uniform process supports the implementation of any application. This turned out to be very helpful, for despite the different functions of the modules and the various inclinations of the development engineers, one always finds recurring features in any implementation. This makes software maintenance easier for engineers other than the original programmer. For example, different language options of a module can be generated without touching any code.

In our environment, a single engineer had complete responsibility for specifying, designing, and implementing each software module. The activities of all of the developers had to be decoupled as much as possible. Since enforced synchronization would have been intolerable, a major requirement for the development environment was to support easy generation of running versions at all involved workstations. Here the Component Monitoring System's self-configuration, implemented as the boot process, proved valuable. All software modules are self-contained and independent. During the boot process the modules are initiated within the current run-time environment. This environment can always be tailored to meet the needs of the module under construction.

The integration process is always executed the same way. The current versions of the operating system and other modules are collected from the workstations on the LAN and are loaded into the current work environment. A configuration table then assigns the modules to CPUs in such a way that each module under construction runs on an emulated CPU. Symbol tables can then be corrected beforehand to allow for symbolic access to all variables and functions.

In summary, the extensive effort invested in the development environment and boot process has been very beneficial to the entire development and maintenance process, as well as to the product's quality. Other projects leveraging from the Component Monitoring System platform have profited and will continue to profit from this comprehensive and comfortable development environment.

messages without having to pay attention to how the information is distributed. Concepts such as interrupts, task control blocks, and the message passing bus chip can be ignored by the programmer, who is free to concentrate on the medical application. Thus, the application programmer's task is to convert the specific functions into programs that can run on the virtual processor, and the operating system's task is to support this program module by means of the current hardware configuration and ensure that any two applications on a single CPU do not interfere with each other.

The features of the virtual processor are defined very formally, and the application programmer can only build on them. The interface specification of a module with respect to the virtual processor is found in the module table and is expressed in terms of the virtual processor.

This abstract model and its formal presentation have proven to be extremely useful, both for separating tasks within the development process, and for automating the integration process.

Execution Model

Every module's program code can be considered a set of routines forming separate execution trees. The entry routines, that is, the roots of the trees, can be executed after the completion of a certain time period or upon reception of a message (Fig. 4). Since functional areas within a module may have different precedence requirements, execution trees can be assigned to one of several application priorities (see Fig. 5). For example, continuous waveform processing has the highest priority assignable, because it must process a batch of samples every 32 ms, and thus may delay the execution of all other functions if given a lower priority.

The total computing capacity of a certain CPU may be distributed among several execution trees within several modules with several priorities. The overhead generated for context switching is minimal.

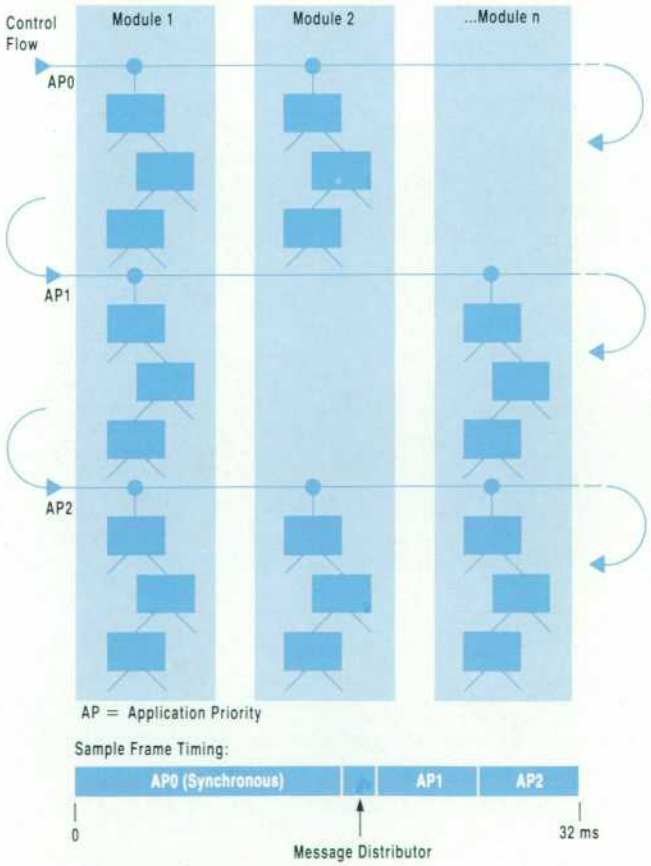


Fig. 5. Application module software can be thought of as a set of execution trees. These are assigned to application priorities, which are virtual processor resources.

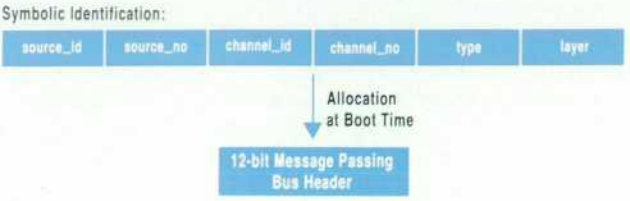


Fig. 6. Every message has a composite symbolic identification, which is evaluated at boot time, when message headers are allocated.

The application programmer specifies the priority of each execution tree. At run time, the execution time is partitioned externally to the module, without effecting its functionality. The programmer also specifies the amount of execution time required. The operating system guarantees every application the timely execution of each tree and checks this in a continuous fashion. This is essential to the safety of patient monitoring.

Communication Model

As already mentioned, communication among software modules and interface cards is implemented as an exchange of messages. Message routing functions reference a 12-bit header, allowing for a maximum of 4096 different data streams. It would have been possible to assign all headers project-wide in advance, but the Component Monitoring System employs a more elaborate process for establishing data paths. Every message has a composite (six-field) symbolic identification assigned. This is evaluated at boot time when headers are allocated (see Fig. 6). Modules transmitting or receiving messages specify this symbolic identification within their module tables. This method allows the implementation of some important concepts. If modules are installed more than once, say for multiple pressure lines, multiple similar data paths have to be established appropriately. This can be done externally to the modules at boot time simply by counting the source numbers shown in Fig. 6. Also, related messages can be collected into message classes by assigning identical keys, for example to the type field. Each message of a specific class then shares a predefined structure.

Programming with message classes represents a powerful method for dealing with configuration dependencies. The operating system supports broadcast messaging in a very convenient way. For example, all alarm messages generated by various sources are transmitted in messages of type ALARM. The central alarm management facility—an application software module—can specify that it wants to receive all alarm messages and then process them in the order that they appear. Using wild-card specifiers for the unknown fields in the symbolic identification keeps the module code receiving broadcast messages regardless of the current configuration. Blocks of memory for class members can be requested with a simple entry in the module table. The configuration dependencies are then resolved at boot time.

Besides broadcasting, the Component Monitoring System operating system uses two other important types of communication: static point-to-point and dynamic point-to-

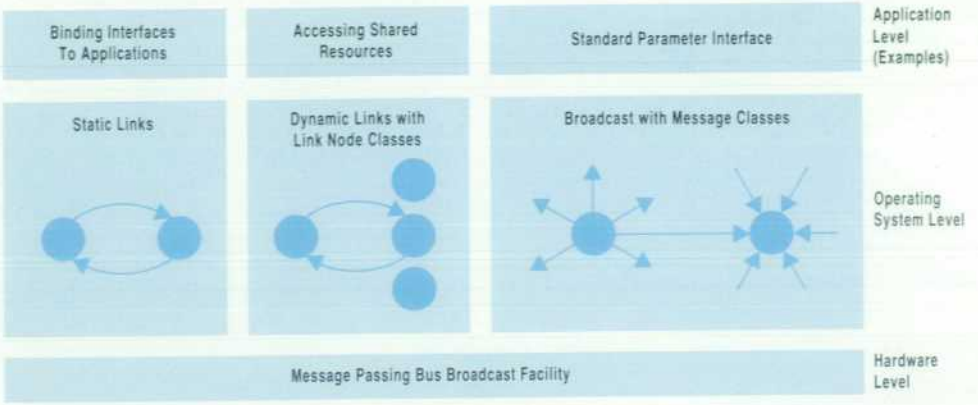


Fig. 7. Different communication models are built on top of the message passing bus broadcast function.

point. Static links are private communication links between two entities, most notably application software modules and interface cards. Dynamic links are valuable when one resource is used by different application software modules at different times. A temporary write to a screen area or the reading of softkeys are examples of this concept. Classes of dynamic link nodes can be defined using the mechanism described above (link nodes are entry points into an application software module whose affiliated message is of some type specified in the receiver's module table). Fig. 7 summarizes how these communication types are built on the message passing bus broadcast facility to serve the upper application level.

The standard parameter interface represents the backbone of patient data communication within the Component Monitoring System. It is a set of class definitions that forms a kind of logical data bus on which all patient data processing modules can broadcast their data. Any receiver can then operate on waveform, numeric, alarm, or other messages in a completely decoupled fashion.

Automated Configuration

Modules contribute to the system's flexibility only if their handling is simple, comparable to the ease of handling patient parameter modules. To achieve this, the Component Monitoring System development environment makes heavy use of automated processes, acting on formal interface descriptions. The module table, for example, is composed of the formal specifications relating to the specific module. When specifying communication behavior and execution requirements, the programmer can reference items of interest by means of symbolic names—the same names that can be found in the program source code.

As long as the system is not powered, all hardware and software components appear unrelated. The computer module cards are all connected electrically, but no logical data path is apparent. The CPUs are "empty" (Fig. 8a). At boot, a monitor configuration table located on the central EEPROM tells the boot process how to arrange software modules on the CPU cards (Fig. 8b). Using the module tables in the individual modules, the boot process binds the modules into the run-time system. After the program code has been transferred to the CPU cards and all configuration dependent data structures have been initialized, all modules will operate as expected. More than 40 application software module instances are installed in the Component Monitoring System.

In this process, the module tables supply all information necessary to install the data paths. In the first step, reservations for message passing bus headers are accepted. All references to communication concepts such as message classes can then be resolved. The process is analogous to the link process for computer object code. More than 800 message passing bus headers are allocated by the boot process; this gives an idea of the amount of communication that is required to operate the system.

Delaying the linking of software modules until the boot process has, among others, one important advantage: it

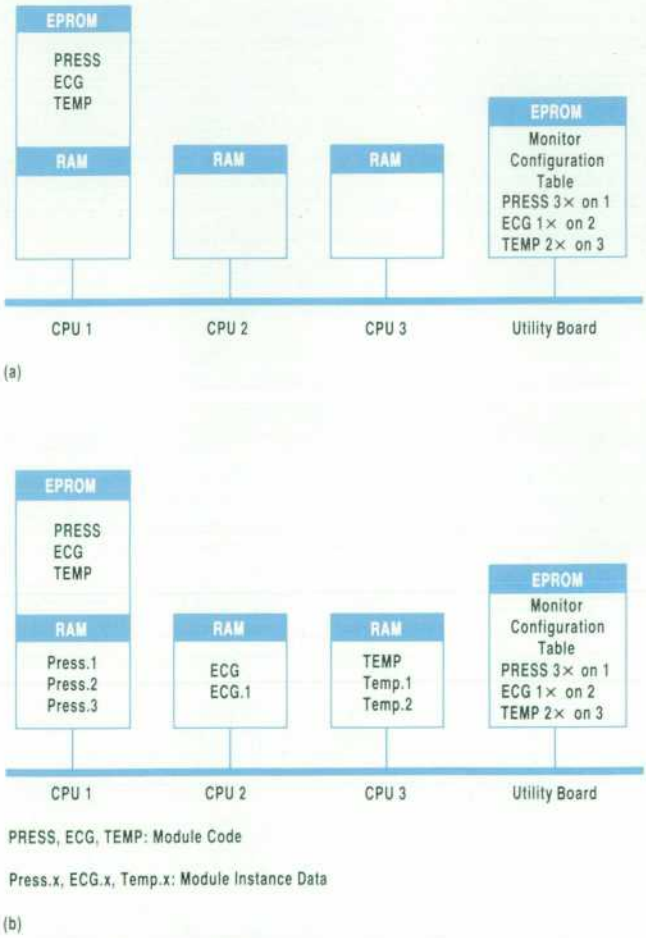


Fig. 8. A monitor configuration table tells the boot process how to arrange the application software modules on the CPUs.

allows configuration independent implementation and an ease of software maintenance that is normally true only of hardware elements.

Conclusion

Both the software design and the Component Monitoring System software development environment (see page 15) placed great emphasis on decentralization and decoupling and on the standardization and formalization of interfaces. The latter provides the opportunity for comprehensive automation, which in turn shows significant advantages:

- Automation of all external activities supports a smooth integration of each module into the total solution. It eliminates communication problems within the development team by separating responsibilities and establishing non-corruptible entities.
- Automated processes are reliable and efficient. They only have to be implemented once, and future users do not need an in-depth understanding to be able to use them.
- Automated processes enforce consistency. Deviations from a standard are prevented by the tools. Specification flaws quickly become apparent. The overall system remains consistent. This is an important contribution to software quality.
- Automated processes maintain flexibility. The evolution of processes can be coordinated centrally, with only a few engineers involved. Users are affected to a much lesser degree.

A project of the magnitude of the Component Monitoring System software development cannot be managed in a reasonable way without a very high degree of automation.

Acknowledgments

The author would like to thank the engineers from the operating system group, namely Martin Bufe, Kai Hassing, Holger Kaun, Paul Kussmaul, and Wolfgang Schneider, for their ingenious work on the operating system and the development environment, as well as their professional support for all application programmers.

Component Monitoring System Parameter Module Interface

This interface is the link between the component Monitoring System computer module and the patient parameter modules. It provides fast response, optimum use of the available bandwidth, configuration detection, and parameter module synchronization.

by Winfried Kaiser

The parameter module interface of the HP Component Monitoring System is the interconnection between the computer module and the module rack. The module rack can house a wide range and a varying number of parameter modules. By means of transducers attached to the patient, the parameter modules measure the patient's vital signs. These devices include the ECG, temperature, and recorder modules, and many others.

The major challenges associated with the design of the parameter module interface can be summarized as follows:

- The system must be able to support communication between the rack interface card in the computer module and a variety of parameter modules that may differ in such characteristics as the sampling rate of the analog signal, the number of signal input or output channels, the kind and amount of data that a parameter module receives from the computer module (control data) or sends to the computer module (status data), and mechanical size (1, 2, or 3 slots wide).
- It is a requirement of some clinical applications that certain waveform samples be measured and made available at an analog output with an absolute delay of less than 20 milliseconds.
- It must be possible to plug any parameter module into any slot in the module rack. The system must identify the parameter module and its position within the rack (configuration detection).
- The communication link of a system under power must not be influenced by plugging in or unplugging parameter modules or even entire racks.
- The parameter modules must be synchronized with each other.
- The link must support the detection of defective devices.

Link Design

The rack interface card in the computer module has one connector to interface to as many as four module racks. A module rack houses a maximum of eight single-width

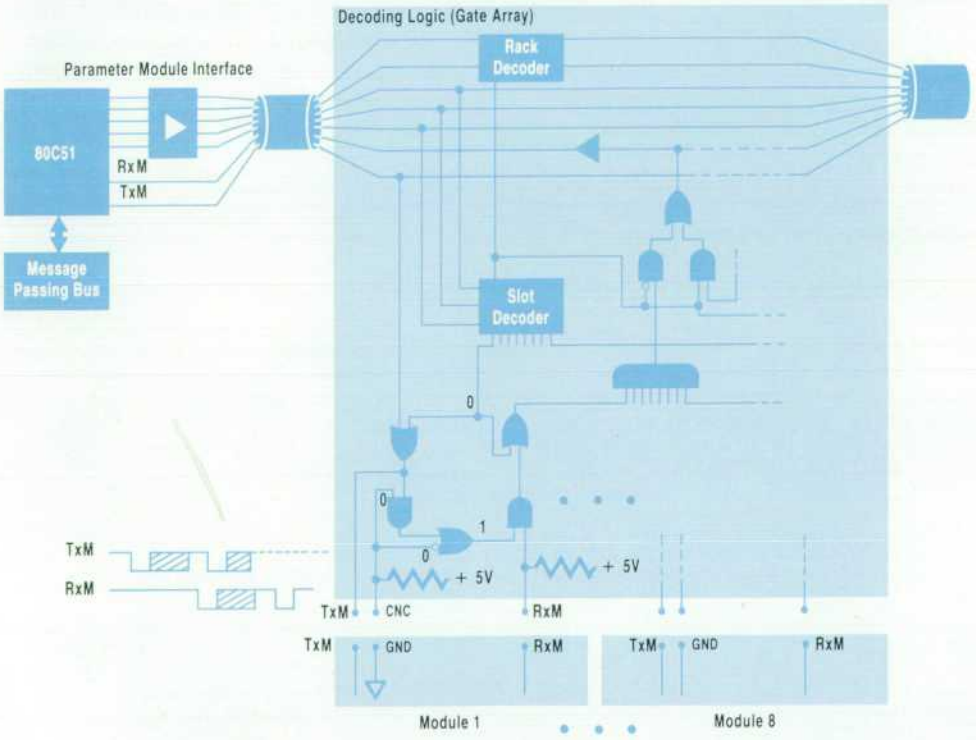


Fig. 1. Up to 32 parameter modules are addressed using five address lines. Decoding logic connects the addressed module to the receive and transmit lines of the rack interface in the computer module.

modules. This means that up to 32 parameter modules can be attached to one rack interface card.

Connections are established to each of the 32 slots by means of five address lines (see Fig. 1). Using these address lines, the decoding logic in the addressed module rack connects one of that rack's slots to the receive and transmit lines of the rack interface in the computer module.

The serial interface of the 80C51 microcomputer (internal UART, full duplex, 500-kbaud) is used for communication between the rack interface card and the addressed parameter module. Because of the fast response time requirement, it was decided that the parameter modules should transmit their information one sample at a time. The rack interface gathers all parameter samples over a period of 32 ms and forms them into the corresponding message passing bus data.

Communication Protocol

The rack interface controller starts the communication with the parameter modules with a special identification cycle. All possible rack slots are addressed, and a special control byte requests identification. A connected module responds by sending its device identification, hardware and firmware revision, and other parameter-specific data.

Using this identification and an internal reference table, the rack interface compiles all necessary data about the connected device types. This includes each device's sampling rate and its number and kind of transmit and receive bytes. After scanning all of the slots, the system knows which parameter modules are available.

Special digital logic together with a simple connected/not-connected connector pin inside each parameter module enables the rack interface to recognize whether or not a module is plugged into any slot, or whether a module is defective. If a parameter module is connected, it responds when it receives the control byte from the rack interface. If a module is not connected, the incoming byte is sent back to the rack interface card without any delay. If there is no response at all a defective module is recognized.

Scan Table

After the system determines which parameter modules are present, a scan table is generated in the rack interface to describe and control all subsequent communication. The scan table consists of 16 2-ms time slices. The table entry for each time slice specifies which parameters are polled during that time slice (see Fig. 2).

The arrangement of the scan table depends on the speed classes of the parameter modules connected. There are five speed classes based on the sampling rate of the parameter modules: 2-ms, 4-ms, 8-ms, 16-ms, and 32-ms. The 2-ms parameters are entered in each column of the scan table, the 4-ms parameters in every other column, and so on. A special algorithm guarantees that the entries are made so that each device is addressed at fixed intervals.

The free part of the scan table is used to address slots that have no modules inserted. When a parameter module is plugged into the rack it is immediately recognized and activated in the scan table, which contains the superset of all parameter modules that are allowed. A module that is removed from the rack is deactivated. Thus it is possible to connect and disconnect any parameter module during normal monitoring.

Analog output devices that need a fast response time are entered at the end of each time slice and receive the data from the selected parameter module within the same time slice. The total delay from input to output is less than 2 ms (see Fig. 2).

Parameter Module Interaction

When a parameter module is addressed by sending it a message (receive interrupt), it responds immediately by transmitting a predefined internal data block, typically consisting of waveform and status data. After the transmission, the parameter module starts an analog-to-digital conversion cycle of the patient signal or performs other tasks depending on the control information in the received message. The result of the analog-to-digital conversion and status information are stored into a module's internal data block. This data is transmitted the next time the parameter module is addressed. Since communication with a particular device always takes place after a fixed interval, the module can synchronize itself with this polling sequence.

At the 500-kbaud data rate of the parameter module interface, a typical communication cycle with a parameter module with one waveform takes about 120 microsec-

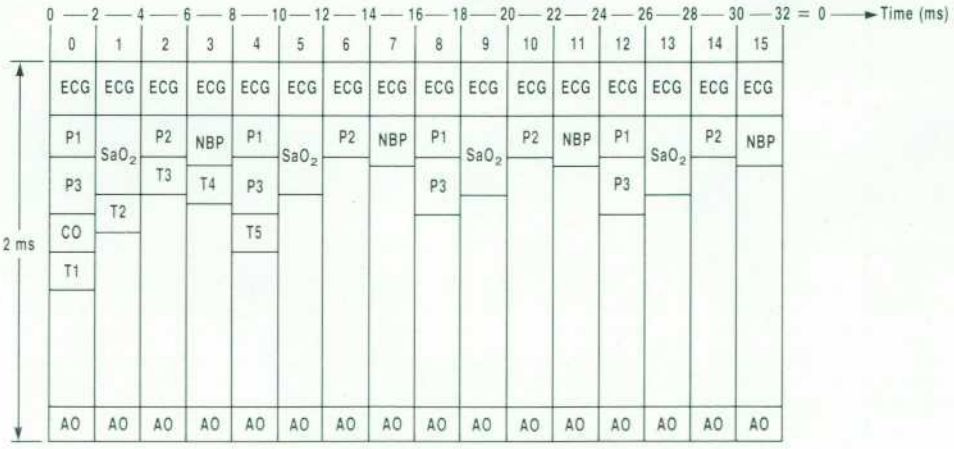


Fig. 2. The scan table entries specify which parameter modules are addressed in each of sixteen 2-ms time slices. Analog output devices that need data immediately receive it within the same time slice.

onds. When communication with one module has been completed, the next device in the scan table is addressed.

This procedure ensures both optimum use of the available bandwidth and synchronization of the parameter module and the rack interface card.

Summary

The parameter module interface represents a very flexible solution for connecting a wide variety of modules to the Component Monitoring System computer module. Although the requirements addressed by the interface are complex, the final implementation is both versatile and rugged, and was kept relatively simple by integrating some of the decoding logic.

Acknowledgments

I would like to thank Dieter Woehrle and Norbert Wunderle for their excellent contributions in developing and implementing the parameter module interface and the firmware for the parameter modules.

Measuring the ECG Signal with a Mixed Analog-Digital Application-Specific IC

Putting the ECG data acquisition subsystem into a Component Monitoring System parameter module mandates high-density packaging and low power consumption, and was only possible by implementing major elements of the circuit in a large mixed analog-digital ASIC.

by Wolfgang Grossbach

Nearly everyone is familiar with one of the most important medical parameters—the electrocardiogram, or ECG. The electrical voltages created by the heart have been well-known to the medical community for nearly a century. In the beginning the ECG was measured by sensitive galvanometers with the patient's hands and feet placed in vessels filled with saline solution. Improvements in the assessment of diagnostic ECG signals have been closely related to the evolution of electronics, great strides being made when amplifying devices such as vacuum tubes and later transistors became available. Today, low-noise operational amplifier circuits are state-of-the-art for ECG signal processing.

Physiologically, the polarization and depolarization of the heart's muscle mass creates a three-dimensional electrical field that changes with time. As a result, voltages can be measured on the surface of the body that represent the pumping cycle of the myocardium. A strong effort has been made to standardize the points at which the voltages should be measured. The most widely used are three differential voltages: From right arm (RA) to left arm (LA), from LA to left leg (LL), and from LL to RA. These voltages are known as ECG leads I, II, and III. The right leg electrode (RL) acts as the neutral pole in this system. This configuration is known as the Eindhoven triangle (see Fig. 1).

ECG Signal Characteristics

The amplitude of the ECG signal as measured on the skin ranges from 0.1 mV to 5 mV. The frequency extends from 0.05 Hz to 130 Hz. Physiological signals like the ECG differ from artificial signals in that they are not reproducible from one time segment to another. They are more statistical in nature and have larger variations in signal characteristics than, say, a signal generator output. This places additional requirements on the measurement system, especially the analog input stages. Although the average amplitude is only around 1 mV, there are large dc offset voltages because of electrochemical processes between the electrode attached to the patient and the patient's skin. These can be as high as ±500 mV. Also, the contact resistance between an electrode and the body surface can vary widely and reach values around 1 MΩ. In addition, the system must be capable of detecting that an electrode has fallen off the patient. Perhaps the largest constraint is the presence of 60-Hz or 50-Hz power line noise. The human body acts like the midpoint of a capacitive divider between one or more power lines and ground. Thus, common-mode voltages as high as 20V p-p can be superimposed on the body. Eliminating this source of noise is one of the major tasks of an ECG amplifier. Fortunately, the ECG signals are differential signals while the power line

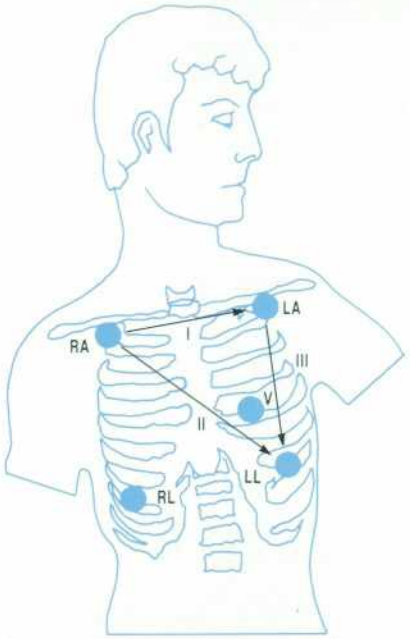


Fig. 1. Placement of ECG electrodes. The colors of the cabling system are standardized. The right leg (RL) connection acts as the neutral pole. (In monitoring applications the RL and LL connections are often as shown here and not on the legs.)

voltages are common-mode, so the noise can be reduced with differential amplifiers.

Another requirement results from artificial pace pulses used to stimulate the heartbeat of some patients. Pacemaker devices are implanted into the chest, generating small pulses up to 1V p-p at frequencies in the kilohertz range. Pace pulses are superimposed on the ECG signal and have to be detected to differentiate them from the peak value of the ECG signal, called the QRS complex. This is important because the heartrate measurement is based on this QRS signal, and an incorrect interpretation would result in an incorrect value.

In emergencies when the heart stops beating (ventricular fibrillation), a commonly used procedure is to apply a voltage pulse of about 5 kV p-p with a 5-ms duration to synchronize the neural stimulus of the heart's muscle mass and bring it back to normal operating conditions. Because of the high voltages needed to defibrillate a patient, the inputs of the ECG circuit must be protected. Other sources of noise are electrosurgery devices, which are used in operating rooms as electronic scalpels. These devices contain high-frequency currents in the megahertz range. The high current density at the tip of the electrode coagulates the protein, thereby stopping bleeding. The ECG module must provide additional filtering against this high-frequency noise.

Integrated Solution

The design goals for the Component Monitoring System ECG module included reduced cost, reduced size, minimum power consumption, and increased reliability and functionality compared to the current patient monitor generation.

The target size was a single-width parameter module. This module measures only 99.6 mm by 36 mm by 97.5 mm (3.9 in by 1.4 in by 3.8 in). It was therefore obvious that we would have to use surface mount technology to meet the size objective. In addition, it soon became apparent that a large percentage of the electronic circuit would have to be integrated in silicon if the entire device was to fit into a single-width module. This and the need for cost reduction on such a high-volume parameter module as the ECG module clearly indicated that an application-specific integrated circuit (ASIC) would be the appropriate solution.

The investigation revealed that the chip size we had in mind and the mixed analog-digital design were real challenges for a fully custom ASIC. Our plan was to integrate the following function blocks into a single chip:

- Full three-channel ECG amplifier with various filter stages of both analog and switched capacitor type
- Precision resistor network for the weighting function
- Three-channel lead select multiplexer
- Precision differential amplifiers with high common-mode rejection ratio
- Eight-channel multiplexer for sequential scanning of all analog signals
- Bandgap voltage reference
- 10-bit analog-to-digital converter (ADC)
- Digital control logic for switching filter corner frequencies, multiplexers, and other circuits
- Serial interface to connect the chip with the surrounding circuits.

To be able to integrate all this, a 3-μm CMOS process was chosen. It is a p-well LOCOS process with polysilicon gates and ion implantation. NMOS and PMOS field-effect transistors are combined. Also available are n-channel JFETs and pnp bipolar transistors. The thin-film resistors are laser trimmable to within 0.1% matching. Available cells include JFET operational amplifiers, bipolar opamps, switched capacitor filters, 8-bit to 14-bit analog-to-digital and digital-to-analog converters, and sample-and-hold amplifiers.

The Electrocardiograph ASIC

The basic functions of the ECG circuit can be seen in Fig. 2, which shows one of the three independent channels. The inputs are switched to the RA and LA electrodes as active inputs. The RA and LA inputs of the chip are connected to the patient. Protection circuits against ESD and defibrillator pulses and current-limiting resistors are provided outside the chip on the printed circuit board. JFET input opamps amplify the signal five times and act as high-impedance input buffers. A precision resistor network (Wilson network) sums the various electrode voltages to achieve the standard voltages for the different ECG selections. The multiplexer selects the appropriate lead voltages from the resistor network. The 10-kHz low-pass filters act as prefilters for anti-aliasing purposes to reduce the high-frequency components in case an electrosurgery unit or other high-frequency noise source is coupling into the module. These are analog filters. They protect the switched capacitor filters with their time-discrete sampling system against unwanted aliasing disturbances resulting from high-frequency noise.

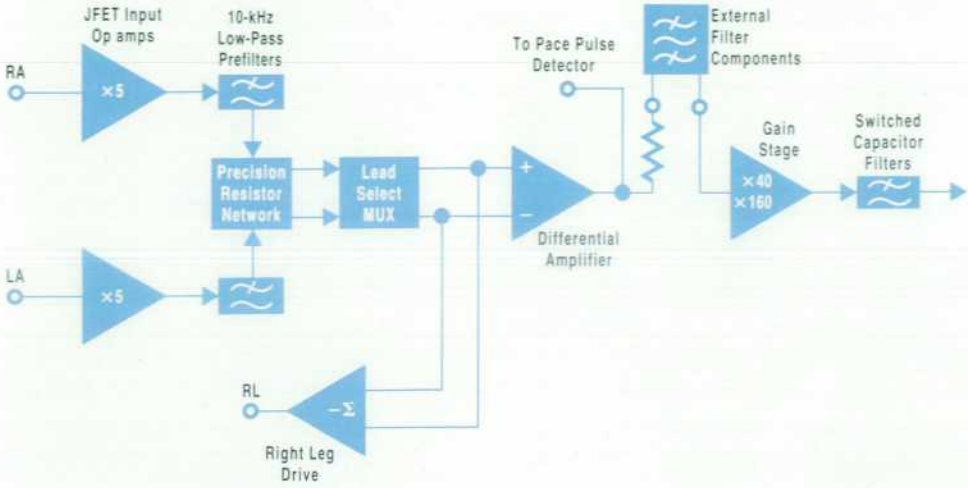


Fig. 2. Basic structure of the ECG ASIC (one of three channels).

The RL input acts as the neutral pole, but is not directly connected to analog ground. It is the low-impedance output stage of an inverting summing amplifier (called the right leg drive) which serves as a feedback circuit, further reducing common-mode power line voltages. The common-mode signal present at the output of the lead select multiplexer is phase inverted and fed back to the patient, thus being subtracted from the common-mode voltage present at the inputs. This helps eliminate unwanted power line noise.

The difference between the two selected electrode signals is derived in the differential amplifier section, which has a gain of one. Up to that point, all gain variations and tolerances affect the common-mode rejection. Therefore, these stages have laser-trimmed resistors where appropriate.

At the outputs of the differential amplifier in each of the three channels, the signal path is split into two parts. For the two main channels, the auxiliary path goes out of the integrated circuit to the pace pulse detector. The pace pulses are identified by their higher-frequency content in the range of 1 to 4 kHz, but only the presence of a pace pulse has to be detected, not the time dependent signal itself. Therefore, it is unnecessary to construct the whole signal path with this large bandwidth. After the pace pulse detector output, low-pass filtering of the ECG signal begins.

For ECG filtering, a minimum lower corner frequency of 0.05 Hz is required. The large capacitor and resistor values needed could not be integrated and therefore the signal is routed from the chip into external filter sections, one for each channel. By means of internal switches, three low-end corner frequencies (0.05 Hz, 0.5 Hz, 3.5 Hz) and two high-end corner frequencies (40 Hz and 130 Hz) can be selected.

The signal flows out of the chip, through the external filters, and back into the chip. It then goes through the main gain stage, which has switchable gain of 40 or 160 depending on the signal amplitude and the resolution needed on the screen. After passing the gain stage, the signal is filtered with a second-order switched capacitor stage to achieve the corner frequency of 130 Hz with as small a tolerance as possible.

The three analog channels described so far are connected to the inputs of a one-of-eight multiplexer, which sequentially scans these three channels and five auxiliary channels every 2 milliseconds. The output feeds into the ADC, a 10-bit converter that has less than ±1 LSB differential nonlinearity and a conversion time of 20 μs. An 8-by-10-bit dynamic random access memory holds all the conversion results temporarily until they are transmitted via a parallel-to-serial converter out of the chip to the module microprocessor. In the opposite direction, all control information is transferred into the chip over this serial interface and latched. The use of a serial data conversion scheme made it possible to use only three output lines and a 28-pin package.

Fig. 3 is a photograph of the ECG ASIC chip.

Pace Pulse Detection Circuit

The dual pace pulse detector is also an ASIC. Its analog parts are built entirely in switched capacitor technology.

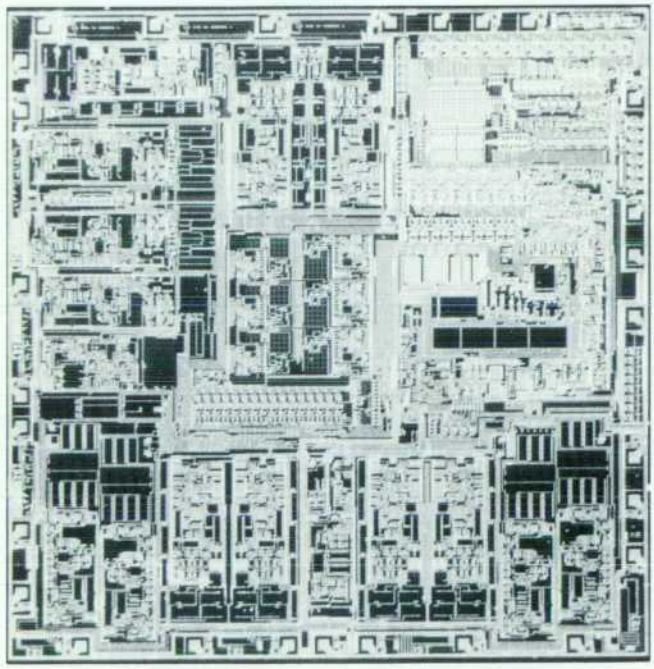


Fig. 3. Photograph of the ECG ASIC wafer. The analog functions cover much larger areas than the digital parts.

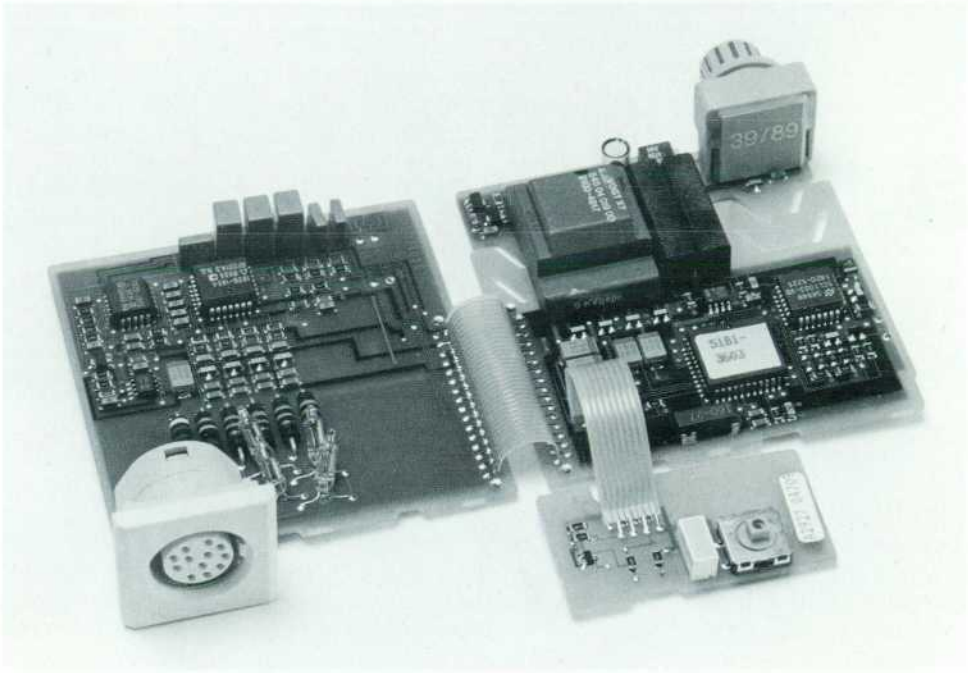


Fig. 4. M1001 ECG module printed circuit boards. The large capacitors on the left board are part of the external filter stages. The ECG ASIC is just in front of these capacitors. The PPD is to the left of the ASIC. The right board contains the digital parts of the module.

This had the advantage of avoiding laser trimming, minimizing wafer area, and thus reducing cost. This chip generates two logic output signals for each channel, indicating whether a pace pulse with either positive or negative polarity is present in the input signal. The information is polled by the microprocessor and sent to the algorithmic software.

Test Considerations

It was clear from the beginning that testing the ECG chip would be a challenge because of the large number of parameters to be measured. The specifications describing the functionality are split into two parts: internal and external specifications. The internal specifications can be tested with wafer probes and help the vendor optimize the production process. They are consistent with the external specifications, which are measurable from outside the chip and are accessible to the customer. The external specifications are the link between the ASIC design and the printed circuit board design and were used as guidelines throughout the design and verification process. Automated test equipment has been set up at HP to test the ECG chip via its serial interface. The same test equipment is used by both the vendor and HP to reduce the number of false measurements resulting from different measurement setups.

Results

Fig. 4 shows the printed circuit boards of the M1001 ECG module. All components between the ECG ASIC (the larger one) and the input patient connector are for protection and filtering against defibrillator pulses, electrostatic discharge, and electrosurgery units. The following table gives an overview of the two ASIC chips (PPD is the pace pulse detector):

Item	ECG ASIG	PPD ASIC
Die Size	6.22 by 6.27 mm	3.43 by 3.99 mm
Analog Functions	24	12
Digital Functions	4	2
Number of Transistors	≈6000	≈2000
Number of Digital Gates	≈550	≈200
Number of Pins	28 PLCC	20 PLCC
Power Consumption	275 mW max.	45 mW max.
Dynamic Range	± 3 V	±3 V
Overall Analog Gain	800	
Noise (referred to input)	2 LSB	

The main problem that was faced in this design project was the complexity of the system, which caused side effects that were not visible in the beginning. The die size was too big for normal packages, so packages with larger cavities had to be found. The simulation time was longer than expected because of the large number of components inside.

In summary, the design objectives were met. The ECG performance is state-of-the-art, and significant reductions in cost, power consumption, size, and component count were achieved in comparison to a discrete solution, which probably would have required a double-width module.

A Very Small Noninvasive Blood Pressure Measurement Device

This small assembly covers the entire blood pressure measurement spectrum from neonates to adults. The packaging of the air pump assembly makes several contributions to the objectives.

by Rainer Rometsch

The noninvasive blood pressure module of the HP Component Monitoring System is a double-width parameter module used to measure and calculate a patient's systolic, diastolic, and mean blood pressure. The method is based on inflating a cuff on the patient's arm until all blood flow is suppressed in this extremity. The pressure in the cuff is then slowly deflated, and by using the oscillometric measurement technique, both the high and low blood pressures and the mean value can be determined.

Physically the noninvasive blood pressure module consists of two parts. One is the electronic board, which contains the power supply, the signal acquisition circuitry, and the interface to the computer module. The other is the pump assembly, which is responsible for the controlled inflation and deflation of the cuff.

Requirements imposed on the pump assembly were:

- Low parts price
- Minimum number of parts
- Robust construction
- Easy to assemble in the parameter module
- Low power consumption at the highest possible pump capacity.

Because of the required size of the pump assembly (it had to fit in a single-width parameter module), and the need to reduce the number of individual parts, a totally new approach was taken in the design of this mechanical part. The solution implemented is a self-contained function block allowing a stringent separation between the electronic printed circuit board and the pneumatic system (see Fig. 1).

The Pump Assembly

The pump assembly consists of the pump and two valves. The pump is a membrane device driven by a dc motor. To meet the requirements of the application, the pump is optimized for high pumping capacity and low air leakage. Air leakage is a big concern because the volume in the cuff has to be deflated in a highly controlled fashion. This is especially difficult for neonatal applications, because neonatal cuffs have very small volumes. We solved the problem by incorporating a reflow valve within the pump module. This pressure valve opens at low flow rates, and therefore does not increase power consumption. The important thing is that it seals tightly at a very low reflow rate.

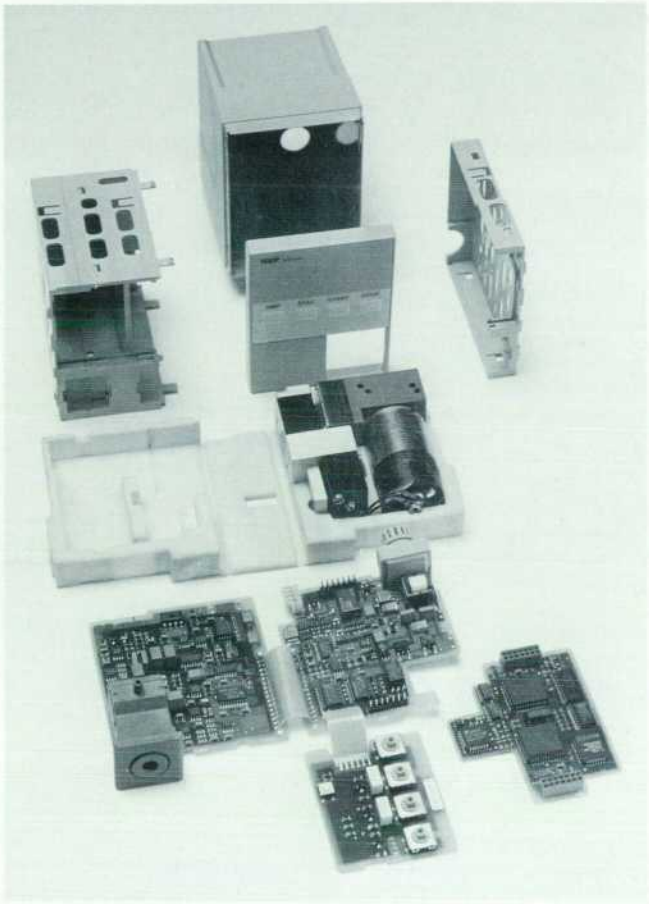


Fig. 1. Pump assembly of the Component Monitoring System noninvasive blood pressure measurement module.

The second element in the pump assembly is a pair of valves, flanged to a machined aluminium extrusion. Two valves are needed to provide a fail-safe circuit that will comply with the safety requirements imposed on noninvasive blood pressure measurement devices. These two valves have considerably different flow characteristics. By automatically switching between the two valves, one noninvasive blood pressure module covers the entire application spectrum from neonates' cuffs all the way to adult thigh cuffs.

The aluminium extrusion is designed to replace all the necessary tubing between the pump and the valves. It is

therefore possible to flange the mounted valve onto the pump without any additional rubber tubing. This contributes to a part count reduction and simplifies production dramatically.

The only connections that have to be made with the pump assembly are a 15-mm-long rubber tube to the noninvasive blood pressure connector in the parameter module, and a power connection to the electronic board for the dc pump. The rubber tube to the noninvasive blood pressure connector is essential because this flexible tubing detaches the pump from the the parameter module housing and thus helps damp acoustic noise caused by mechanical vibration.

Packaging

The entire pump assembly is encapsulated in a polyurethane package. This relatively simple part contributes in more than one way to the goals of the overall solution. All noise generated by the dc pump is muffled by the package to a degree that is acceptable in the hospital environment. The pump assembly survives the 1-m drop test because sufficient kinetic energy is absorbed by the package to avoid damage to the mechanics. Packaging of this part for shipment from the vendor to HP has been minimized to a simple protective cover.

The outline of the foam package is identical to the inner contour of the parameter module. Therefore, no additional parts are needed to embed the pump assembly in the inner enclosure of the parameter module. The elimination of additional screws or clamps has helped reduce production time and part count.

Conclusion

The Component Monitoring System noninvasive blood pressure module meets all of the above described objectives. Because the pump assembly and the electronic board are delivered as prefabricated parts, the total time to build the module has been reduced to about two minutes. The result is a small, robust, self-contained noninvasive blood pressure module, to our knowledge one of the smallest noninvasive blood pressure devices in the world.

Acknowledgments

The author would like to thank the materials engineers, Eberhard Mayer and Willi Keim, for their continuous support and professional advice.

A Patient Monitor Two-Channel Stripchart Recorder

Small enough to fit in a double-width HP Component Monitoring System parameter module, this recorder embodies simplicity of design, a highly tooled mechanism, and sophisticated printhead power management.

by Leslie Bank

The medical environment requires a record of the care that has been given to a patient, both for the patient's file and as a legal document. For patient monitoring equipment like the HP Component Monitoring System, the record has traditionally been a continuous strip of paper of various widths. An example of a recording from the Component Monitoring System's two-channel recorder is shown in Fig. 1. Fig. 2 is a photograph of the recorder.

In the past, the hospital had three options to provide recording capability for a patient, each of which was less than ideal:

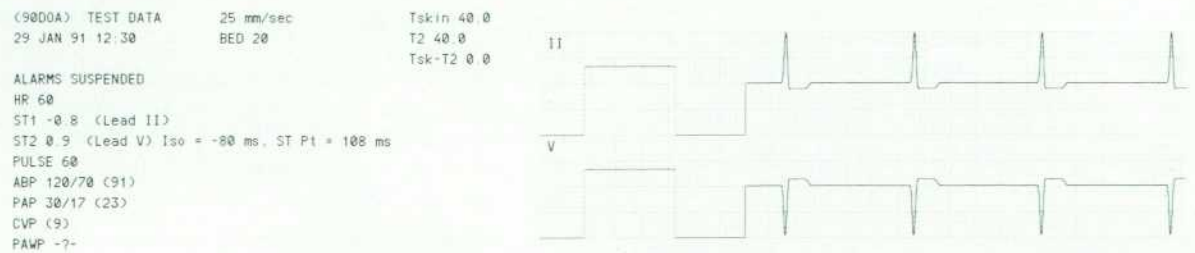


Fig. 1. Typical two-channel stripchart recording of patient data.

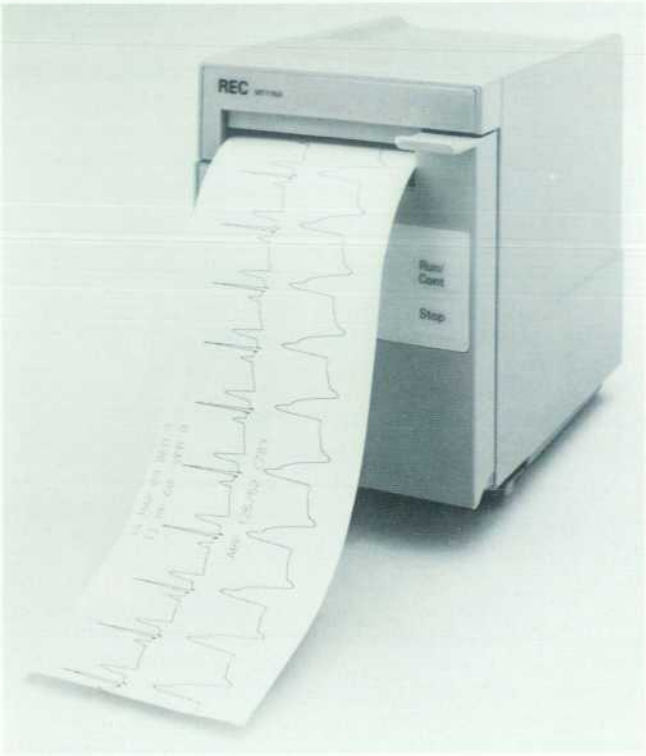


Fig. 2. The HP Component Monitoring System two-channel recorder module.

- Purchase a recorder for every bedside. This is very expensive in these days of cost containment.
- Use a central, shared recorder. With much of the patient care given at the bedside, not having a recorder nearby is a distinct disadvantage.
- Mount the recorder on a cart and wheel it to the bedside when needed. This takes up too much of the available room at the bedside and is also inconvenient.

The Component Monitoring System philosophy of allowing the monitor configuration to change with the patient's needs extends to the recording function. The two-channel recorder can be moved around like any other parameter module. This approach, along with the requirements for ease of use, high reliability, high performance for many types of applications, low manufacturing cost, and low power led to the following set of major specifications:

- Size: Double-width parameter module
- Power consumption: Approximately 6 watts maximum
- Number of waveforms: 3
- Lines of character printing: 3
- Paper: 50-mm-by-30-m rolls (fan-fold paper would not fit in the desired package size).

These specifications resulted in a number of major technical challenges.

Size. Fitting the paper, motor and drive mechanism, electronics, and supporting structure into a package of this size was a major accomplishment.

Power Consumption. Chemical thermal paper is used in this recorder. A printhead consisting of a linear array of resistors is in constant contact with the paper. When power is applied to one of these resistors, the resistor gets hot and a mark is made on the paper. This, combined with the power requirements of the motor and electronics, normally would require much more power than the 6 watts that are available. In addition, there can be no ventilation in the housing. Meeting the high-temperature specifications was difficult because of the internal heat generated by the power-consuming components.

Ease of Use. Recorder operation should be flexible to meet the various medical applications. It should be intuitive for the occasional user. Most of the Component Monitoring System recorder operation is part of the normal control structure. The difficulty for the recorder design team was to make the paper loading easy while not using any power to aid paper feeding.

Reliability. Recorders, which have moving parts that wear, tend to be less reliable than equipment that does not have moving mechanical parts. A simple mechanical design along with high-quality components and a severe testing program resulted in a highly reliable product.

Manufacturing Ease. This recorder was designed for high-volume assembly. Much effort was spent in minimizing part count, in using the molded parts to perform multiple functions, in designing adjustments out, and in making the instrument easy to test.

Mechanism

Paper is loaded by opening a door and inserting the roll of paper into the paper compartment. The paper is then threaded around a drive roller and pulled taut, and the door is closed. As the door is closed, a cam is engaged which lowers the printhead. The roller is driven by a stepper motor which is connected to the drive roller by a drive belt. The roller is driven when the motor turns. The paper has enough wrap around the drive roller to ensure that it can be driven under the printhead. Enough back tension must be provided to make the paper track properly, yet too much tension increases the motor torque requirements, which in turn increases the power required. This turned into an interesting design trade-off. Sealed ball bearings are used on the drive roller to minimize power requirements while keeping paper dust out of the bearings.

Two injection-molded frames form the chassis. The printhead and drive roller are captured between the chassis halves, while the motor, paper door, power supply board, and digital board are all mounted to the outside of the chassis. The entire assembly is enclosed in a double-width module case.

Electronic Hardware

The digital board contains two Intel 80C196 16-bit microcontrollers which communicate with each other via a shared RAM. Each microcontroller contains a serial port. The I/O processor uses its serial port to communicate with the monitor's computer module via the parameter module interface (see article, page 19). It receives digital commands, waveforms, and text data from the monitor. It interprets the commands and transforms the data into a format compatible with the printhead. It also monitors the front-panel and door-open switches and the paper-out sen-

sor. The I/O processor communicates the recorder status to the monitor.

The other processor takes the information from the I/O processor and ships it to the printhead via its serial port. The energy applied to each resistive dot in the printhead is tightly controlled by varying the printhead strobe times. This ensures high-quality printing and long printhead life with minimal energy use. This processor varies the printhead strobe based upon printhead temperature, resistance, and voltage, which are measured by the onboard analog-to-digital converter. The motor speed data is also sent to the motor drive chips, which are located on the power supply board.

The power supply board transforms the 60 volts received from the monitor into 15 volts required by the printhead and motor, and into the 5 volts required by the logic. This power conversion is performed by a switching power supply with a typical efficiency of 83%. The motor is driven by two stepper motor chips which, under control from the digital board, microstep the motor to provide accurate chart speed with minimal power. The peak energy supplied to the printhead is provided by a large capacitor. In case of extremely heavy printing, the power to the printhead may sag. To prevent the 15-volt supply from sagging too much, a current limiter is placed between the printhead energy storage device and the 15-volt supply. Finally, the optical isolators for the serial data lines to the monitor are on this board.

Printhead Control

Character and grid generation are provided by the recorder. The selected characters and grid are combined with the waveform data, rasterized, and sent to the printhead to energize a number of resistors (dots) in the printhead. The printhead is loaded three times for each dot printed. If the dot has not been fired and is "cold", it is fired for all three loads. If the dot has been fired in the last cycle, it is "hot" and is fired for only one load. If the dot has been fired two cycles ago, it is "warm" and is fired twice. This results in a historical firing of each individual dot and precise temperature control of each dot. In addition, each of the three loads is accompanied by a strobe of the printhead. Each strobe time is varied based upon printhead voltage, temperature, resistance, and chart speed. For example, as the temperature of the printhead increases, the amount of energy provided to all dots decreases. This results in a lower printhead temperature rise and less thermal shock to the dot resistors, while providing consistent printing quality. In addition to all this, the entire printed area is dithered up and down over time. This equalizes the use of all the printhead resistors and improves the life of the printhead.

Acknowledgments

The techniques presented here have resulted in an efficient product design that builds upon the strengths of the Component Monitoring System. The simplicity of design, highly tooled mechanism, sophisticated printhead power management techniques, and much testing have resulted in a reliable, high-performance yet cost-effective solution for our customers. Much effort and hard work went into making this product happen. I cannot hope to thank everyone involved on the recorder project, but I would like to give special thanks to Gerry Patrick, Renee Olson, Walter McGrath, Chris Rothwell, Jeff Berry, Dominic Lucci, Harry Schofield, and Jim Larsen.

Patient Monitor Human Interface Design

A design based on human factors leads to an intuitive and easy-to-use human interface for the HP Component Monitoring System.

by Gerhard Tivig and Wilhelm Meier

The design of the human interface for the HP Component Monitoring System involved a coordinated effort of R&D, marketing, and industrial design, working with valuable inputs and feedback from the principal users—the intensive care unit (ICU) nurse and the anesthesiologist. Fig. 1 illustrates the basic elements of the design process for the human interface.

The functionality of the Component Monitoring System goes beyond the classical real-time patient monitoring functions. The monitor offers extensive support for medical procedures, such as cardiac output and S-T depression and elevation measurements, a powerful data management capability with various calculation and report facilities, and a review facility for alarms and patient information from "another bed" using the proprietary HP serial distribution network (SDN). This functional complexity had to be handled with a single consistent and simple operating structure so that it did not lead to a complex user interface. Because it is a key element in the user's ultimate buying decision, usability was a critical issue in the design.

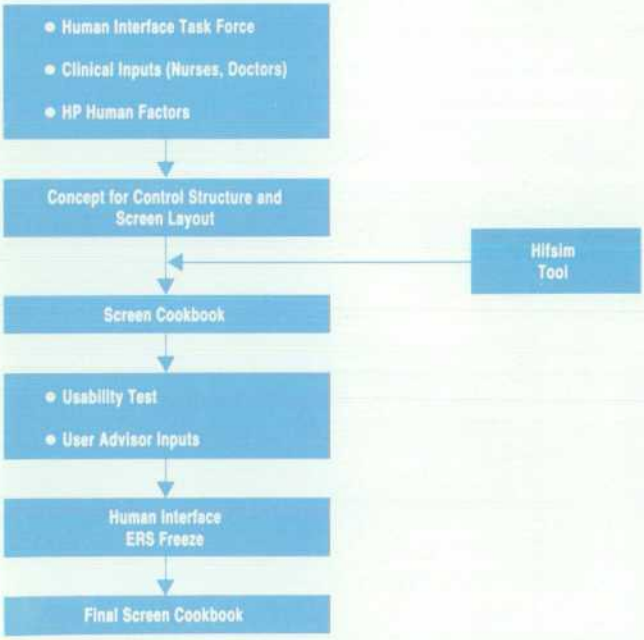


Fig. 1. Design process for the human interface of the HP Component Monitoring System.

Environments and Users

The Component Monitoring System is used in a variety of environments, including the surgical ICU, the neonatal and cardiology ICUs, and the operating room. There is a wide spectrum of users, including the nurse in the ICUs and the nurse anesthetist, the anesthesiologist, and the perfusionist in the operating room.

The primary user in the operating room is the anesthesiologist. Some of the tasks performed are of a clerical nature, such as logging patient and life support device data, observing the monitor, and scanning the area. There are also physical tasks, not directly related to the monitor, such as patient preparation, administration of drugs and fluids, and patient observation.

In the surgical and neonatatal ICUs, 90% of users are nurses. The tasks performed by the nurse include 30% clerical activities, such as recording medical data, writing down and checking doctors' orders, writing down the medication plan, and filling out the patient's flowsheet. The other 70% of the tasks performed are of a physical nature, such as administering fluids and drugs, taking measurements, making physical examinations, ensuring patient hygiene, and performing medical procedures.

In most cases, nurses and physicians have no computer experience. It can be expected that many of them will have doubts about the introduction and use of computer-based monitoring equipment. Therefore, it was considered advisable not to make the Component Monitoring System look like a computer.

The main focus is on the patient. The nurses and the physicians do not have time to interact extensively with the monitor. They are in a crowded and stressful environment, where it is not unusual to encounter critical situations requiring immediate action to prevent degradation of the patient's situation. Clinical personnel also face a wide variety of equipment from different manufacturers, all with different user interface standards.

Equipment training often includes no more than one or two hours of instruction at monitor installation time. The turnover of the nursing staff may be very high. Because the workload is heavy, there is no time to read extensive operating manuals, instruction cards, or help texts. Because of economic pressures on the health care system

and clinical personnel shortages, especially in nursing, less time is available for in-service training.

All of this suggests that intuitiveness and ease of use are fundamental requirements for the Component Monitoring System human interface.

Design Objectives

As the performance and computational power of a patient monitor increase, the challenge is how to present and use the medical information provided by the monitor in an easy-to-interpret, simple, interactive way that will lead to more efficient patient care delivery. It is possible to control a monitor with two buttons and a lot of key pushing and watching. It is also possible to have 100 buttons or more for the same job and assign a distinct function to each button. An optimum is somewhere in between.

The main goal was to design a consistent control structure for all applications in the monitor and across all present and future members of the Component Monitoring System family. Working towards a simple model in the user's mind was considered more important than reducing the number of keystrokes required to access a given function to an absolute minimum. Having formed a model of how the system operates, the user can extrapolate how a particular function might work. If the system is consistent, the user's prediction will work, the system will be perceived as easy to use, and user acceptance and satisfaction will increase. The control structure needs to be self-explanatory to the novice user and allow fast access to the experienced user. Access to critical functions requiring immediate action (like silencing an alarm or freezing the screen) should be simple and fast and should not interrupt the user's activity in a given operating window.

Minimizing operating complexity by reducing the number of nested operating levels and thus eliminating the need for "navigation aids" has been a major quantitative goal. Each Component Monitoring System function is accessible within three operating levels, reinforcing the same access to all functions. There is a home key, labeled **Standard Display**, which always brings the user back to the standard resting display. Because monitoring functions vary in their complexity, the human interface design implements simple things in an easy way while making complex tasks possible.

Elements of the Human Interface

The main elements of the Component Monitoring System human interface can be seen in Fig. 6 on page 12. All user interaction and data visualization take place through a human interface unit consisting of a 14-inch high-resolution display (monochrome or color) and a keypad integrated in the screen bezel. Optionally, a remote keypad can be attached to the Component Monitoring System through the standard HP-HIL interface. The remote keypad duplicates all of the keys on the screen bezel and has an additional alphanumeric entry capability. The screen bezel also contains the sound generator for the alarm interface and the visual alarm indicators, which are color-coded alarm lamps (red, yellow, green). The controls and lights on each patient parameter module are integrated into the overall operating concept. Each single-width parameter module has one or two keys. One key is always a setup button, which allows direct access to the setup menu for that parameter module. The other key is optional and allows quick operation of functions, such as zeroing a transducer, starting a cardiac output measurement, or calibrating the CO_2 analyzer.

Most operations are controlled by a mix of twelve hardkeys and seven softkeys. A group of arrow keys on the right side of the keypad (up, down, left, right, confirm) support the pointing and select functions of the user interface.

Hifsim and Its Benefits

The control structure and the screen layout were exposed to nurses, physicians, and anesthesiologists in the early stage of the design process. This was possible through the use of a simulation tool.

At the time the human interface design started, very few simulation tools were available, and in most cases they didn't match the designers' requirements. We chose to develop our own simulation tool, called Hifsim. This took four engineer-months. Hifsim runs on an HP 9000 Series 300 workstation under the HP-UX operating system.

The intended use of the Hifsim tool for usability tests made it mandatory to come up with a keypad integrated into the screen bezel to resemble as much as possible the way a nurse would interact with the monitor. Similar pixel resolution and useful screen size to that of the final monitor were mandatory.

Special hardware was developed for the simulator. It consists of a metal cover over the workstation's 19-inch display, leaving an opening similar to the Component Monitoring System's useful screen area. An HP-IL button box was modified as a keypad replacement and was integrated into the cover. The electronics of the button box were used to connect a set of hardkeys embedded into the screen bezel, forming a close approximation of the final screen bezel layout (see Fig. 2).

Hifsim has two main sections: the screen generator and the simulation section. The screen generator is basically a

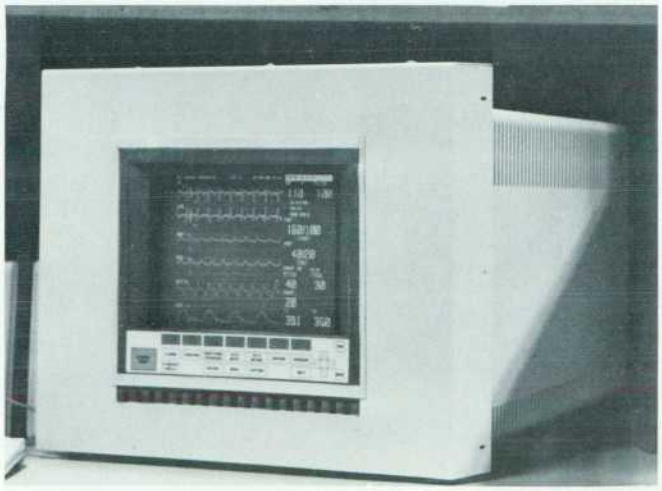


Fig. 2. The Hifsim simulator hardware interface resembles the Component Monitoring System's.

compiler that interprets the Hifsim screen definition language and converts it into commands for the HP Starbase graphics software. This language is adapted to the characteristics of the planned Component Monitoring System display hardware in terms of resolution, character sizes, fonts, colors, and special graphic elements.

The benefits of the screen generator are:

- Serves as a screen design tool
- Ensures consistency in screen design
- Enabled early selection of the Component Monitoring System color scheme
- Generates "screen cookbook"
- Supports usability tests
- Aids in software implementation
- Supports trade shows, demonstrations, and training.

Both sections of Hifsim are data driven. This means that the screen and operating dependencies are described in files. Every change in the screen content or operating sequence is implemented by editing these files while Hifsim is running. This supports the idea of interactive screen design and makes Hifsim a true screen design tool.

The monochrome version supports two intensities of green. Up to eight colors can be used in the color version of Hifsim. Each color can appear with full or half intensity. Again, similarity to the final display hardware attributes was mandatory for the simulator, and the color map of the workstation made it possible to generate any desired color. This allowed us to come up with a good definition of the Component Monitoring System color scheme under the restriction of the available hardware very early in the human interface design process.

Building a screen means specifying the screen objects along with their attributes in terms of color, size, position, line style, and so on. The ability to define waveform objects in terms of wave amplitude, trace length, position, and color was essential for the proper design of the real-time waveform display. The screen definition language supports primitives for text, waveforms, rectangles, size bars, value and alarm bars, lines, and polygons.

Hifsim made it possible for the human interface design team to visualize and distribute the screen design in the "screen cookbook", which is a collection of about 200 screen hardcopies bundled together to illustrate the Component Monitoring System human interface design. The cookbook was an essential element in the human interface design process. It was used to get clinical user and HP management feedback and approval very early in the design process.

The effort spent in building Hifsim was repaid during the implementation of the human interface software. All screen definition details were used in the actual software implementation with virtually no changes. The implementation of the interface's task window command language resembles the primitives used in the Hifsim screen definition language.

The basic functionality of the Component Monitoring System was developed jointly with the HP Waltham Division in the U.S.A., and Hifsim was used there as well for screen designs and simulation in parallel with the R&D effort at the Böblingen Medical Division. This helped achieve inherent consistency. Because the same tool was used to generate all of the Component Monitoring System screens, the screens' look and feel are consistent across all Component Monitoring System functions.

Hifsim was used widely in exposing the human interface design during shows, demonstration sessions, and marketing training at a time when no finalized Component Monitoring System hardware or software was available. This allowed the design team to get very early feedback on its user interface design.

Usability Testing

Hifsim was a prerequisite for being able to set up the Component Monitoring System usability tests. The purpose of the usability tests was to discover which features of the human interface design were effective and which needed to be improved, and to do this testing early in the design process where changes could still be made in the human interface design.

An extensive usability test session was organized in the Boston area by an independent research institute that specializes in human interface studies and human factors research. Our objective was to conduct an independent evaluation of the monitor's user interface, basically the control structure and the screen layout. The test was conducted on a sample of 13 nurses and anesthesiologists who were asked to perform typical patient monitoring tasks. A second objective was to assess the value of usability tests as an aid to the design process of a monitor's user interface.

A game plan was worked out that included a list of 30 different scenarios commonly performed by the clinical personnel in operating rooms and the ICUs. The test sessions were conducted by a moderator who first read the task scenario and then asked the test subjects to perform it. All sessions were videotaped and members of the Component Monitoring System R&D and marketing teams watched them in a separate room. This way the design engineers got firsthand insights into user reactions to the human interface.

Before each session the moderator gave a very brief explanation about how the monitor works. This demonstration was kept to a minimum to test how easy it would be for a nurse to operate the monitor with almost no previous training. The test subjects were asked before the test what functionality they expected to activate with each hardkey. In this way we got more feedback on how intuitive the Component Monitoring System keypad labeling was.

At the end of each session the test subjects were asked to pretend that they had to train the moderator to do a simple procedure, such as changing the leads on the ECG or adjusting the pressure alarm limits. The purpose was to see if they could recall the procedure they had performed about an hour ago. This was a measure of how well they had learned and how well they understood the operating concept.

The general assessment was that the Component Monitoring System user interface is sound, easy to learn, and

effective to use. A significant number of recommendations and problems were found, many of which had not been reported in previous tests with clinical specialists and HP employees. For example, the monitor has a function that allows the user to activate or suspend the monitor's alarming capability. This function was implemented in the prototype as a toggle key. To suspend alarms, the user had to press a softkey labeled Suspend Alarms. The label then changed to Activate Alarms and an ALARMS SUSPENDED message appeared in the upper part of the screen. The subjects frequently overlooked the message. They were therefore confused to see the softkey label changing to Activate Alarms. They were not sure whether the alarms were on or off. Even with extensive explanations, they had trouble understanding the functionality of the toggle softkey. Because this is a critical function that involves patient safety, we separated this function into two separate softkeys.

The recommendations from the usability tests were incorporated in the human interface design and new tests were conducted. After these were successfully passed, the human interface ERS (external reference specification) was finalized.

The usability tests were an essential milestone in the human interface design process. However, the tests only evaluated the system's ease of learning and initial ease of use. They did not evaluate how users would feel about the monitor after they had used it on a daily basis. The basic difference is that users who know the monitor don't read labels anymore. They simply push keys in a "prestored" sequence. This emphasizes the importance of consistency in the Component Monitoring System human interface design. In addition, these tests did not reflect how the user would interact with the monitor in a clinical environment, in critical situations where fast access to some basic functions is essential. Let's come back to the example of the suspend/activate alarm function, finally implemented with two softkeys. After release of the Component Monitoring System, we found that users in the operating room require a one-push key to suspend or activate the monitor's alarms. This is the way they are used to operating other monitoring equipment. In addition, having direct access to the alarm suspend function helps the user whenever special procedures are done on the patient that require alarm suspension. This led to the decision to add a hardkey on the keypad for the suspend/activate alarm function.

The verification process did not stop here. Tests in the clinical environment were conducted in the U.S.A. and various European countries before release of the Component Monitoring System to assess its usability and to test specific monitor functionality. With each Component Monitoring System release, further fine tuning of ease-of-use aspects has been done, but the main operating concepts have proved sound.

Designing for Ease of Use

To ensure intuitiveness and ease of use, a number of design decisions were made for the human interface of the Component Monitoring System. These are discussed in the following paragraphs.

Intuitive and Explicit Labeling. The human interface always uses verb+noun combinations as softkey labels (Change Lead, Adjust Alarms, Select Parameter). It always uses nouns or objects for functional entries on the keypad (**Parameters, Patient Data, Monitoring Procedures**).

In the past each front-panel control had one function, which needed a label for explanation. To keep the product's appearance unconfusing, it was necessary to abbreviate labels. This made them hard to interpret and to localize. The function of a key is much clearer if both a verb and a noun are part of the label. Then the control clearly does "something to something".

All function keys act only as softkeys—they don't have an additional meaning as a hardkey. There is always only one function assigned to a given control. There are no hidden functions and no automatic screen actions, which are perceived as unexpected, are not obvious to the user, and require extra training effort.

Task Window Appearance. All task windows or setup menus have the same layout and appear in the same position on the monitor's screen. All information needed to perform a given task is included in this window. The window height is a function of the amount of information that has to be presented. However, the basic design goal was to keep these windows as small as possible to minimize the amount of screen they cover.

Screen Eye Movement. The most critical information, such as patient alarms, is placed on the top right side of the screen. Because the vital sign numerics are critical for the patient status assessment, they always appear on the right side of the screen. Prompts and status messages are always shown in the top middle portion of the screen.

Context Sensitive Help. The Component Monitoring System help function is intended to replace the traditional instruction card. Upon request (pressing the **Help** hardkey) the system provides one or two lines of information about the functionality of the currently activated softkey or choice. In the case of multistep procedures a more detailed description of the procedure steps is shown as permanent help inside the operating window. None of the help components hides the currently active task window.

Consistency. This issue is critical for the ease of use of the monitor. The same functions are kept on equivalent keys across different operating windows (e.g., the Adjust Alarms function is always the rightmost softkey in any parameter task window). The same wording is used for a function that appears in several windows (e.g., Change Scale is used as a softkey label whenever a change in a parameter's amplitude is implemented). All softkey labels are printed with an initial uppercase character followed by lowercase letters. Highlighting is always used to indicate that a given field is currently active. Blinking is always used to indicate that an alarm condition is present. Operating the monitor from the screen bezel or the remote keypad is the same. All bezel keys appear in the same layout on the remote keypad. Rules and guidelines ensure that application software modules present task windows in a consistent way. This applies not only to the run-time task windows but also to all parameter configu-

ration windows. All have similar appearances and identical controls.

Color. Color is additional and redundant and never used as the only coding scheme. Color is used basically to differentiate real-time waveforms displayed in an overlapping fashion. Pieces of information that belong to one parameter source (such as the real-time waveform, numerics, and the trend wave) always have the same color. All operating windows have the same color (cyan) and all softkey labels are white on cyan. Alarm severity is expressed in the colors of the alert messages. Life-threatening alarms are in red, caution or warning alarms are in yellow, and inoperative conditions are shown as green messages. A red X-bell symbol is used throughout the system to indicate that alarms are turned off.

Avoiding Operating Errors. All choices for a given function are always shown. There are no hidden choices. The status of a given setting is shown before a change is initiated. Prompt messages and prompt sounds are used to inform the user if an action cannot be executed properly. Actions like pressing **Confirm** or finishing multistep procedures (e.g., zeroing a pressure line) always result in a prompt message and sound.

Graphics. In addition to digital readouts, graphic elements are widely used. This includes size bars for amplitude adjustments and audible volume control and alarm and value bars to indicate the current range of alarm limits.

User Defaults and Configuration Sets. The basic design goal is that it be possible to turn on the monitor, attach the transducers and electrodes to the patient, and start monitoring without any further settings or adjustments. This means that the monitor will initiate at power-on with a set of user-definable settings. These user defaults can be specified at installation time and changed whenever required. They are stored in nonvolatile memory and read after monitor restart. This applies to every parameter module in the Component Monitoring System.

In addition, a whole set of user settings related to one specific Component Monitoring System element, such as the display or the recorder configuration, can be bundled together and accessed by a single keypush. For example, all screen related attributes, such as waveform assignment to display channels, number of waveforms, speed of waveforms, and overlapping formats, can be put together as one screen choice. Up to three different screen choices can be stored in nonvolatile memory. This supports applications in the operating room, where depending on the course of surgery, specific screen layouts have to be accessible without complex interaction.

Finally, the concept of a configuration set supports the monitor's ease of use and flexibility by customizing the parameter algorithm behavior and the parameter settings according to the specific environment (operating room or ICU) or to the patient's age (adult, pediatric, or neonate). The user can specify or change the monitor's behavior simply by selecting one of the four available configuration sets prestored in the monitor. This again simplifies the monitor's setup in an environment like the operating room, where patients of different ages undergo surgical interventions.

The Resting Display

The resting display is what the Component Monitoring System shows when no user interaction is taking place. Fig. 3 shows a typical resting display.

Since the monitor's main task is to measure a patient's vital signs and give an alarm if a critical situation occurs, the top line is reserved for alarm information. It also contains the patient's name, the current date and time, and the basic configuration of the monitor—for example, a classification of the patient and the application area for which the internal algorithms are optimized.

The next line contains status and prompt messages informing the user about events that are not as critical as alarms, but give information about such things as successfully finished recordings or parameter calibration procedures.

Depending on the Component Monitoring System configuration and the user's choice, the resting display shows four, six, or eight real-time waveforms of the measured parameters with the digital values derived from the waveforms displayed next to them. For better vertical wave-

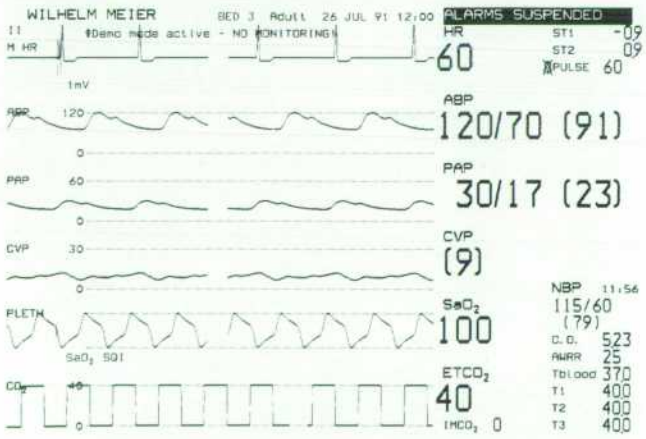


Fig. 3. Typical resting display.

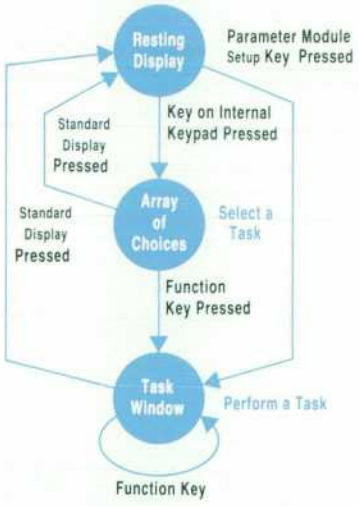


Fig. 4. General operating structure.

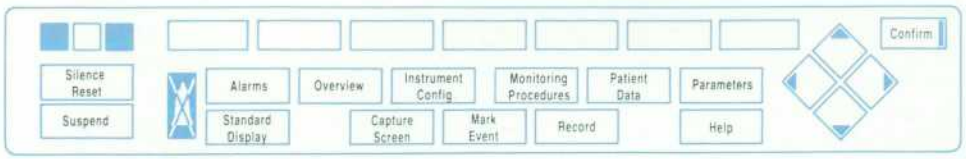


Fig. 5. The keypad.

form resolution, up to two groups of waves can share a larger sector on the screen. This overlapping of waveforms allows the user to correlate different waveforms on the time axis.

Each waveform channel can be assigned a different speed. Two presentation modes of the waveforms are possible: either the waveforms are fixed on the screen and old waveform samples are erased by the newest, or the waveforms move across the screen with the newest samples always next to the digital values.

The content of each channel can be configured. Additionally, the user has the choice of three preconfigured screens to make it possible to adapt quickly to changes of the patient's condition.

Digital Values

The user has definite expectations about where, when, and in what format digital values should appear. The requirements for the arrangement of the digital values were:

- The values of a parameter plugged into a rack or turned on should show up automatically. It is unacceptable to have to position the value of such a parameter manually. It must appear in the right position.
- The digital values must be placed next to their corresponding waveform if possible.
- As many values as possible have to be shown with large digits. On a full display it is acceptable for the less-important values to be shown with small digits, but not on a display with just two measured parameters.

These requirements are met by an elaborate algorithm. It is an iterative process that tries to find a place for all digital values available in the system. It first places all values next to their waveforms with large digits. It then places all other values, according to a priority list, in the right column next to the waveform values. Temperature values are first assigned large digits.

If there are still unassigned numerics left but no more space available, the algorithm starts decreasing values in size, starting with those of lowest priority, and repeats the process. As a last resort, temperature values are allowed to share the same place, alternating at two-second intervals.

Operating Concept

The general operating structure of the Component Monitoring System human interface is described by the state diagram shown in Fig. 4.

Keypad. After the keys on the parameter modules, which are mainly used to enter the menus and adjust parameter settings, the keypad underneath the screen is the main tool for users to interact with the monitor. Fig. 5 shows this keypad. As mentioned above, a handheld keypad for remote operation has some additional functions available.

The first row of keys on the keypad consists of seven function keys. Their functions are defined by the menus that appear on the screen.

The next row of keys is used to enter six different categories of monitor interaction.

The lowest row contains keys that immediately start actions that are frequently used in the hospital's daily routine. The **Standard Display** key always returns control to the resting display.

The **Silence/Reset** key is used to silence or reset alarms. The **Suspend** key is used to suspend or activate instrument alarm capability. There are alarm-indicating LEDs on the left and a diamond of four cursor keys and a **Confirm** key on the far right. The last group of keys gets highlighted if they can be used.

The Array of Choices. If one of the keys in the middle row is pressed, the user immediately gets a display of all of the interactions that belong to the category described on

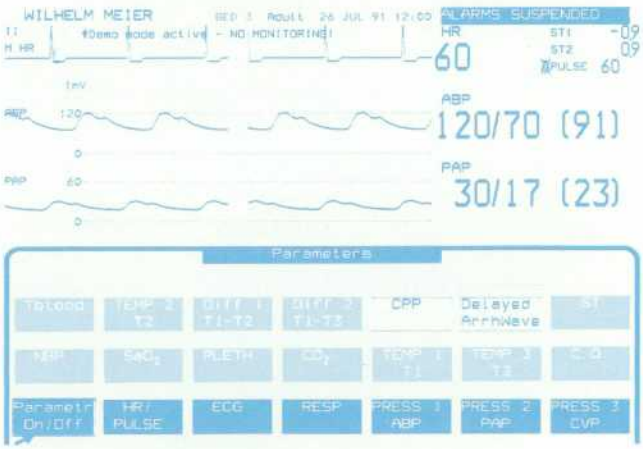


Fig. 6. The **Parameters** array of choices.

the entry key (see Fig. 6). There are no hidden functions that the user might remember but does not know where to look for.

Behind some of the entry keys there can be more than seven functions. Thus the user is shown an array of choices with up to four possible lines of softkeys. The number of entries depends on the category and the configuration of the Component Monitoring System. The active line is shown in full intensity. It can be moved up and down either by repeatedly pressing the entry key or with the cursor keys.

The array of choices illustrates three important mechanisms that occur consistently throughout the operating concept.

- Resources, such as the place occupied by the array of choices, are used according to the system configuration.
- Selected items are shown in full intensity
- Two methods are consistently allowed for selecting an item: either by repeatedly pressing the key that was used to enter the context, or by using the cursor keys. Usability testing has shown that there are personal preferences for either method depending on the user's background. As a third method, touch would not clash with the operating structures, although it is not offered at this time. Inverse areas in half intensity could be activated by touch.

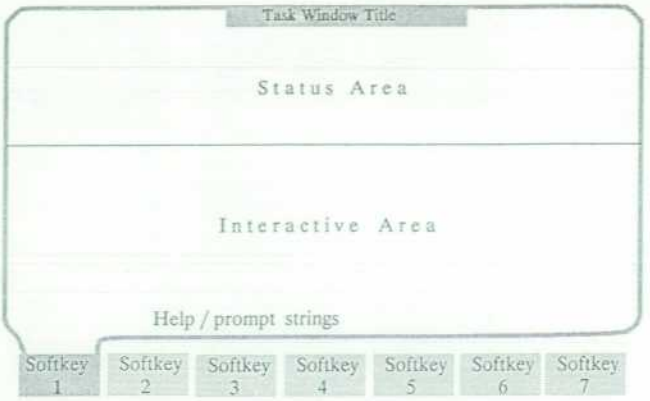


Fig. 7. The generic layout of a task window.

Task Windows. The array of choices is an intermediate step in entering the next operating level, the task window. Fig. 7 shows the generic layout of a task window. The possible functions are labeled with inverse softkeys, which do not change in this context. The currently active function is highlighted and linked to the interactive area above, which can contain items to be selected or special contents needed for this specific softkey.

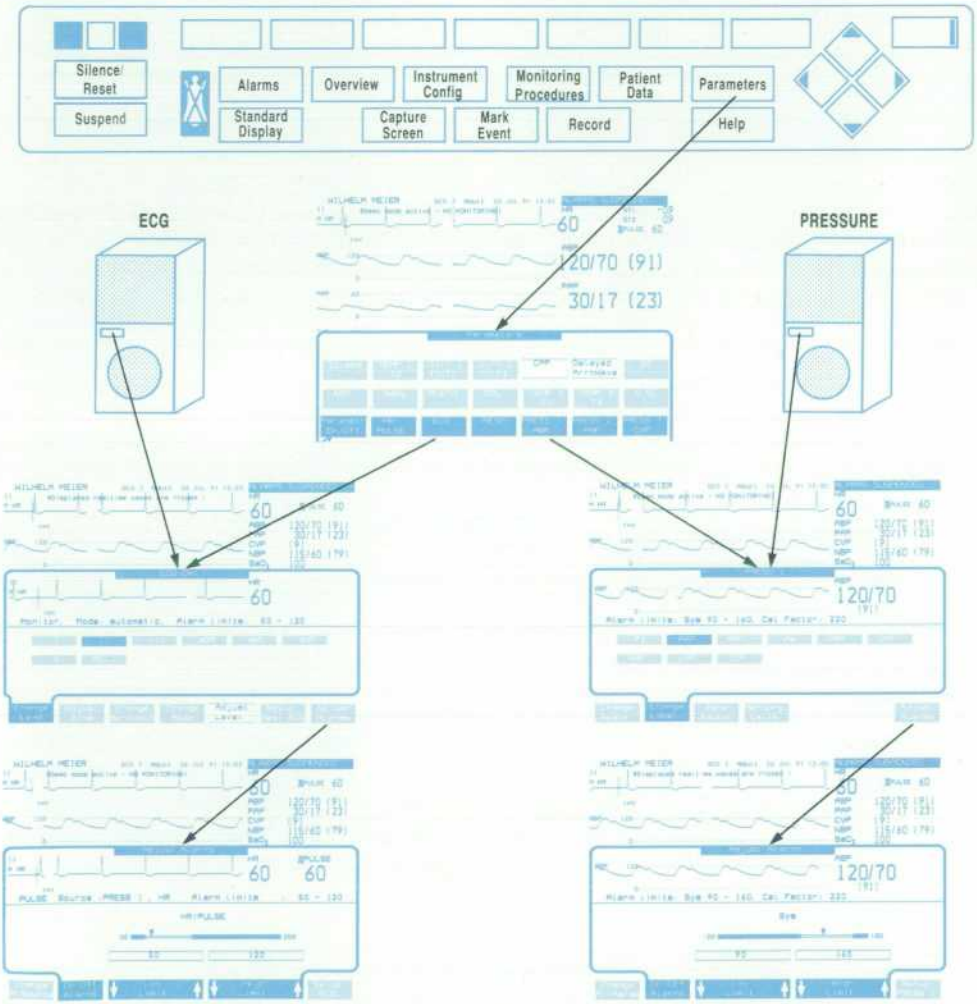


Fig. 8. Overview of the Component Monitoring System operating levels.

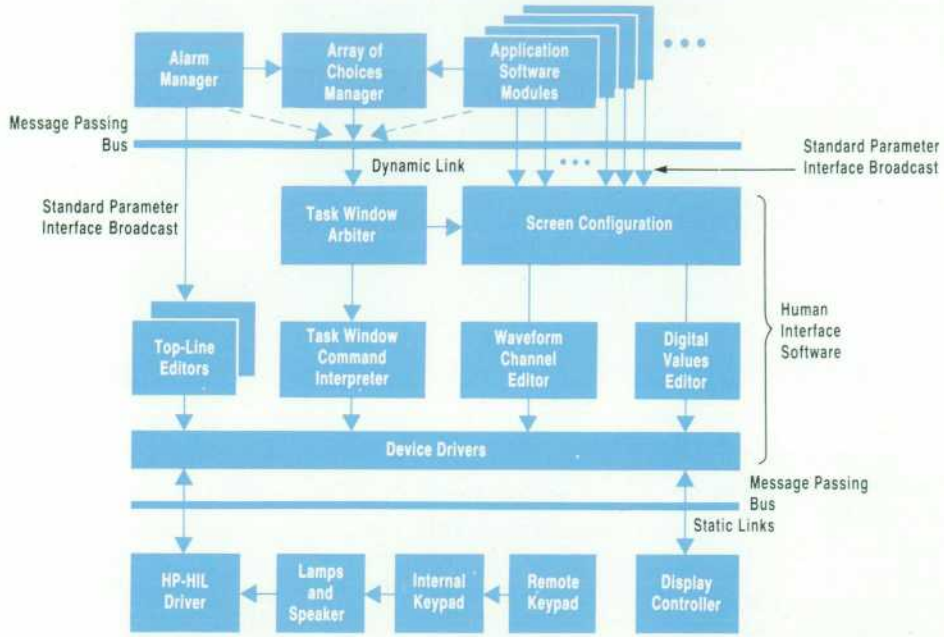


Fig. 9. Layered model of the human interface software.

The status area underneath the title contains all status information about the current task—including a real-time waveform if available—to make sure that the user does not have to select a function just to get more information. The user only needs to press a softkey if something is to be changed.

If required, the rightmost softkey can be used to jump back and forth to a subsequent task window. As an example, Fig. 8 shows an overview of the operating levels for three parameter setups.

Human Interface Software Architecture

The human interface software is embedded in the overall Component Monitoring System software architecture. It is one of the large data sinks that make intensive use of the communication model with its message passing concept. The well-structured information provided, for example, by the standard parameter interface (see article, page 19) makes it possible to add new parameters with virtually no changes to the human interface software. It also allows resources to be used very effectively by allocating memory depending on the number of messages to be processed.

Fig. 9 shows the layered structure of the human interface software module. Each parameter module, even a new one, broadcasts its standard parameter interface messages and is automatically recognized by the screen configuration software. To get a task window, any application software module either applies directly to the task window arbiter or specifies an entry in the array of choices. If selected, the array of choices manager arranges a dynamic link between the parameter module and the task window arbiter.

Any application software module can present information in a task window by using a command language that supports the specific elements of the human interface. This is an effective way to achieve the required consistency across all task windows. The content of the task windows is determined by the application, but human interface related definitions are coded in the command language. Thus, most changes affecting the human interface design have to be done in the human interface software only.

A powerful standardized keyhandler builds the interface between the application software and the task window command language. The command language hides the pixel coordinates of the display controller from the application software. Thus, the application software does not have to be changed in case the display technology changes, for example to LCD. The coordinate system of the task window command language is the same as was used during the human interface screen simulation.

There are asynchronous FIFO buffers in the path between the task window commands and the connected hardware devices, mainly the display controller. A special handshake mechanism based on the monitoring of token messages guarantees that the FIFOs cannot be flooded in peak situations.

By setting up the human interface module two or more times in the monitor configuration table (see article, page 13) and plugging more display controller cards into the computer module, several independent displays can be connected to one Component Monitoring System.

Acknowledgments

Many thanks to Steve Emery of HP's Waltham Division, whose assistance and application knowledge gave us valuable help in defining the human interface.

Globalization Tools and Processes in the HP Component Monitoring System

Software design and localization are decoupled. All languages are treated in the same way. A database contains the text strings for all languages, and automated tools aid the translator.

by Gerhard Tivig

The HP Component Monitoring System is an international product designed for a worldwide market. Among the requirements for the product were introduction of localized versions simultaneously with the shipment of the standard product, full Asian language support, and low incremental effort for localization in any new language.

At first release, the product was localized in the following languages: English, German, French, Dutch, Swedish, Italian, and Spanish. A Kanji/Kana prototype version was available as well. The current release is also localized in Danish, traditional Chinese, and simplified Chinese.

Localization Goals

To fulfill the requirements, a number of goals were set forth very clearly in the design phase of the Component Monitoring System software. The major goals were the decentralization of localization efforts, the automation of the localization process, and the standardization of interfaces.

Decentralization. Decoupling the software design and implementation process (R&D responsibility) from the localization process (technical marketing responsibility) makes it possible to produce a localized Component Monitoring System without interrupting the software engineers working on their software modules. The coordination and timing of the translations are not directly coupled with the software development process.

Automation. Automated processes to generate localized Component Monitoring System software allow efficient generation of localized versions whenever they are needed, especially in prerelease phases (regulatory approval, clinical trials, demonstrations, etc.). The automated processes transform all Component Monitoring System text strings from plain English to the equivalent hexadecimal character representation. Automatic format checking is part of this process. Translation of all text strings of a Component Monitoring System software release in a single pass improves the consistency of the translated text—similar terms are translated the same way in various places. The same translator is responsible for text strings and for the Operating Guide translation.

Standardization. A well-structured native language support (NLS) database is needed. The generation process for localized software and the translation process are the clients of this database. The NLS database is part of the Component Monitoring System software maintenance system.

Simple and standardized interfaces between the components of the localization process are necessary. This includes common file formats for the NLS database, common tools for accessing and handling text strings, and common tools and processes to translate and generate localized software.

Design Decisions

Specific design decisions had to be made to achieve these goals. Among these are:

- The HP standard Roman8 character set is supported. This allows localization of up to 14 Western European languages with one Roman8 character generator, which is located on the display controller function card. This considerably simplifies the handling of European language options.
- All character codes are two-byte codes. Thus all text strings use two-byte character codes. This allows support of Asian languages as well as all European languages in a consistent way. For Roman8 characters, the upper (unused) byte is cleared.
- A given text string has a fixed field length across all languages. Thus the field length of a given text string is not language dependent and the access of a software module to its text strings is language independent. In addition, all text strings are terminated with an end-of-string charac-

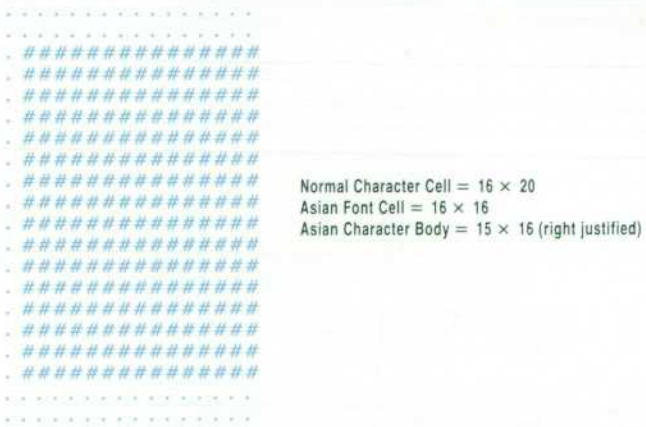


Fig. 1. Component Monitoring System standard character cell.

ter. There is no language dependency in the way strings are handled in different languages.

- Text strings are separated from module code. All software modules are language independent. Inside each software module, all text strings are located in TEXT directories, thus being separated from the code (PROG) directory. Changing text strings from one language to another does not affect the Component Monitoring System code. No recompilation of the software is necessary when a new localized Component Monitoring System version is produced.
- Standard HP16 codes for all Asian languages are supported. This allows the Component Monitoring System to handle all Asian languages identically and supports the connection of Asian printers as well. For each Asian language, a specific Asian EPROM card with the complete font set is supported.
- The standard character cell supports all Asian language fonts. The standard character cell is 16 pixels wide by 20 pixels high. The Asian fonts (Kanji/Kana, Chinese) are handled as right-justified 15-by-16-pixel characters (see Fig. 1).

A pixel is 0.219 mm wide by 0.352 mm high, giving an aspect ratio of 1.6. An Asian character should have a square appearance, so the display controller firmware doubles each pixel in the x dimension. This means that a Kanji character takes twice as much space in a horizontal string as a Roman8 character. Since each Kanji character occupies two normal character cells, all Asian strings are limited to half the length of Roman8 character strings. The Asian translation tool takes this restriction into consideration. Fig. 2 shows the traditional Chinese translation of a typical Component Monitoring System task window.

The NLS Database

The NLS database contains all strings that are visible to the Component Monitoring System user. They show up mainly on the screen, but may also be present on the keypad and the patient parameter module panels.

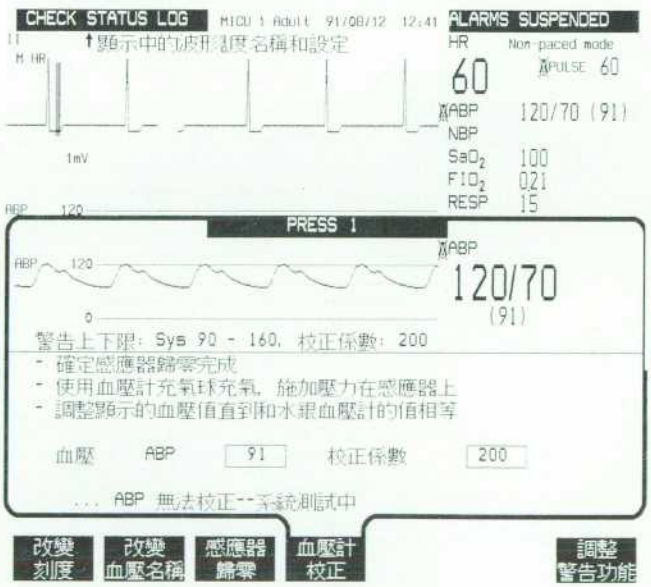


Fig. 2. Traditional Chinese translation of the pressure calibration task window.

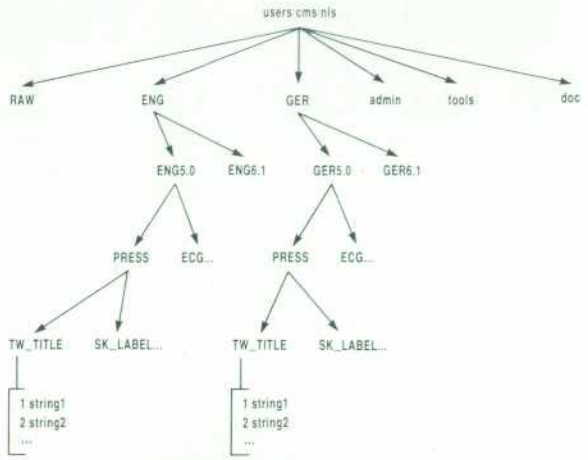


Fig. 3. NLS database structure.

The database is organized under the HP-UX file system as hierarchical file directories (see Fig. 3).The database is an integral part of the Component Monitoring System documentation and is maintained with the HP-UX rcs utility.

Below the main entry, a distinct entry called a LANG(uage) tree is provided for each language. There is a basic directory where all localizable strings are stored in plain English. This is the RAW directory. Its structure is identical to all of the LANG trees but it is not known to the translation tool and to the text compiler. Whenever text strings are added, deleted, or changed, this directory must be updated.

The LANG trees are organized as a collection of valid NLS revisions, such as ENG/ENG6.1 or GER/GER6.2. The English revision is the starting point for all further translation activities. All LANG trees have an identical structure and are composed of a collection of NLS entries such as ECG, PRESS, and so on. Each software module has one NLS entry in the database. Keeping all text strings of one software module together eases and improves the translation of the text strings of that module.

Each NLS database entry contains a set of NLS files that incorporate the text strings. A standard file format is established for all NLS files (see Fig. 4a). It is processed by the NLS tools and recognized by the translation tool. NLS files contain title, header, context, and text sections. The title section contains the pathname, language, and revision of the NLS file. The header section is a list of format specifications, such as .sz for string size or .ic for initial caps, which are read by the hexpander tool (see below). In the context section, advisory information is given to the translator to make translation of that NLS file easier. It is read only by the translation tool. The text section is the body of the NLS file. It contains a sequence of items, each item identified by a text code and a text string.

NLS Tools

The hexpander and the syntax checker are used to generate and check the hexadecimal character strings in the ENG directory. The text compiler interfaces the NLS database to the C source code. Fig. 5 shows how the NLS tools interact with the NLS database.

The hexpander takes the plain English text from the RAW directory and generates the hexadecimal character strings in the ENG directory according to the format specifications (see Fig. 4b). A utility called make_hexpand automates the process of generating and checking the syntax of the hexpanded ENG strings.

The syntax checker reads the hexpanded files and flags syntactically incorrect strings (e.g., too long). A similar checker is incorporated into the translation tool to check the hexpanded ENG NLS file before translation takes place.

The text compiler links the C source code with the NLS database, which contains text strings collected in files. References to these files include the language, the module entry (such as ECG or HEART), and the specific file containing a given class of text strings (such as SK_LABEL or ALERT). Fig. 4 shows an example of the ECG/SK_LABEL text file.

In the TEXT directory, the programmer specifies a source file (of class .txt) which contains the references to the NLS files, such as !ECG/SK_LABEL 2.1. The .txt file is identical to the .c file except that it has the NLS file references, (preceded by the escape character !), which must be resolved before the file can be compiled (using make). The

```
Title     "CMS TEXT FILE                          -ECG/SK_LABEL
              Language: ENG
              $Revision: 8.5 $
              $Source: /users/cms/nls/RAW/ECG/RCS/SK_LABEL,v$

Header    "Format Specs:
              .ic       "initial caps
              .ce       "centered within each of the two lines
              .sz8      "string length (NORMAL characters)

Context   This file contains the softkey labels for all softkeys in the ECG
          task windows. In the task windows the softkey labels consist of 'verb + noun'
          because they relate directly to an action. All softkey labels except those containing
          parameter labels (e.g., PRESS) are written with initial capitals.

Text      "Softkey 6/Page 0
          2.1 "Adjust"
          2.2 "Alarms"

          [ Enter the alarm page of the ECG/HR parameter ]
```

(a)

```
          " Softkey 6/Page 0
          2.1 "Adjust"
          | 0020 0041 0064 006A 0075 0073 0074 0020 |

          2.2 "Alarms"
          | 0020 0041 006C 0061 0072 006D 0073 0020 |

          [ Enter the alarm page of the ECG/HR parameter ]
```

(b)

Fig. 4. (a) Example of an NLS file from the RAW directory. (b) Text section of the hexpanded file in the ENG directory.

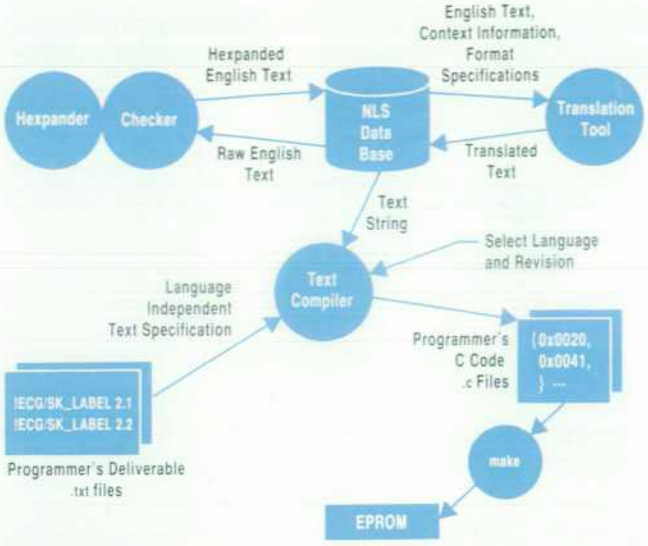


Fig. 5. NLS tools.

text compiler resolves these references. It reads the hexpanded NLS file (Fig. 4b), extracts the hexadecimal equivalents of the referenced text strings, and replaces the NLS file references in the .txt file with these strings. The output is the C source file (of class .c).

Thus, the text compiler is simply a preprocessor that takes care of the text strings. The programmer of the software module does not have to know what hexadecimal strings are ultimately loaded into the .c file. This is language dependent and does not affect the .c code.

To automate this process, the make_lang utility was established. This utility is a script that executes the text compiler for every software module that contains references to localizable text strings. The text compiler resolves these references by inserting in the indicated places in the .txt files the hexadecimal equivalents of the text strings. The output .c file is then compiled by the make utility in the usual way. An example of calling the make_lang utility is:

```
make_lang "NLSREV=7.2" "LANG=ENG"
```

Localization Process

The process established to implement the localization activities is shown in Fig. 6.

R&D is responsible for the generation and maintenance of the NLS database and the RAW and ENG language trees. The checked-in ENG revision is the starting point for all translations. This ENG tree is provided to the technical marketing group together with a delta list containing all changes from the previous ENG revision. This group is responsible for driving and coordinating the translation process. When this process is complete, the translated LANG tree is loaded back into the NLS database. Automated utilities such as make_cms are used in R&D to generate the localized Component Monitoring System software. R&D is responsible for providing the EPROM cards with the localized software. A quality assurance cycle similar to that for the ENG version is then started for each localized version. Part of the QA process is a consistency

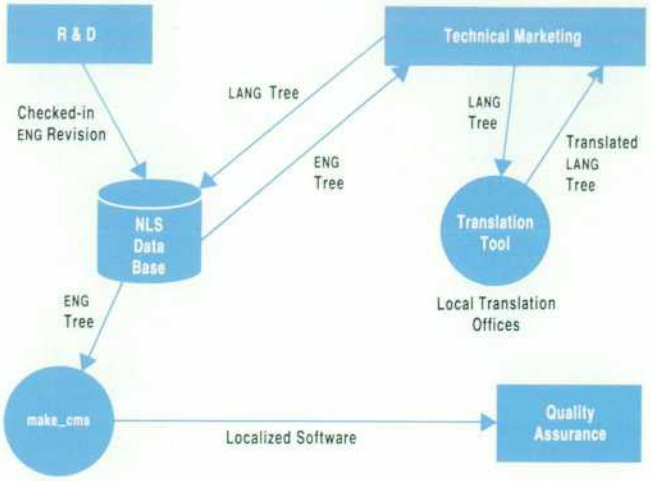


Fig. 6. The localization process.

and wording check of the localized software. The language verification is scheduled and coordinated by technical marketing. The same translator who did the Component Monitoring System translations is assigned to language verification of the Component Monitoring System product.

Translation Tool

The localization goals could not have been achieved without a powerful and versatile translation tool. Because nothing was available off the shelf, we had to write our own. The tool is personal-computer-based, thus allowing translations of the Component Monitoring System text strings in each local HP office. The tool supports 16-bit character codes. It handles the standard Roman8 character set and allows printing of the translated strings on a HP LaserJet printer.

The tool is designed to facilitate translations of large quantities of text. The tool reads the English source files and presents the translator with the destination fields for the translated strings, which are then written to the respective LANG tree.

Translation is possible for a complete LANG entry, for specific NLS entries (e.g., all strings of the ECG software module), or for specific text files inside one NLS entry. Printouts can be made from each of these translation levels. The user interface is softkey driven.

Presently, this tool is used throughout the HP Medical Products Group and is supported by the CAD/productivity group at the Böblingen Medical Division. It allows efficient translations with a clear, standard interface to the NLS database and the Component Monitoring System software development group. Its major benefit and achievement is the separation of translation activities from the software development efforts.

Acknowledgments

Many thanks to Ulrich Muskatiewitz who wrote the translation tool and extended it to cover Asian languages, to Satoshi Yamada of Yokogawa-Hewlett-Packard for his extensive support in implementing Asian languages, and to Harald Greiner who was responsible for implementing the Asian fonts and associated control mechanisms in the display controller firmware. Thanks also to Martin Katschner whose continuous support of the translation tool and translation activities allowed us to localize the Component Monitoring System product in seven languages on time for the first release.

The Physiological Calculation Application in the HP Component Monitoring System

This application converts raw real-time data into derived values the clinician can use to assess the patient's hemodynamic, oxygenation, and ventilatory condition.

by Steven J. Weisner and Paul Johnson

The HP Component Monitoring System bedside monitor provides the clinician with a variety of vital-sign parameters such as heart rate and respiration rate. These raw values and the associated alarms are very important in monitoring the patient. However, the human body is not a collection of independent physiological systems. Rather,

all major body systems interact in a variety of ways, many of which can be calculated by combining the raw parameter values into meaningful indicators.

Physiological calculations are used routinely by many hospitals as part of their normal assessment and record-keeping process. Calculations provide a way of quickly reducing a large number of variables into a single number that represents a comprehensive physiological function. For example, to measure the load applied to the left ventricular heart muscle during the period of the heartbeat when the blood is ejected from the heart into the rest of the body (ventricular ejection), a variable called systemic vascular resistance (SVR) can be calculated from measurements of the mean arterial blood pressure (ABPm), central venous pressure (CVP), and cardiac output (CO).[1]

Studies have shown that calculated values such as pulmonary vascular resistance (PVR) and left and right cardiac work (LCW/RCW) are good predictors of major malfunctions or mortality in intensive care patients.[2] Other studies have validated the efficiency of using calculations such as stroke index (SI) and left and right ventricular stroke work (LVSW/RVSW) for preoperative assessment of unacceptable risks for major surgery.[3]

The Typical Calculation

Poiseuille's law describes the laminar, constant flow of Newtonian liquids through rigid cylindrical tubes. According to this law, the ratio of pressure drop to the rate of flow is a function of all of the forces that retard this flow (i.e., radius, length, and viscosity). Blood does behave as a Newtonian fluid in blood vessels that are greater than 0.5 mm in diameter. Blood flow through these vessels is generally laminar, although the arterial tree exhibits more pulsatile behavior. Although blood vessel radii do vary slightly because of the applied pressure of the blood, Poiseuille's law can be used to calculate a first-order approximation of resistance by applying Ohm's law for electrical circuits.

Just as resistance in a circuit is equal to the voltage difference divided by the current flow, vascular resistance (R) can be approximated by dividing the pressure difference between the inlet of the vascular bed (P1) and the outlet of the bed (P2) by the blood flow (Q).

$$R = (P1 - P2)/Q.$$

In medical terms, we measure the difference between the mean arterial (ABPm) and venous (CVP) pressures and divide by the cardiac output (CO). The resultant value is converted from units of mmHg/l to units of dyne-s/cm^5 by multiplying times 79.97. This value is called systemic vascular resistance (SVR).

$$SVR = 79.97(ABPm - CVP)/CO.$$

With this value, the clinician can get a measure of the constriction of blood vessels (vasoconstriction) or expansion of the blood vessels (vasodilation). Changes in SVR are related to other cardiac failures such as hypovolemic shock, left ventricular failure, cardiogenic shock, and hypoxemia.[4]

Other calculations used to assess the state of the cardiovascular system are shown in Fig. 1.

The two pressure measurements ABP and CVP are acquired through invasive pressure catheters attached to the patient and monitored through the Component Monitoring System parameter modules. The cardiac output parameter is obtained through a CO parameter module and measured using a monitoring procedure, which requires the clinician to interact with the Component Monitoring System. The acquisition of the output value (SVR in this case) and the presentation of the calculations to the clinician are described in the following sections.

Data Management Package

The calculations package in the Component Monitoring System is a subset of a more general data management package. This package consists of seven Component Monitoring System application software modules, as shown in Fig. 2. The data acquisition module acquires and averages raw parameter data (e.g., heart rate) over a one-minute period. This raw data is available as a broadcast message on the Component Monitoring System's internal message passing bus. The one-minute-average data is stored in a buffered RAM database. The database module provides 24 hours of data storage for 16 continuously monitored pa-

Body Surface Area: (Boyd's Formula)

$BSA = (3.207 \times WT^{(0.7285 - 0.0188 \times \log WT)} \times HT^{0.3}) / 10000$ Units = m^2

Cardiac Index :
$CI = CO / BSA$ Units = $l/min\text{-}m^2$

Stroke Volume :
$SV = CO \times 1000/HR$ Units = ml

Stroke Index :
$SI = SV / BSA$ Units = ml/m^2

Systemic Vascular Resistance :
$SVR = 79.96 \times (ABPm - CVP)/CO$ Units = dynes-sec/cm^5

Systemic Vascular Resistance Index:
$SVRI = SVR \times BSA$ Units = dynes-sec-m^2/cm^5

Pulmonary Vascular Resistance :
$PVR = 79.96 \times (PAPm - PAWP)/CO$ Units = dynes-sec/cm^5

Pulmonary Vascular Resistance Index :
$PVRI = PVR \times BSA$ Units = dynes-sec-m^2/cm^5

Left Cardiac Work :
$LCW = CO \times ABPm \times 0.0136$ Units = kg-m

Left Cardiac Work Index :
$LCWI = LCW / BSA$ Units = $kg\text{-}m/m^2$

Left Ventricular Stroke Work :
$LVSW = SV \times ABPm \times 0.0136$ Units = g-m

Left Ventricular Stroke Work Index :
$LVSWI = LVSW / BSA$ Units = $g\text{-}m/m^2$

Right Cardiac Work :
$RCW = CO \times PAPm \times 0.0136$ Units = kg-m

Right Cardiac Work Index :
$RCWI = RCW / BSA$ Units = $kg\text{-}m/m^2$

Right Ventricular Stroke Work :
$RVSW = SV \times PAPm \times 0.0136$ Units = g-m

Right Ventricular Stroke Work Index :
$RVSWI = RVSW / BSA$ Units = $g\text{-}m/m^2$

WT – Body Weight in g
HT – Body Height in cm
HR – Heart Rate in beats/min
CO – Cardiac Output in l/min
ABPm – Arterial Blood Pressure Mean in mmHg
CVP – Central Venous Pressure in mmHg
PAPm – Pulmonary Arterial Pressure Mean in mmHg
PAWP – Pulmonary Arterial Wedge Pressure in mmHg

Fig. 1. Hemodynamic calculations performed by the calculation evaluator module of the Component Monitoring System data management software package.

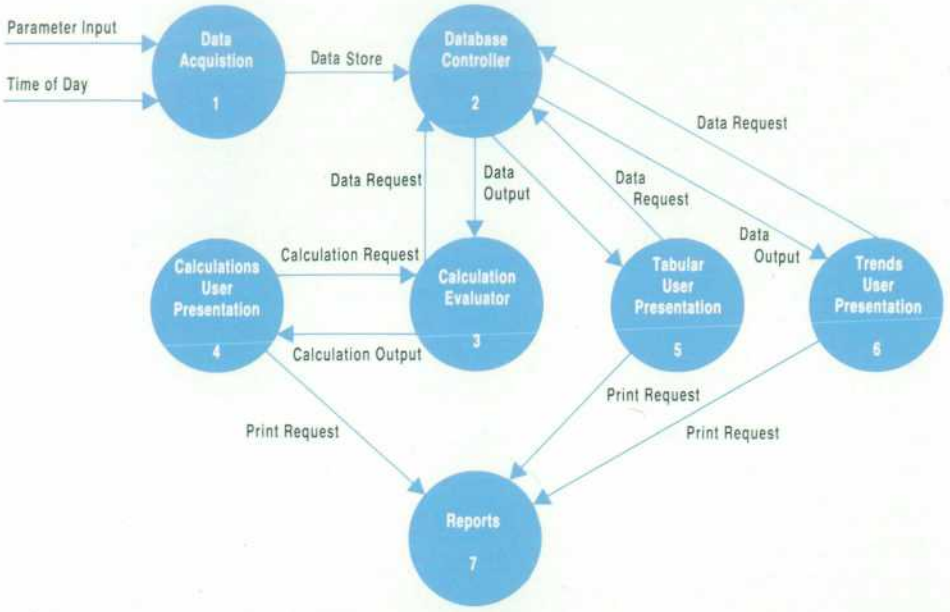


Fig. 2. Data management data flow diagram. The data management package consists of seven application software modules.

rameters with one-minute resolution. Parameters that are measured intermittently or as part of a procedure, such as noninvasive blood pressure or cardiac output, are referred to as aperiodic parameters. The database module allows storage of 36 aperiodic parameters, each containing up to 96 measurement points. All retrieval of the data is mediated by request messages and return-data messages sent across the message passing bus.

The acquired data can be presented in four forms. A tabular data display (UPC_TABULAR) presents 13 rows of parameters in eight columns of time. A graphic trends display (UPC_TRENDS) shows up to nine parameters in graphic form on three separate axes. Calculations are done by two modules: the calculation evaluator (CALC), which performs the calculations, and the presentation module (UPC_CALC), which provides the user interaction with hemodynamic, oxygenation, and ventilation calculations. The clinician can also review the calculated data as a function of time in a tabular format.

Finally, there is a report package, which provides printed copies of any of the tabular, trends, or calculation displays. This report is preformatted and can be printed locally at the bedside or remotely on a central printer.

Calculation Evaluator

The calculation evaluator module (CALC) is a collection of services associated with physiological calculations. These services are invoked by means of messages sent to the CALC module. Typically, applications such as UPC_CALC invoke the functions of acquiring the appropriate reference time and input parameters for the calculation and then calculating the output values.

In addition, the CALC module provides a separate service to calculate the body surface area (BSA), which is used as a common index for many physiological calculations, and is also used outside of the data management package by the cardiac output module.

Calculation Presentation

The clinician obtains the services of the calculation evaluator through the user presentation module of the calculations software, UPC_CALC. UPC_CALC is a single Component Monitoring System application that provides access to both calculation entry and calculation review frames.

The first step in performing physiological calculations is for the clinician to select a physiological calculations group from the set of predefined options: hemodynamics, oxygenation, and ventilation. This is accomplished by selecting the appropriate entry key in the **Patient Data** array of choices. Once this is done, the Component Monitoring System human interface software establishes a link between UPC_CALC and the monitor display. The calculation entry frame is shown in Fig. 3.

After the calculation group has been selected, the clinician can then perform one or more of the following ac-

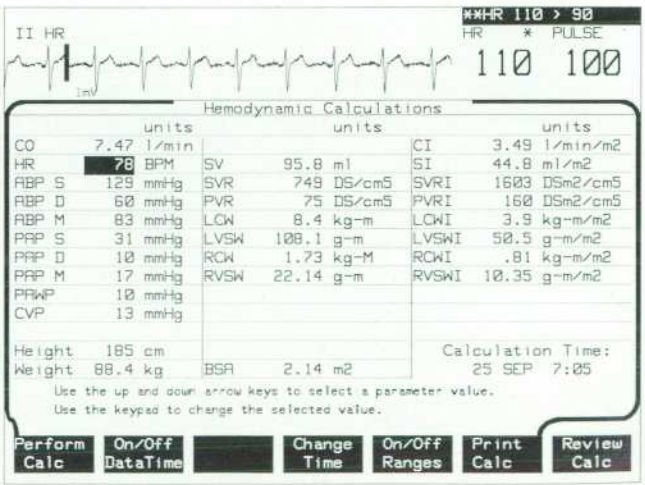


Fig. 3. Hemodynamics calculation entry frame after calculations have been performed.

R.G.Mortis Bed : Report :25SEP1989 14:31
MR # :123-456-7890 Adm Phys :M.Craig Admit :09-25-89
DOB :02-23-48 Att Phys :P.White

Hemodynamics Review

25SEP	7:05	8:00	9:15	9:50	10:15	10:30	11:00	13:10
SV	95.8	102.9	78.3	74.7	80.6	87.5	89.4	96.5
SVR	749	877	1021	980	926	925	928	955
PVR	75	78	91	79	99	109	112	115
LCW	8.4	8.6	9.6	9.5	9.1	9.2	8.8	8.5
LVSW	108.1	125.9	106.5	99.6	100.8	109.5	109.4	118.2
RCW	1.73	1.86	2.01	1.83	1.87	2.00	1.94	1.42
RVSW	22.14	26.58	22.37	19.31	20.82	23.80	24.31	19.69
BSA	2.14	2.14	2.14	2.14	2.14	2.14	2.14	2.14
C.I.	3.49	3.36	3.29	3.32	3.39	3.43	3.34	3.25
SI	44.8	48.1	36.6	34.9	37.6	40.9	41.8	45.1
SVRI	1603	1878	2184	2097	1983	1979	1986	2044
PVRI	160	166	194	169	212	233	239	246
LCWI	3.9	4.1	4.5	4.4	4.2	4.3	4.1	4.0
LVSWI	50.5	58.8	49.8	46.5	47.1	51.2	51.1	55.2
RCWI	.81	.87	.94	.86	.88	.93	.91	.66
RVSWI	10.35	12.42	10.45	9.02	9.73	11.12	11.36	9.20
C.O.	7.47	7.20	7.05	7.10	7.25	7.35	7.15	6.95
HR	78	70	90	95	90	84	80	72
ABPS	129	135	139	135	130	130	120	115
APBD	60	65	69	65	60	62	75	70
ABPM	83	90	100	98	92	92	90	90
PAPS	31	33	34	32	30	31	28	27
PAPD	10	12	13	12	10		10	9
PAPM	17	19	21	19	19	20	20	15
PAWP	10	12	13	12	10	10	10	5
CVPM	13	11	10	11	8	7	7	7
Height	185	185	185	185	185	185	185	85
Weight	88.4	88.4	88.4	88.4	88.4	88.4	88.4	88.4

Fig. 4. Hemodynamics calculation review frame.

tions by using the labeled softkeys and a remote keypad for alphanumeric entry:

- Select a calculation time
- Enter or edit input parameters
- Calculate output parameter values
- Display alternate parameter attributes
- Print a report of the calculations.

UPC_CALC uses the measurement time of the principal input parameter as the reference time for all calculations. In the case of hemodynamic calculations, shown in Fig. 3, the cardiac output parameter drives all of the other calculations. Thus the time of the last CO measurement is used as a reference. All other input parameters used in the calculations are retrieved from the data management package database through a CALC module service, based on that reference time. The clinician can choose to override this time by using the Change Time key to select a different reference time.

Not all input parameters used to perform calculations are automatically acquired by the Component Monitoring System. By using the remote alphanumeric keypad, the clinician can enter a numeric value for any of the input parameters. The clinician can also override an automatically acquired parameter simply by entering a new value. All of these manually entered values are stored in the database and are used in subsequent calculations.

Once all the necessary calculation time and input parameter changes have been made, the clinician can request calculations of the output parameter values. UPC_CALC sends a request to calculate the output values to the CALC module, which performs the appropriate calculations. CALC then sends a return message back to UPC_CALC containing the list of output parameter values, labels, normal ranges, and measurement units. UPC_CALC uses this information to show the output values on the Component Monitoring System display.

The clinician may wish to compare the output values to the expected normal physiological ranges for these values. When the ON/OFF Ranges softkey is pressed, UPC_CALC toggles between showing the output parameter units and the normal ranges.

UPC_CALC also serves as the presentation layer software for the calculations review frame. The review frame presents the clinician with a tabular format of all previous calculations performed for this patient. As in the calculations entry frame, the clinician can compare values against normal ranges and obtain a printed report, as shown in Fig. 4. Typically, this report might be included with the patient record to aid the clinician in assessing the patient's past and current physiological states.

Conclusion

The Component Monitoring System data management calculations package provides the clinician with a means of reducing the large volume of raw vital-signs data into a manageable set of variables. Measures of cardiovascular performance, blood oxygen content and delivery, and respiratory gas exchange can be obtained through the hemodynamic, oxygenation, and ventilation calculations. These calculations are vital to the clinical diagnosis and prognosis of the critically ill patient.

Acknowledgments

The authors greatly acknowledge the efforts of the data management project R&D, software quality engineering, marketing, and management teams, specifically Rick Beebe, Krishen Bhan, Jeff Corliss, Peter Dabos, Debbie DeRosa, John Doue, Joe Dumas, Dave Ellis, Sandy Foglesong, Bill Francis, Kathryn Graham, Jack Harrington, Carla Joliat, Jean-Luc Kastner, Wolfgang Krull, Fran Michaud, Ram Mukunda, Tasha Perdew, Kari Perry, Egon Pfeil, Sue Poliner, Jim Rueter, Linda Straw, Charlotte Swartz, Paul Tessier, Gerhard Tivig, and Jack Ward.

References

1. D. Pollard and E. Selliger, *An Implementation of Bedside Physiological Calculations*, Hewlett-Packard Company, 1985, HP publication number 5954-1779.
2. W.C. Shoemaker and L.S. Czer, "Evaluation of the Biologic Importance of Various Hemodynamic and Oxygen Transport Variables," *Critical Care Medicine*, Vol. 7, no. 9, 1974, pp. 424-431.
3. L.R. Del Guercio and J.D. Cohn, "Monitoring Operative Risk in the Elderly," *Journal of the American Medical Association*, Vol. 243, no. 13, 1980, pp. 1350-1355.
4. S.S. Yang, et al, *From Cardiac Catheterization Data to Hemodynamic Parameters*, F.A. Davis, Philadelphia, 1972.

Mechanical Implementation of the HP Component Monitoring System

The part count and the number of different parts are dramatically lower than for previous designs. Fewer than ten vendors are used for purchased mechanical parts.

by Karl Daumüller and Erwin Flachsländer

From the mechanical perspective, the HP Component Monitoring System offered several challenges. Among the most important were the definition of the architecture of the computer module and the design of the sheet-metal and plastic parts for this component. Other mechanical highlights include the implementation of the display front assembly and the construction of the parameter modules.

Computer Module Chassis

The general design objective for the computer module was to create a flexible, compact instrument that could easily be extended and upgraded. In accordance with the modular concept of the Component Monitoring System, the computer module had to be designed so that the function cards could be handled as independent modules. All function cards were to be accessible without having to remove or disassemble major parts of the enclosure.

From the production point of view, the following general design objectives had to be met:

- Minimum part count
- Minimum number of parts with different stock numbers
- Minimum vendor count
- Use of preferred parts
- Compliance with all relevant medical safety standards
- Simple and automated assembly.

We also committed ourselves to design an enclosure that could be assembled and serviced with only one tool (all you need is a screwdriver).

The clinical environment mandates that the product be easily cleaned and have no sharp corners, sharp edges, or deep indentations. Liquid spilled over the Component Monitoring System is not allowed to create a hazardous situation for the patient or the user, nor may it leak into the unit. Last but not least, the constraints of the electronics had to be taken into consideration.

The requirements were sometimes contradictory. For example, on one hand, the chassis needs to have low RFI emissions, while on the other, it needs sufficient openings to dissipate as much heat as possible. The maximum internal temperature rise cannot exceed 15°C. Heat management is made more difficult by the fact that fans are not acceptable in the monitoring environment. This implies that natural convection is the main mechanism for dissipating heat.

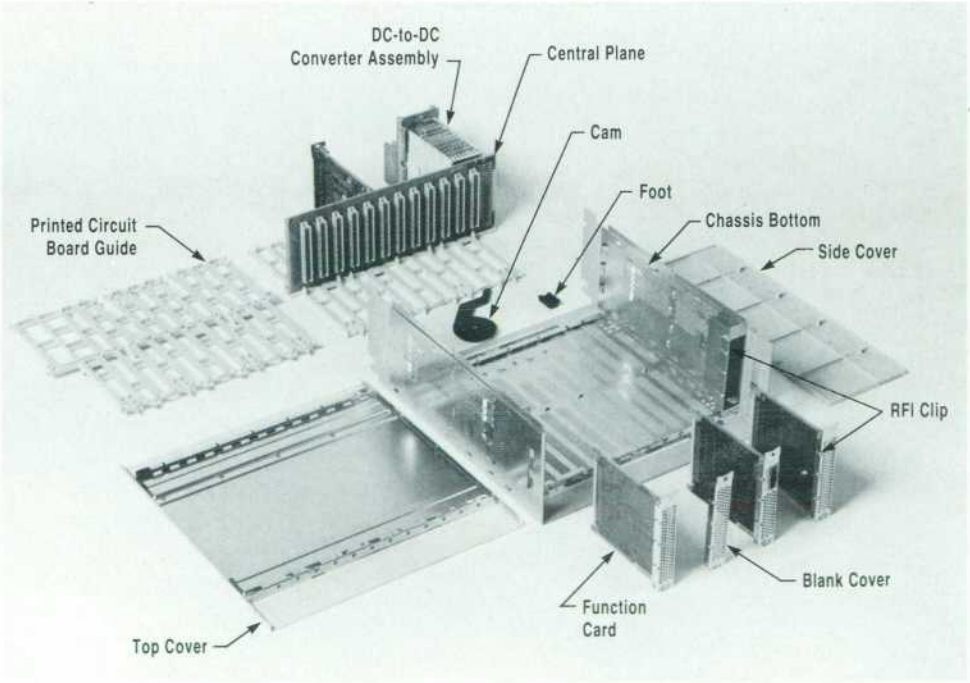


Fig. 1. Exploded view of the computer module of the HP Component Monitoring System.

Extensive measurements with simulated electronic circuits and calculations in the early project phase were a good basis for the architectural design of the computer module. The knowledge gained from these studies and the demand for easy access to the function cards led to the present design.

Product Design

Two large sheet-metal parts form the enclosure of the computer module (see Fig. 1). The bottom part of the chassis is made of 1.25-mm-thick steel and has a large number of openings for ventilation. Mounting holes and pressed-in threads for the instrument's feet and locking cam are located here. The inner part of this U-shaped component has a number of indentations and cutouts. This construction allows the plastic guide for the function cards and the central plane to be snapped in place without any screws.

The second large sheet-metal part is the top cover of the chassis. Offset bends similar to those in the bottom part of the chassis make it possible to snap the plastic function card guide into the lid. Pressed-in threads are mounted on the offset flange to hold the function cards within the computer module.

Two large indentations with strong steel strips riveted to the top cover provide a quick and easy way to mount the 14-inch display on the computer module should this be desired. A combination of indentations with an undercut, feet with noses, and the cam forms a tight locking mechanism between the display and the computer module. This technique was first used by the HP Medical Products Group in 1981 for the HP 8040A cardiotocograph. This well-established mechanical interface for stacking instruments or attaching them to wall or ceiling mounts or carts was a must requirement.

After the U-shaped bottom cover and the lid of the chassis have been assembled, all function cards can be inserted by simply sliding them into the enclosure. Metal board covers mounted on the rear ends of the function cards seal the remaining openings of the computer module. Each board cover contains openings for RFI clips and the function card's external connectors, and provides a mounting hole for fixing the function card to the frame. The remaining area is perforated for ventilation, except for the space needed to silk-screen the board name and number. Fig. 2 shows the interior of the computer module with the function cards inserted.

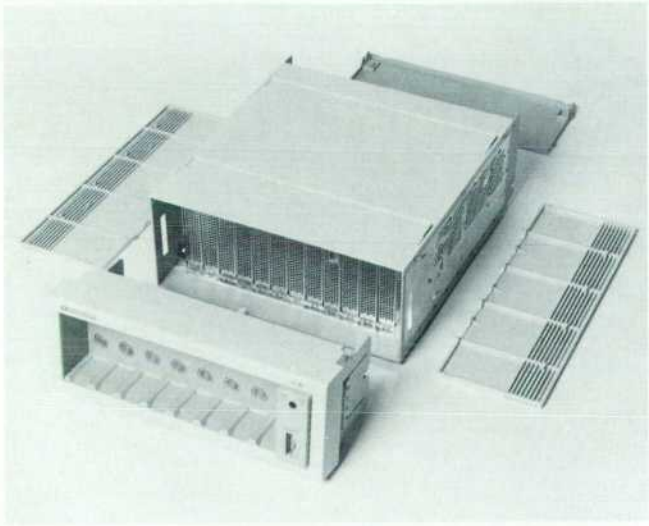


Fig. 2. Inside the partially assembled computer module.

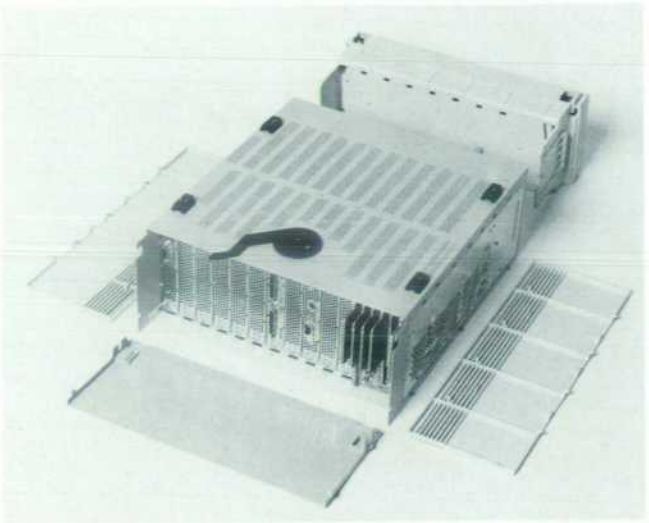


Fig. 3. Bottom view of the computer module with unlatched internal rack and side and rear covers.

As described earlier, the function cards are held in place by plastic guides in the top and bottom parts of the chassis. The printed circuit board guide is an injection-molded part that can be used in both locations by simply turning it over.

After the top cover is installed, the left and right side covers can be attached to the computer module. Again, a single injection-molded part fits both sides. This part contains all the vents and openings needed for thermal management. The side covers also conceal the six screws that attach the chassis top to the bottom. The customer can easily remove the side covers for cleaning by unlatching the internal rack (see Fig. 3).

For visual and cable management reasons, a rear cover was designed. This injection-molded part contains molded-on pivots and latching elements. Another injection-molded part with magnetic strips glued on fills the indentations on the top cover when the instrument is installed without a display on top (see Fig. 4). This completes the set of plastic parts for the computer module.

The material used for all plastic parts except the chassis feet is Bayblend®, a polycarbonate/ABS blend. The chassis feet consist of two components: a highly filled polyamid for the body and a block copolymer for the inside element. The cam is molded from polyacetate.

Display Front Assembly

The Component Monitoring System can be equipped with a choice of displays. The basic models are 14-inch monochrome and color displays. These displays consist of two major components: the bezel or front assembly, and the display consisting of the CRT, the deflection electronics,

Fig. 4. Assembled computer module with integral parameter module rack.

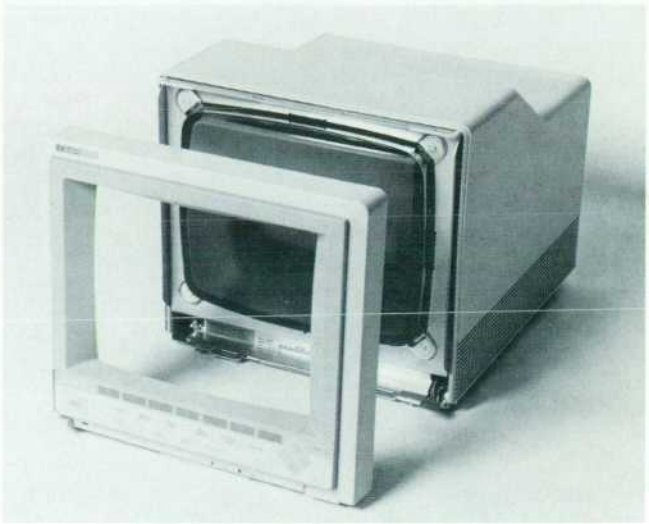


Fig. 5. The display consists of the front assembly and the display itself.

and the main power supply for the Component Monitoring System (see Fig. 5). The latter part is called the common unit.

The common unit is a purchased part. To minimize the number of options the vendor has to build and supply, this part of the display contains no language-specific elements. All options, like local language, are restricted to the front assembly only.

Objectives like design for manufacturability, clinical requirements similar to those for the computer module design, and the limitations introduced by the electronic circuits played an important role in the development of the Component Monitoring System displays.

The front assembly consists of a total of ten parts, which can be assembled simply by snapping the components in place. The main element is the plastic band (see Fig. 6). This part attaches to the common unit. It also serves as a pickup frame for the bezel, the human interface printed circuit board, a power knob, and a protection cover.

The protection cover, made of thermoformed polycarbonate, shields the human interface card from condensed water or cleaning fluids, which might drip from the CRT screen onto the printed circuit board.

The bezel is attached to the band by snap-fit connectors and presses against the rim of the CRT. Since the mono-

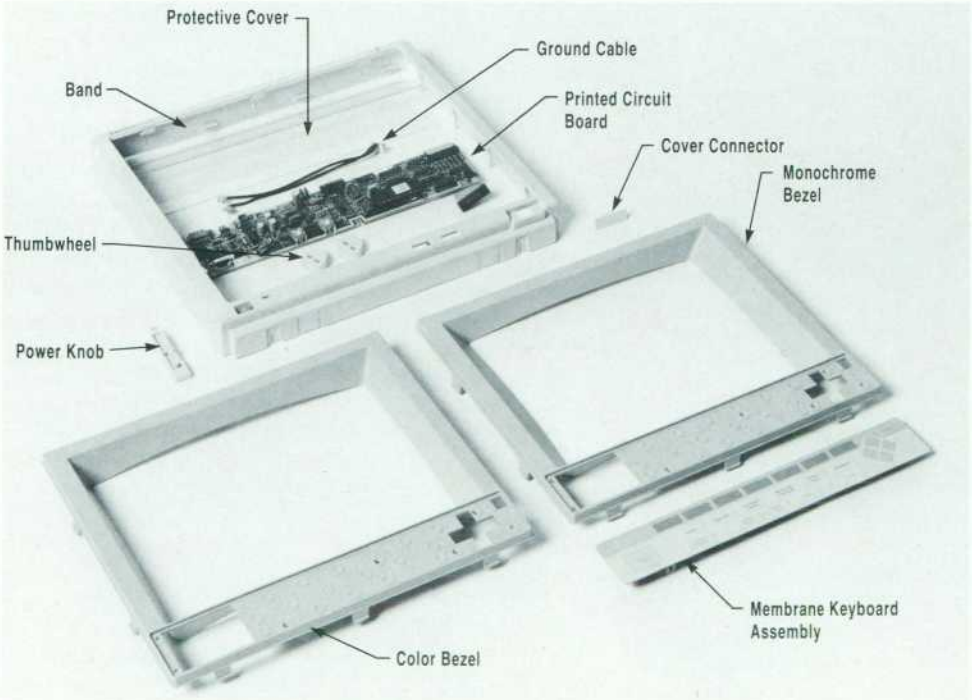


Fig. 6. Exploded view of the front assembly.

chrome and color displays do not have the same screen curvature, two bezels had to be designed. The bezel provides room for a membrane keyboard, which is the main control panel of the patient monitor. This keyboard is based on a double-sided printed circuit board. The contact elements are metal domes of different sizes. They not only make contact when pressed, but also give a good tactile feedback. The domes are covered with an embossed polycarbonate polyester overlay, which has been silk-screened from the rear to prevent abrasion of the nomenclature. Currently, the membrane keyboard is available in 11 different languages.

Design Objectives for the Parameter Modules

The parameter module mechanical design included the development of the plastic enclosure, the connectors, and the overlays and the mechanical part of the printed circuit board design. Currently two module types exist: single-width and double-width (Fig. 7). The prime objectives for the design were that it be simple to insert modules and pull them out of the rack, that the modules be rugged, and that the housing be compact, measuring only 100 mm by 100 mm by 36 mm. In addition to the general mechanical objectives listed at the beginning of this article, the parameter modules have to meet two special requirements. First, they must withstand a drop from a height of one meter onto a concrete floor. Second, for patient safety reasons, all connections to the patient are electrically floating with respect to ground. This isolation between floating and nonfloating parts is tested at 16 kV and is implemented as part of the electronic circuit in each parameter module.

From the electronic standpoint, two of the approaches to meet these objectives were to use surface mount technology for mounting the electronic components, and to apply new ways of assembling the printed circuit boards to achieve high packaging density. On the mechanical side, new ways had to be explored to build thin-walled injection-molded parts that could withstand the mechanical and thermal stresses and still be durable enough for their long hard life in the clinical environment. The entire mechanical design was done on the HP ME 10 system. Before making the final molding tools, a large number of modules were premolded using aluminium tools. The advantages were that tests could be conducted with close-to-final parts at an early stage in the project, and larger quantities could be built at a moderate cost for the extensive prototyping phase.

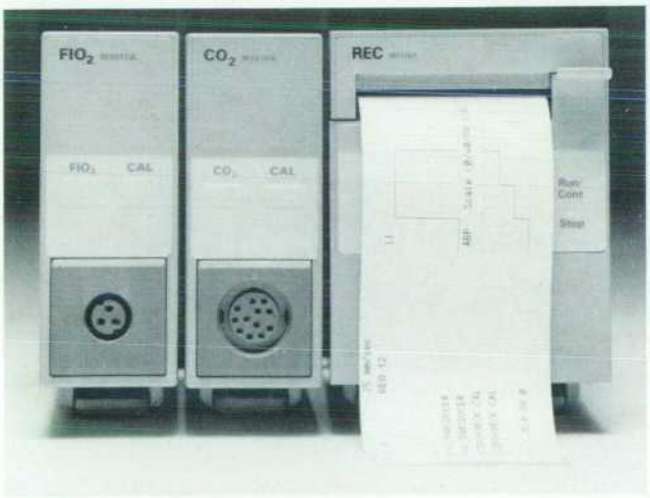


Fig. 7. Single-width and double-width parameter modules.

Parameter Module Design

The single-width parameter module consists of an assembly of seven parts (Fig. 8). These include five molded plastic parts for the enclosure, one front overlay, and one printed circuit assembly. The double-width module has two additional parts for the housing, and in the case of the noninvasive blood pressure module, a complete pump assembly, which is built in (see article, page 25).

The plastic housing of the parameter module is divided into an outer enclosure and an inner frame. This is necessary to provide the 16-kV isolation between the floating and nonfloating grounds. Between the outer housing and the inner frame there is space for shielding material such as copper, mu-metal foil, or thin-walled steel sheet. So far, only the noninvasive blood pressure module has made use of this kind of shielding.

The inner left and inner right frames are multipurpose parts for both module widths. The double-width module also has an additional middle part.

To provide maximum volume within the modules, all plastic parts have very thin walls. Nevertheless, they have to survive a one-meter drop. They also have to be resistant to cleaning agents and disinfecting solutions. We have found that Bayblend meets all these requirements. This compound includes both polycarbonate and ABS. Polycarbonate improves the ruggedness of the material while ABS has a positive effect on the chemical resistance.

The printed circuit assembly within the module consists of three boards: a digital board including the power converter, an analog board, and a board with LEDs and keyswitches mounted on it. The three boards are interconnected by flexible layers soldered onto the boards. For component loading, soldering, and test the three printed circuit boards are handled as one partially routed board with small bridges between the individual boards and an outer frame to hold all of the parts in place. Currently, we have nine parameter modules in production, which represent a total of three different routing contours. The overall size of the outer frame is identical for all parameter modules, thereby contributing to our standardization effort by making it possible to use identical pickup frames for the different assembly stages. At the last stage of the production process the three printed circuit boards are broken apart, folded like a sandwich and inserted into the plastic enclosure.

Additional mechanical parts that were designed for the parameter modules are the patient cable, the patient connector, and the module-to-rack connector. The patient cable connector had to be compatible with HP's existing monitoring equipment. One disadvantage of the existing system is the limited number of mechanical keys available. For the Component Monitoring System we therefore redesigned the connectors and extended the number of

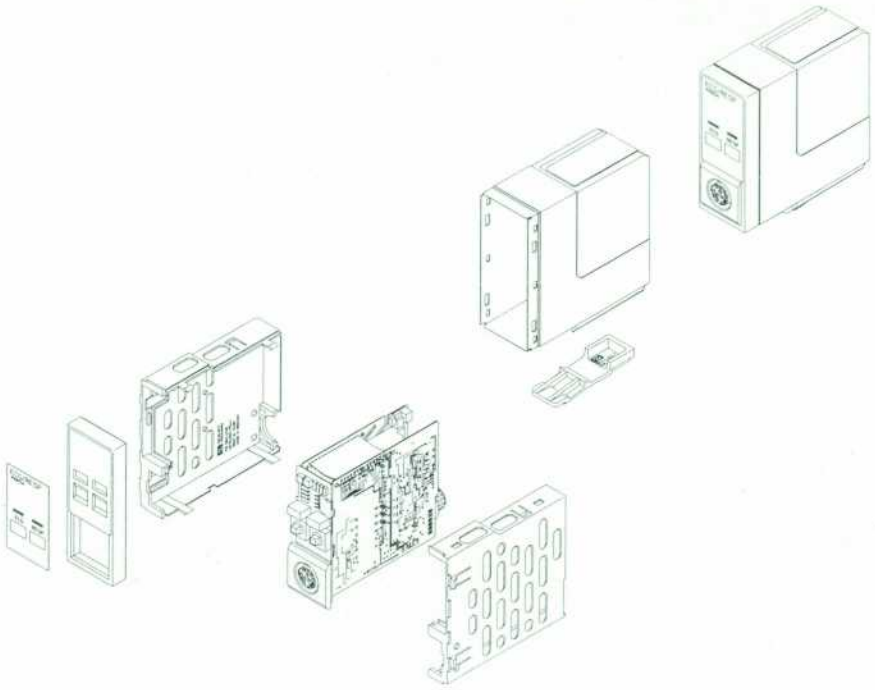


Fig. 8. Exploded view of a single-width parameter module.

possible keys so that each parameter has a dedicated configuration.

Module Assembly

Assembly time for a single module is one minute. No screws or fasteners are needed–all parts snap fit together. The normal assembly procedure includes the following steps:

- Fold the printed circuit boards together.
- Insert the printed circuit assembly into the left frame.
- Snap on the right frame (the inner module is now complete).
- Slide the inner module into the rear part of the plastic parameter housing.
- Snap the front and rear housings together
- Add the snap lock to the module.

The module is now ready and waiting for shipment. In the last production step, the proper language overlay is glued onto the front frame.

Conclusions

The mechanical design of the Component Monitoring System meets all of the design objectives. The E-score (a measure of ease of assembly) is a high 77 on a scale of 0 to 100. Only one type of screw is needed for connections where good grounding or stability is required. Part number and part count are dramatically reduced compared to former designs, and the total vendor count for all mechanical parts is less than ten.

Acknowledgments

The authors would like to acknowledge the contributions of a number of people who were involved in the mechanical implementation of the Component Monitoring System. Fritz Stepper started the design. Roland Boss continued the design and did all the optimization and integration work for easy manufacturability. Thanks also to Rainer Rometsch who did the mechanical design of the display front assembly. Werner Roessler was involved in extensive thermal design measurements and plastic part design. Special thanks also to Alfons Schmid, Herbert van Dyk, our material engineers Hartmut Würfel and Eberhard Mayer, our samplemaker Ingo Gabelmann, and Otto Schuster, who was the responsible production engineer. Last but not least, special thanks to John Murphy for his assistance.

An Automated Test Environment for a Medical Patient Monitoring System

The AUTOTEST program controls a keypusher and patient simulators to automate the testing of the software for the HP Component Monitoring System.

by Dieter Göring

The HP Component Monitoring System is a completely new patient monitor. It is based on a dedicated operating system and has an integrated data management system. It can process data from as many as 32 patient parameter modules and has an RS-232 interface for connecting a printer (see Fig. 1).

Its alarm system has a very complex structure. There are three priorities: red, yellow, and green. There are alarm messages, sounds, and lights. Alarms can be turned on or off for all parameters or only selected ones. Alarms can be sent over the serial distribution network to central stations, arrhythmia computers, or other medical devices.

Automated Test Environment

The ideal test setup for the Component Monitoring System was easy to define. We needed a Component Monitoring System patient monitor with all parameter modules installed, and we needed a human being medically connected to the monitor to (1) provide all of the patient signals such as heart rate, blood pressure, and so on, (2) change these signals to create alarm situations such as asystole or low blood pressure (it is said that Tibetan monks could do this), (3) operate the monitor like a physician or a nurse, (4) watch the monitor's display and verify correct operation (parameter numeric values, alarm messages), and (5) synchronously document all events.

Our solution to these requirements is the AUTOTEST application (see Fig. 2).

The AUTOTEST application controls programmable patient signal simulators which play the role of a critically ill patient (items 1 and 2 above). It also controls a keypusher, which can capture and execute keystrokes to operate the monitor (item 3 above). It cannot "watch" the monitor's display, but "takes a snapshot" of all important information (parameter numeric values, all alarm and inoperative messages) of the display's content whenever needed. All this information is sent over the serial distribution network every second (item 4 above).

AUTOTEST documents a test run completely into a protocol file (item 5 above). This can prove that a certain test case has been run and that the unit has passed the test. This is important, because regulatory agencies may request this data, even years after release.

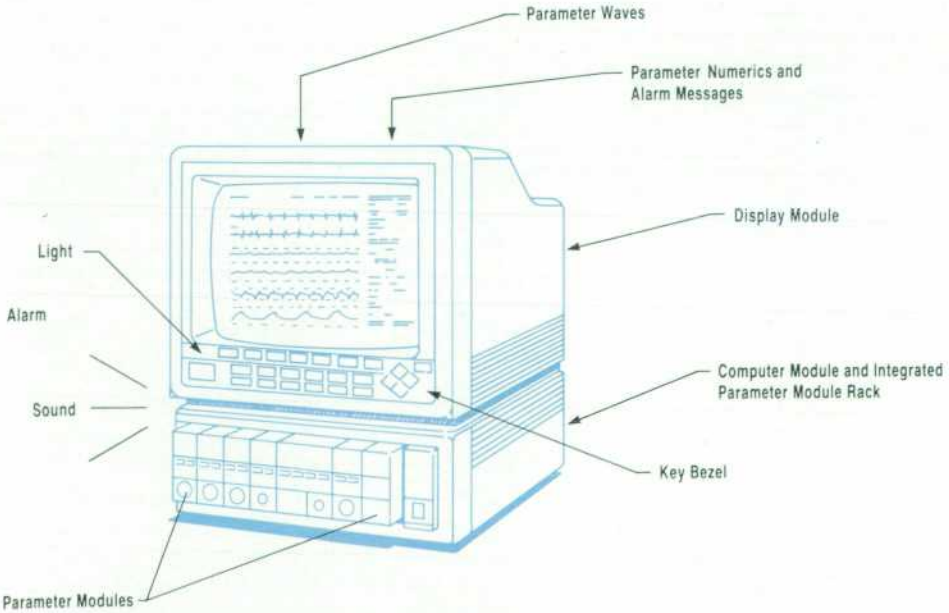


Fig. 1. HP Component Monitoring System with integrated parameter module rack.

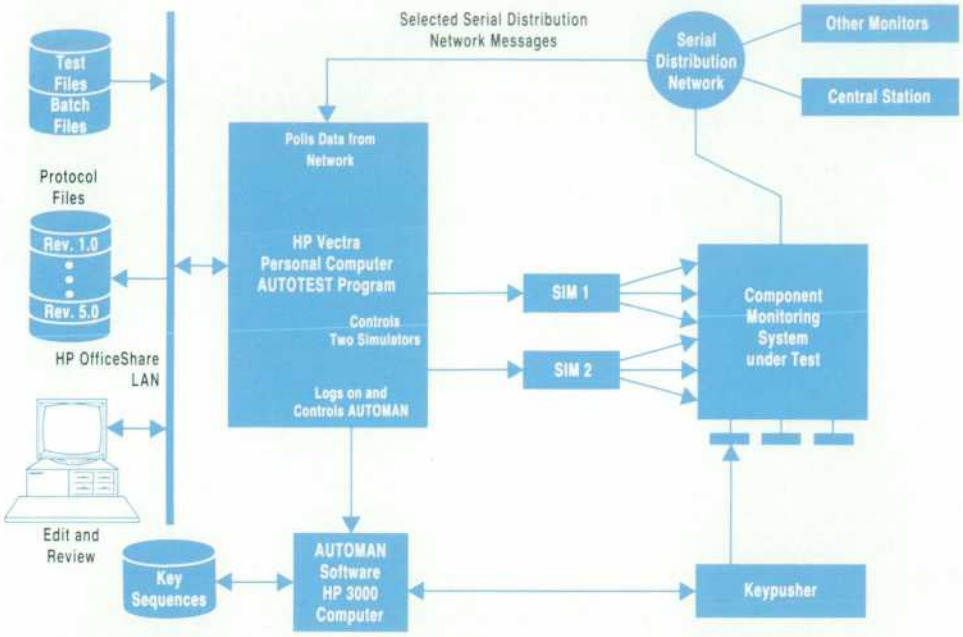


Fig. 2. The AUTOTEST setup for automated Component Monitoring System testing.

AUTOTEST Requirements

The AUTOTEST program requires an HP Vectra ES/12 personal computer with enough disk space (or better, a large disk on a LAN server), serial and parallel interfaces, additional dual serial interfaces for the simulators, and an HP 78361A serial distribution network interface. Also needed are an HP AUTOMAN keypusher and at least two Dynatech 217A programmable patient signal simulators. These simulators are widely used in medical R&D, testing, and training. Two terminal ports on an HP 3000 computer are needed, one for the Vectra PC and one for the AUTOMAN box.

The software includes the AUTOTEST program package (written in C), AUTOMAN software on the HP 3000 computer, and a smart editor on the Vectra PC for reviewing the protocol files, which can be very large.

The AUTOTEST Program

AUTOTEST is a very simple but flexible application. It reads serially through an ASCII test file and executes each line as a command line. The following are available:

- Commands to control Dynatech or other RS-232-driven simulators connected to serial ports of the Vectra PC
- A command to send a keystroke file to the AUTOMAN application running on an HP 3000 computer
- Commands to get the monitor's data and optionally all alarm messages from the serial distribution network
- A command to pause the test (lets the tester read the instructions) and wait for a comment or just a keystroke to continue
- Comment lines
- A "delay after" parameter (seconds) for every command.

All commands, all comment lines, all data polled from the serial distribution network, and all keystrokes are echoed into a protocol file. The system time of the Vectra PC is also written into the protocol file before every command line. Fig. 3 shows a test file and the corresponding protocol file.

Loops or branches are not permitted within a test file. However, for each test file it is possible to select the number of repetitions, and a batch feature allows queuing of test files in any combination.

Any ASCII editor can be used for creating and maintaining test files.

Test File Development

The test files were developed in three steps. First, we wrote high-level test scripts based on the external reference specifications, covering all of the functionality. Second, we gave these scripts to the R&D engineers for review. Third, we started development of the modular test files, beginning with the most important ones (alarms and inoperative conditions) and those that are tedious to test manually. Tests were grouped into 100% automated tests, runs with a few manual interventions, semiautomated tests, and manual tests.

The test files improved in effectiveness over time. Updates were performed constantly when new bugs were detected in the software being tested.

When R&D had finished the implementation of all of the system's functionality, the development of all of the test files was also complete. Thus, for all of the defect-fixing rounds of testing, we ran almost exactly the same tests and could show very clearly the trend of the defect rate (see Fig. 4).

Results

With AUTOTEST, a test cycle now takes only seven working days. A test cycle consists of 60 hours of automatic tests, mostly run overnight and on weekends, 45 hours of semiautomatic tests, and 5 hours of manual tests. There is also some destructive testing by selected experts, which is done in parallel with the systematic testing. Test documentation is complete when the testing is finished. The system provides a complete regression test package

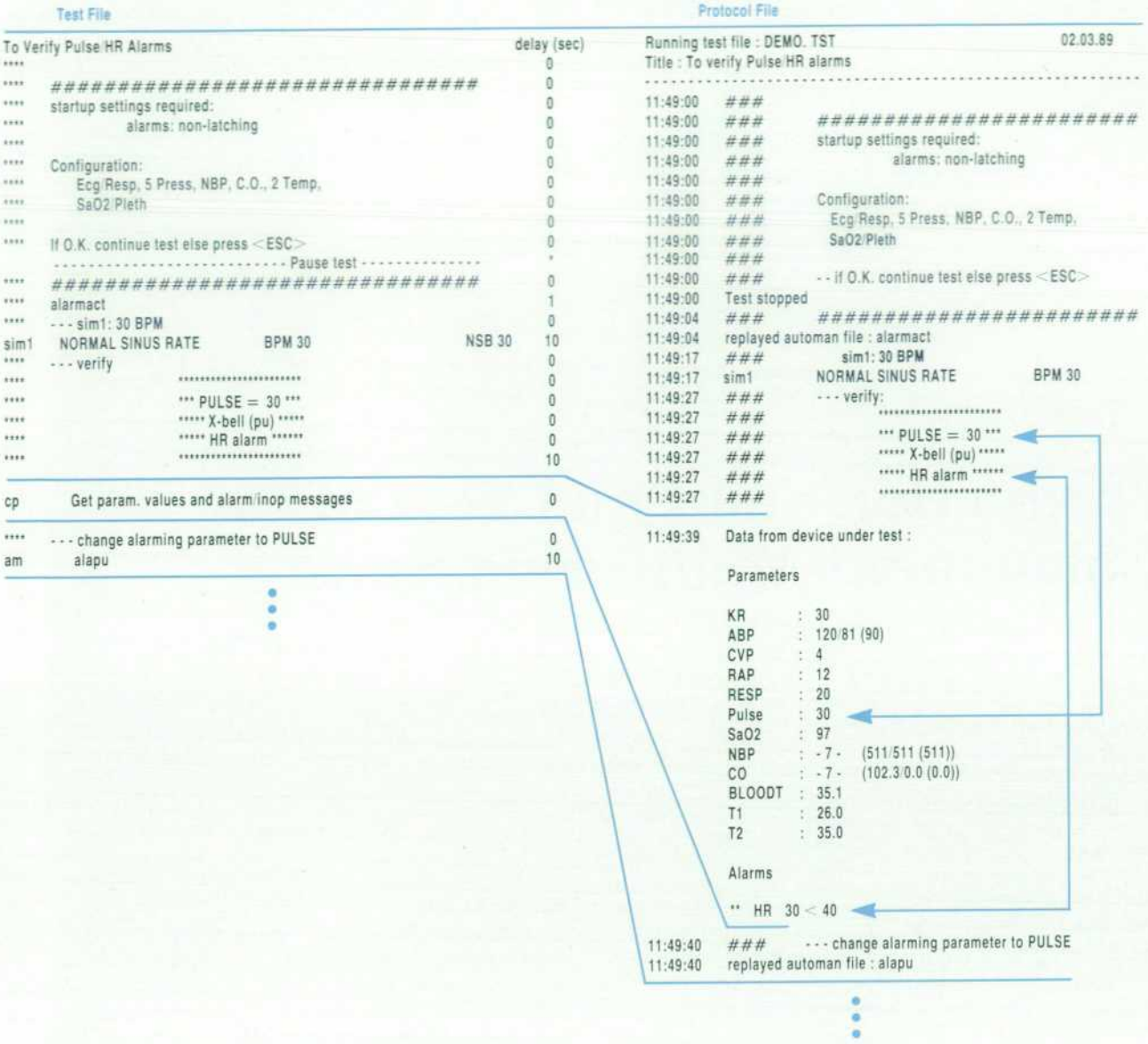


Fig. 3. An example of a test file and the resulting protocol file.

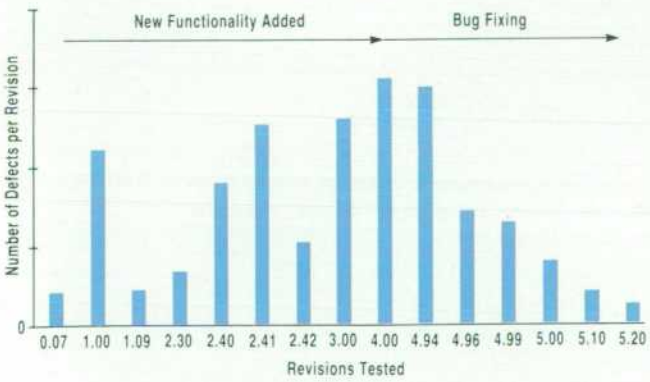


Fig. 4. Defect rate trend as a result of automated testing.

that can be used for testing revisions and can be easily adapted to testing new parameters.

We wrote one special test file that exercises the Component Monitoring System with very fast random keypushing. As long as the software was not very stable, this test file caused the system to crash frequently within a short period. With normal testing, we would have had to wait weeks to see so many failures. The R&D engineers liked this test very much because it gave them a good chance to trace the software components and find the causes of the crashes quickly.

Conclusion

AUTOTEST is an excellent example of an automated structured testing implementation. Even though it seems to be specially designed for the Component Monitoring System, it is not. With some limitations (for example, no

keypushing) it can be used for testing any HP patient monitor. It can be used simply for a form of guided testing in which AUTOTEST tells the test operator what to do and what the desired result is, and requests a keystroke P for passed or F for failed. This makes it possible to have untrained people running the tests and still get complete documentation. AUTOTEST runs on an HP Vectra personal computer and is written in C, so it is portable and can easily be modified or extended.

Production and Final Test of the HP Component Monitoring System

A vertically oriented material flow minimizes handling and simplifies customization. Automated final test systems minimize human errors and collect data for monitoring process quality.

by Otto Schuster and Joachim Weller

One of the keys to success in manufacturing a new product is the concurrent design of the product and its production processes from the very beginning of a project. Therefore, a team of experienced manufacturing engineers was integrated into the HP Component Monitoring System project and physically located in the R&D laboratory. In this way, product designs and production process designs were able to influence each other before all details had been worked out.

The plan was also to transfer the product to production concurrently in Böblingen and in Waltham, Massachusetts. Therefore, manufacturing engineers from Waltham joined our team to cover division-specific aspects and to ensure productive communication.

Another key to the product's success was the definition of manufacturing goals to which all parties were committed. The table below shows some of these goals and compares the Component Monitoring System with HP's previous generation of bedside monitors.

Total Part Number Count	–67%
Vendor Count for Mechanical Parts	–47%
Number of Printed Circuit Board Outlines	–50%
Autoloading Percentage	+27%
Manufacturing Cycle	–50%

Material Flow

To reduce material in process and to reduce manufacturing cycle time, it is essential to streamline processes without moving material back and forth. Therefore, we built up a vertically oriented material flow. Products and assemblies can be built independently up to the point where they will be assigned to a customer order (see Fig. 1). This is supported by a product structure that allows assignment to a customer order just before packaging. For example, the ECG modules are all built identically up to the last step, where the product is localized by applying the overlay label in the appropriate language.

Final Test

The objectives for the final test systems were:

- Flexibility to support different types of devices under test (DUT) without changing the test setup.
- Use of standard hardware, or design and documentation of nonstandard hardware according to HP standards for manufactured products.
- Self-test and self-calibration features wherever possible to reduce maintenance.
- Accuracy based on the specifications of standard instruments that are calibrated periodically.

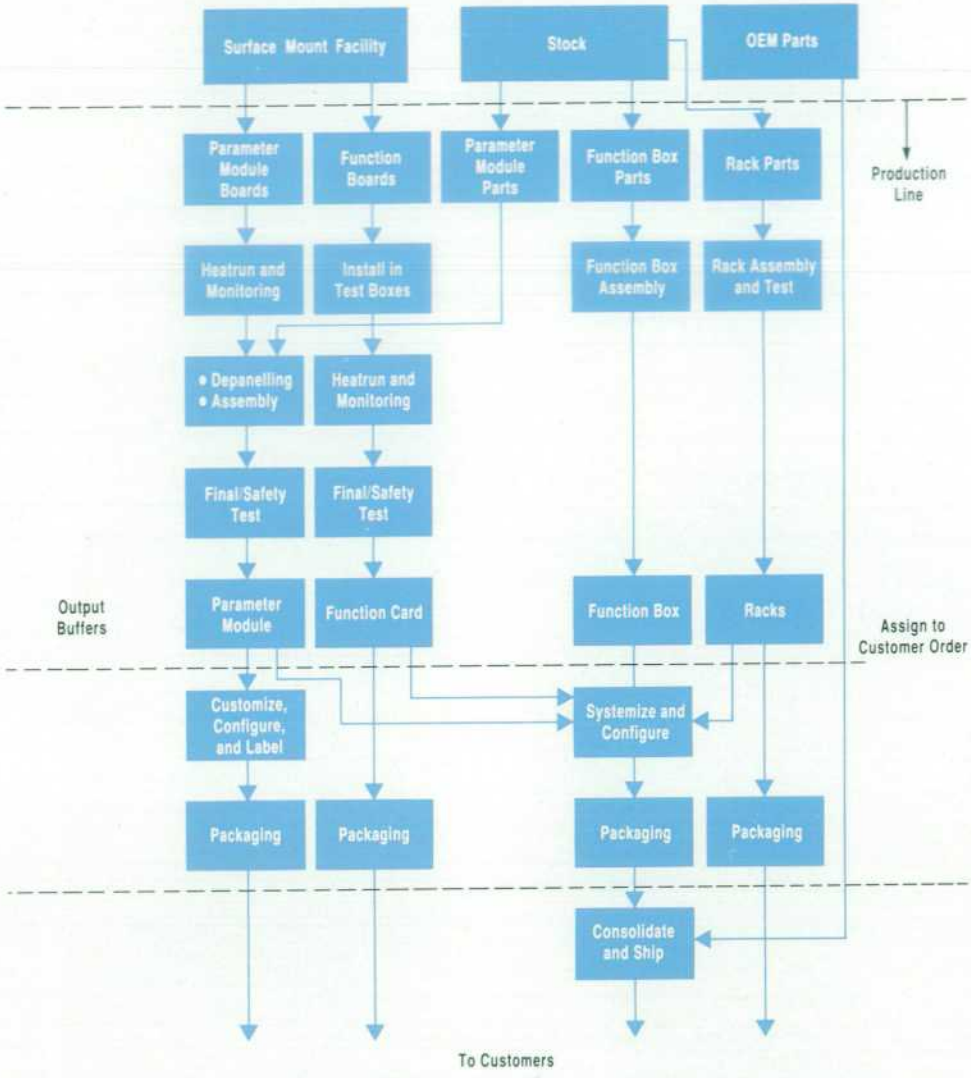


Fig. 1. HP Component Monitoring System production material flow.

- A modular test concept that allows easy addition of new parameters, that is, each type of parameter module has its individual software module for specifications, test list, test procedure, and drivers.
- Ease of learning and operation, even for relatively unskilled personnel. This is achieved by a high degree of automation, which makes it possible to operate the final test systems with very little operator interaction. In addition, color coding on the display for device types and status messages and automatic recognition of the device under test are implemented. Instead of manual adjustments, calibration data is generated by the final test station and stored in the EPROM of the DUT. To avoid human errors, test results do not have to be interpreted by the operator.
- Data accumulation for statistical process control.

Integration of Test Systems

The final test systems are based on HP 9000 Series 300 Pascal workstations, multiprogrammers, diverse HP-IB (IEEE 488) instruments, and a parameter module interface, which provides the communication link between the DUT and the workstation. All systems are connected to a shared resource manager for sharing the files required for operation and archiving files containing test results. This includes both the final test systems and the temperature cycling test systems.

An HP-UX workstation connected to the shared resource manager provides access to the files for other HP-UX terminals on a local or wide area network (see Fig. 2).

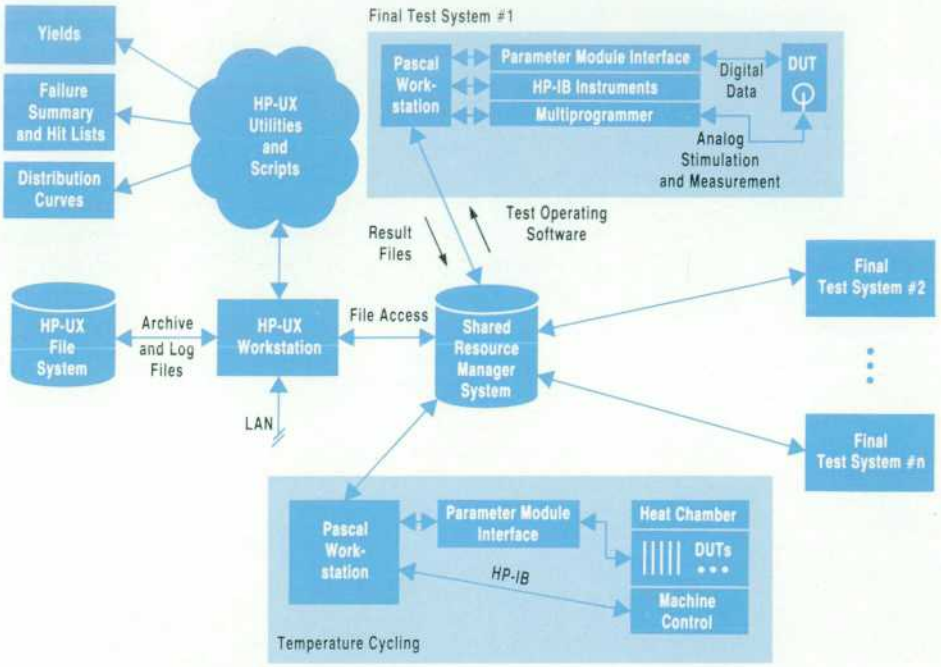


Fig. 2. Integration of the final test systems with the temperature cycling test system.

Monitoring Process Quality

To improve quality and keep it at a high level, it is important to analyze process data online. Therefore, the test results of the last 200 devices of each device type are held on disc for data analysis. Three different levels of information can be generated. The first level is the yield of printed circuit boards or modules. A diagram generated daily shows trends and can be used as a trigger for a more detailed analysis. This analysis represents the next level and includes failure summaries and failure hit lists for individual tests. The third level provides the distribution curve for test results over the test limit range along with the following statistical process parameters (see Fig. 3): mean, minimum and maximum values, standard deviation, c_p value (process capability), and c_{pk} value (process controllability).

This data is not only used for process monitoring. It has been used during Component Monitoring System prototyping to qualify each test and to verify the specification limits. Thus, early information about the producability of a new product is obtained well before the product is introduced into production and valuable feedback is provided to the designers.

Date: Fri Nov 30 16:00 MEZ 1990

Test: M1016 PowerConsumption_Test

−150 %		0.0 %	
−140 %		0.0 %	
−130 %		0.0 %	
−120 %		0.0 %	
−110 %		0.0 %	
−100 %	- MIN -	0.0 %	
−90 %		1.0 %	**
−80 %		0.0 %	
−70 %		0.5 %	*
−60 %		6.5 %	**********
−50 %		5.5 %	*********
−40 %		12.9 %	*********************
−30 %		11.4 %	*******************
−20 %		27.4 %	**
−10 %		0.0 %	
0 %	- NOM -	24.9 %	**
10 %		0.0 %	
20 %		6.5 %	**********
30 %		1.5 %	***
40 %		0.0 %	
50 %		1.5 %	***
60 %		0.0 %	
70 %		0.5 %	*
80 %		0.0 %	
90 %		0.0 %	
100 %	- MAX -	0.0 %	
110 %		0.0 %	
120 %		0.0 %	
130 %		0.0 %	
140 %		0.0 %	
150 %		0.0 %	

- evaluation test #2: PowerConsumption_Test datacount = 201 -

----Absolute----		----Relative----	
mean =	1.061343	mean =	−19 %
peakmin =	0.988000	peakmin =	−90 %
peakmax =	1.148000	peakmax =	70 %
stddev =	0.024933	stddev =	2.516335
		cp =	1.324678
		cpk =	1.071605

Fig. 3. Final test analysis.

Conclusion

The challenge for manufacturing was not only that the Component Monitoring System was a new product, but also that it was our first product designed in surface mount technology and the first product for the Böblingen surface mount technology center. The parallel ramp-up of production in Böblingen and in Waltham has proven that concurrent production process design is not just an alternative, but rather the only way to succeed.

Acknowledgments

We would like to thank all of the people who contributed to the design and implementation of the Component Monitoring System manufacturing process, in particular Günter Hasert, Wolfgang Strenzl, Frank Keil, and Dieter Frank from the Böblingen Medical Division manufacturing development engineering team and Frank Lyons, George Kinzie, and Chris Leary from the Waltham Division.

Calculating the Real Cost of Software Defects

Using data from a well-established software metrics database and an industry profit loss model, a method is developed that computes the real cost of dealing with software defects.

by William T. Ward

In response to the HP corporate-wide 10× software quality improvement initiative, much attention has been focused on improving the quality of software products developed throughout HP. The motivation for software quality improvement is most often expressed in terms of increased customer satisfaction with higher product quality, or more generally, as a need to position HP as a leader in quality software development.

A more fundamental motivation to support the initiative for higher software quality can be developed when software defect cost data is considered. The data presented in this paper is drawn from an extensive software project database maintained at the HP Waltham Division for product releases over the past five years. When software defect cost calculations are performed on this data, a very compelling "bottom line" perspective on software quality emerges; software defects are very expensive and early defect prevention and removal techniques can substantially enhance the profit realized on software products.

This paper will present a general model that can be used to calculate software defect cost data for any software or firmware product. Data from actual HP Waltham projects will be used to provide examples of software cost calculations.

The Need for Metrics

As an example of the need for substantive software quality cost data, consider the situation a project manager might encounter when attempting to justify the purchase and use of a new software development tool such as a static code analyzer. If the cost of the tool is $20,000 and if there is reliable data to suggest that the tool will uncover 5% of the total number of software defects during typical use, is the project manager justified in acquiring and using the tool?

To provide answers to this type of question, it is important to have access to a reliable database of software quality metrics. Such a database is maintained by the software quality engineering group at the clinical systems business unit of HP's Waltham Division. This database has become an essential component of software quality activities at HP Waltham and is invaluable for such tasks as project scheduling, resource planning, project and product quality status reporting, and software defect cost calculations.

In addition to maintaining the metrics database, the software quality engineering group works with R&D in testing and process improvement activities.

Software Quality Metrics Database

Fig. 1 indicates the development phases of a typical software project, with the phases indicated in which metrics are collected and stored into the software quality database. Data is gathered from a variety of sources including software defect logging, product comparison studies, proj-

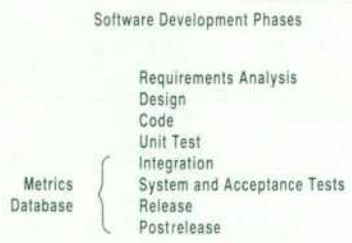


Fig. 1. Software life cycle and the phases the software quality engineering metrics database covers.

Fig. 2. Software defect find and fix cycle.

ect post-mortem studies, code complexity and size analysis, and project schedule and resource plans. The physical data resides mainly in a standard HP STARS* database, which has been augmented with additional fields, files, and reporting utilities. All of the products represented in the metrics database are firmware-based medical devices such as critical care monitors, arrhythmia analysis computers, and clinical databases.

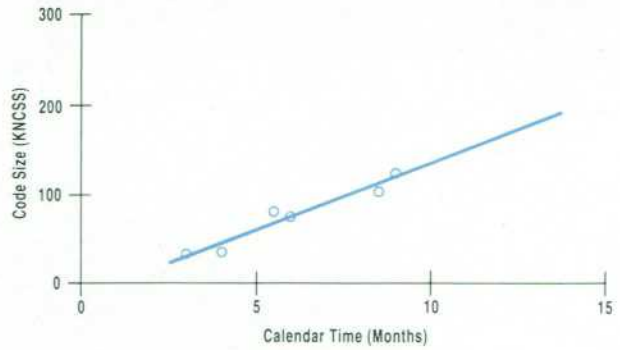
Figs, 2, 3, and 4 represent various types of useful data that can be extracted from the database. Fig. 2 documents the steps that are typically required to find, fix, and retest a defect discovered by the software quality engineering group during integration and system or acceptance testing. The engineering effort for this activity, which is shown as 20 hours, represents the average effort for finding and fixing one typical software defect. This value has been calculated using hundreds of data points from multiple software projects that have been tracked with the software quality database. Fig. 3 is an example of how an accurate schedule for the integration through the release phases can be developed using historical project data from the database. In this case, it is clear that a stable and linear relationship exists between product code size and resultant calendar time. Finally, Fig. 4 tabulates various software metrics from multiple software projects. This data can be very useful for developing project comparison studies.

The data presented in these figures is a small subset of the data that exists in the database. This specific data has been presented because of its applicability to software defect cost calculations.

Looking for Software Defect Costs

Software defect costs can be investigated using a variety of different approaches. For example, costs can be calculated on a prerelease or a postrelease basis, or costs can be determined per defect or per project phase, or costs can be weighted based on code size or programmer productivity. The software defect cost data developed in this paper focuses on the cost per prerelease software defect that is found and fixed during the integration through the release phases of project development. This approach is used because of the abundance of reliable data points available for study and because of the potential utility of the results.

*The Software Tracking and Reporting System, or STARS, is an HP internal database for tracking software defects.



KNCSS = Thousands of Noncomment Source Statements

Fig. 3. Calendar time for integration through release phases versus code size for HP Waltham clinical systems business unit projects. Each point represents a specific project in the database.

	Project A	Project B	Project C	Project D	Project E	Project F
Code Size (KNCSS)	125	77	270	45	105	35
Calendar Months (Integration Through Release)	9	6	12	4.5	8.5	3
Number of Defects Requiring Fix	269	133	443	110	210	137
Prerelease Defect Density	2.15	1.72	1.6	2.5	2.0	4

Fig. 4. Typical software metrics for projects in the software quality database.

The Software Defect Cost Equation

The calculation of prerelease software defect cost proposed here is based on the formula:

$$\text{Software Defect Cost} = \text{Software Defect Rework Cost} + \text{Profit Loss}$$

Software defect rework cost is determined by the amount of effort and expense required to find and fix software defects during the integration through release phases of a software project. Profit loss is the revenue loss that is caused by lower product sales throughout the entire postrelease lifetime of the product. The lower sales factor is caused directly by the lengthy find and fix cycle of prerelease defects that force a schedule slip and result in a loss of market-window opportunity.

Many other factors could probably be used to determine the software defect cost but our data shows that the rework cost and profit loss factors have a major impact on the result and will supply a close first approximation of the final value. Table I lists a set of product and project software factors that will be used to calculate a software defect cost value. All of these factors represent typical values derived from our database.

Table I
Typical Values in the Metrics Database

Code size	75 KNCSS
Calendar time for prerelease testing	6 months
Number of prerelease defects found and fixed	110 defects
Prerelease defect density	1.5 defects/KNCSS

Software Defect Rework Calculation

This calculation is very simple and is based on data presented in Figs. 2 and 4 and Table I. A typical product will have 110 software defects found and fixed during the project test phase. Each of these defects will require 20

engineering hours to find and fix. The total prerelease software rework effort then is:

Software Defect Rework Effort = 110 × 20 = 2200 engineering hours.

To convert this effort value to dollars requires the $/hour software engineer factor. As a close approximation of an industry standard value, we will use $75/hour as the standard charge for the services of a software engineer. (This includes basic salary + administration overhead of 75%).

Software Defect Rework Cost = 2200 hours × $75/hour = $165,000.

On a per-defect basis, rework cost can be determined as:

Rework Cost per Software Defect = 20 hours × $75/hour = $1500.

These calculations are useful in highlighting the true waste factor of poor software quality. Each software defect is responsible for $1500 of unnecessary expense, and for a typical project $165,000 is required for software rework.

Software Defect Profit Loss Calculation

The other major factor contributing to software defect cost is product profit loss because of missed market-window opportunities and the resultant loss of product sales. In other words, if a product release date slips because the software defect find and fix cycle is unnecessarily long, then potential product sales are irretrievably lost and overall lifetime profit dollars will be less. Such factors as rapidly obsolete technology and the availability of competitive products also contribute to the potential loss of sales.

Several industry models [1,2] have been proposed that can be used to quantify the profit loss factor. Fig. 5 presents one of these models and will serve as the basis for our calculations. For the following calculations we assume a 1000-unit customer base of a $20,000 product with a 15% profit margin. This will yield $3,000,000 in lifetime profit. Assuming a six-month slip in product release because of

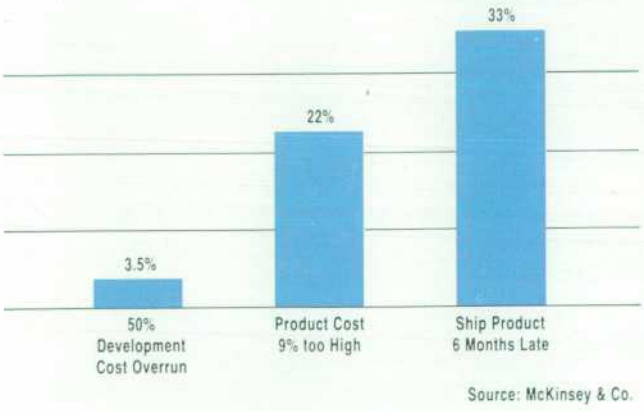


Fig. 5. Percentage of profit loss associated with product release problems. The type of products this data represents have a short product life of around five years. Examples include word processors and other consumer electronic products.

Code Size: 75 KNCSS	
Software Test Phase: 6 Months	
Number of Prerelease Defects: 110	
Software Rework "Waste": $165,000	($1500 per defect)
Profit Loss Due to Rework: $1,000,000	(Approximately $9000 per defect)
*Total Cost of Software Defects: $1,165,000	($10,500 per defect)

*Software Rework Cost + Profit Loss

Fig. 6. Software defect cost summary for a typical software project.

the software defect find and fix cycle, Fig. 5 suggests a 33% loss in profit.

Profit Loss = $3,000,000 × 33% = $1,000,000

Using the data on the number of prerelease defects given in Table I, on a per-defect basis, profit-loss can be determined as:

$1,000,000/110 defects ≈ $9000 per defect.

It may seem extreme to say that every prerelease defect causes a product to be late to market. However, because of the nature of our business, it is important that our products perform reliably in the critical-care medical environment. This means that each defect of a high enough severity level that is found during prerelease tests must be fixed and retested before final release. It is this test, fix, and retest cycle that delays product release and contributes to the cost of poor software quality.

The $1,000,000 Opportunity

Fig. 6 summarizes the software defect cost data calculated in this paper. The variables used in these calculations will vary from one organization to another, but the fundamental algorithm for computing software defect cost is applicable to most cases. Although the product cost and profit margin numbers used here are for illustrative purposes, they are typical for large software systems. Therefore, with the potential for a cost of $10,500 per defect and $1,165,000 per project, there is ample financial basis for a number of potential remedial actions.

Quality Awareness. Most software engineers probably have no idea about the cost of reworking software to find and fix a defect once the code enters the integration and test phases. They should be made aware of the savings possible if more defect detection could be done in the early stages of product development.

CASE Investment. There are a large number of CASE tools and methodologies available to augment the software development process. Examples of modern CASE technology include static code analyzers, debuggers, execution profilers, formal inspections of design and code, structured analysis and design, and so on. Most of these technologies can be acquired for a financial investment of $10,000 to $30,000. If each software defect has a $10,500

cost, then it is clearly appropriate to consider the use of CASE to improve software quality.

Software 10× Program: When it becomes clear that software quality improvements can yield substantial financial rewards, then the goal of a 10× gain in software quality assumes additional impetus. Consider that a 10× improvement of the number of prerelease software defects for the typical software project presented in this paper would yield almost an additional $1,000,000 in profit. That figure is a powerful bottom line motivator.

Conclusion

This paper has presented a technique that can be used to calculate software defect cost values. Historical HP Waltham software quality and project data has been applied to cost calculations so that realistic results might be obtained. Although additional investigations, such as a determination of postrelease software defect cost, might provide a more detailed analysis of cost, the data presented in this paper is accurate and provides compelling financial motivation for improved software quality.

References

1. B.C. Cole, "Getting To The Market On Time." *Electronics*, Vol. 62, no. 4, April 1989, pp. 62–67.
2. D. G. Reinertsen, "Whodunit? The Search for the New-product Killers." *McKinsey and Company Report*, 1983, pp. 35-37.

A Case Study of Code Inspections

The code inspection process is a tool that can be used early in the software development cycle to help improve the quality of software products and the productivity of development engineers.

by Frank W. Blakely and Mark E. Boles

Code inspections have become an integral part of the software development life cycle in many organizations. Because it takes some project time and because engineers initially feel intimidated by the process, code inspections have not always been readily accepted. Additionally, there has not always been enough evidence (metrics) to prove that for the time and effort invested, the process has any value in reducing defects and improving overall software quality. Since the early days, the process has become better understood and documented, and recent articles have provided concrete metrics and other evidence to justify the value of the process.[1,2,3]

This paper describes our experiences in bringing the code inspection process to HP's Application Support Division (ASD). We describe both the positive and negative findings related to using code inspections. Although we only have metrics for one project, our main goal here is to present how we implemented the inspection process and to illustrate the type of data to collect and what might be done with the data.

Background

In 1988 our division was in the process of searching for best practices and methodologies that could help us meet or exceed the company-wide 10×quality improvement goal. Design and code inspections were two of the methodologies that we proposed. Management agreed to sanction a pilot project using code inspections, with implementation of a design inspection process deferred to a later date.

The authors attended the software inspections class given by HP's Corporate Engineering software engineering training group. This knowledge was then imported to our division, and several classes were taught to the engineers to prepare them to be participants in the code inspection process.

To begin using the code inspection process on a pilot basis, a software project that involved enhancing an existing product was selected. To ensure that we could improve the process and learn from our experience, we decided to record and analyze the results from each code inspection. This way we would have some data to back up any claims we had regarding the value of the process. The metrics and data we decided to collect and analyze included:

- The criteria to use in selecting modules to inspect.
- The criteria for selecting participants in the process.
- The methodology used while performing the inspection.
- The relative effectiveness of code inspections in improving quality versus standard testing via module execution.
- The merits of doing code inspections before or after bench testing a software module.
- The number of defects found in a product in the first year after release that might have been found in code inspections.

- The costs of addressing defects found in the first year after release during the inspection phase versus during the maintenance phase.

Engineers from the quality and productivity department of our division taught the inspections class before starting the pilot project. They also acted as moderators for the formal code inspections. R&D software engineers from several different projects participated as authors, readers, and inspectors. There was also a technical marketing engineer who participated as an inspector.

Implementation of the Code Inspection Process

Developing a set of criteria for which modules to inspect, who should participate in the inspections, and what methodology to follow were the first issues that needed to be resolved in implementing the inspection process. Two methodologies were used in our implementation: formal and informal inspections. The pilot project used the formal, more structured process presented in the inspections class, while later projects used both formal and informal methodologies. The comparative success of the two methodologies has not been fully determined, but some of the results are described later.

Module Selection. Because a software product is made of many modules, the time it would take to inspect every module might be prohibitive. Therefore, criteria must exist to determine which modules should be inspected. Two criteria were used to identify the modules to be inspected. First, modules were selected only if they were modified in the course of the project (remember this was an enhancement project). Second, the complexity of the modified modules was determined. Module complexity has been shown to be a good indicator of the defect proneness of a software module—the higher the complexity, the greater the likelihood there are defects. Therefore, complexity was used to identify those modules that were the best candidates for inspection. The original plan was to determine the complexity of modules using the McCabe complexity tool,[4] which is based on the McCabe complexity metric.[5] Unfortunately the McCabe tool could not accurately identify the complexity of the modules written in MS-DOS macro assembler language. This led to attempts to roughly quantify the complexity of these modules, relying on the opinion of the module's author as to its complexity. The result was that modules were inspected based on the amount of modification to the module and a rough estimate of their complexity.

Participant Selection. A code inspection is a structured process in which each participant has a clearly defined role. Inspection participants were selected based on their knowledge of the language in which the module was written. Attempts were also made to select engineers who were also knowledgeable about the high-level structure of the product. We could not find many engineers with this knowledge so this criterion was abandoned and knowledge of the development language became the predominate criterion.

Formal Code Inspections. The methodology we used for formal code inspections required that the author select the module to be inspected and prepare inspection packets for the moderator, code reader, and inspectors.* The code inspection packet was distributed two days before the scheduled inspection date. The packet consisted of a cover sheet and listings (with line numbers) of the module to be inspected. The cover sheet detailed the time and place of the inspection meeting, the participants, and a brief description of the module to be inspected. Because some of the engineers who participated as inspectors and readers were unfamiliar with the design of the modules, additional information was sometimes added to the packets to improve the effectiveness of the process. Additions included module descriptions, pseudo code, and design overview documents describing the design of the module.

Before the meeting the moderator was responsible for confirming the availability of the meeting participants for the time and date of the inspection. Inspectors were required to prepare for the inspection, and if they could not guarantee adequate preparation, notify the moderator so that the inspection could be rescheduled. Usually on the the day before the meeting, the moderator would check to make sure that everyone was ready.

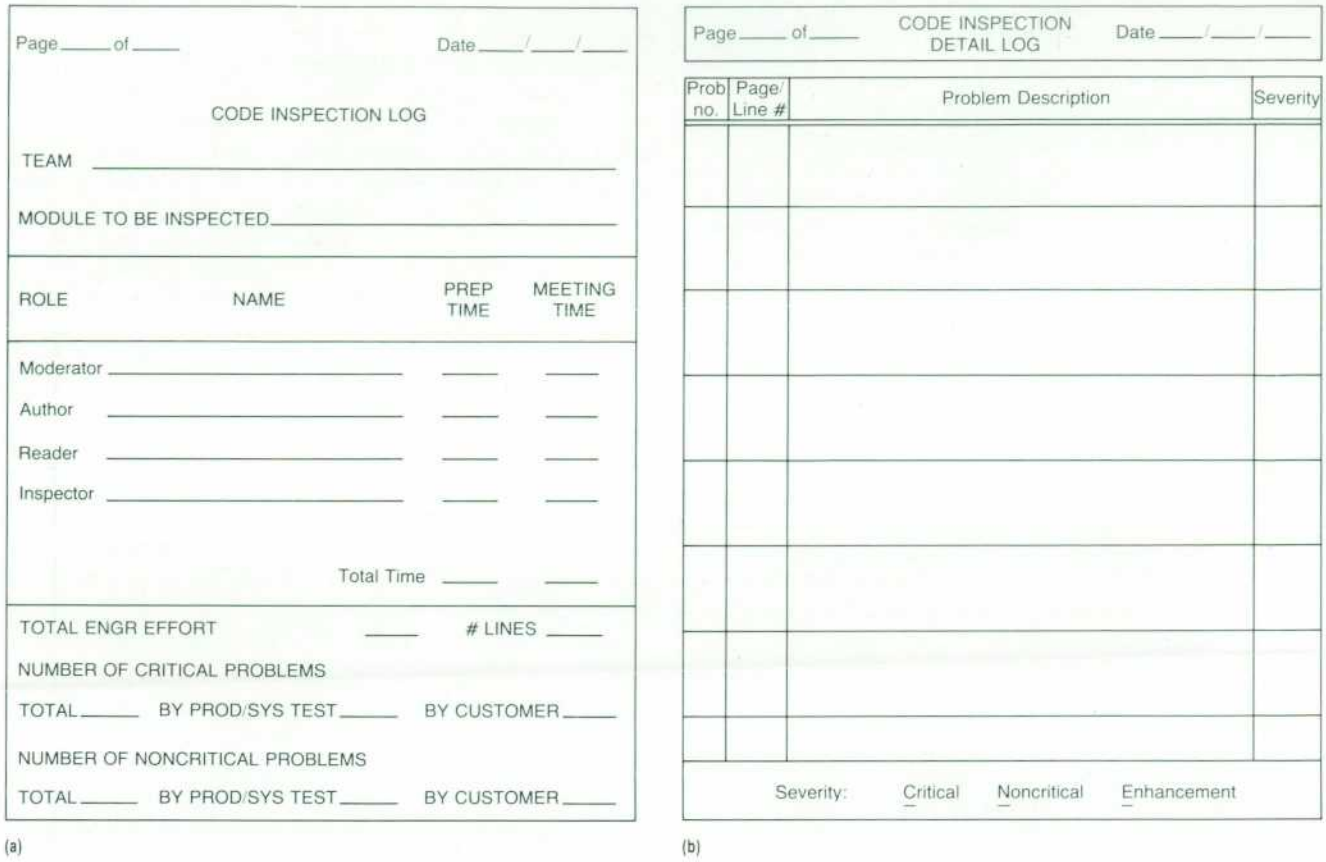
During the meeting the following rules were followed to avoid conflicts and ensure a productive process.

- Critiques of the coding style used in the module being inspected were avoided.
- Problems were to be indicated and identified, but solutions were not to be offered during the inspection meetings.
- Comments were to be phrased in a nonthreatening way, focusing on what the module did, as opposed to what the author might have done.
- Antagonistic ways of expressing points of view were avoided.

The meetings were limited to one hour. Data from other divisions showed that longer meetings drained the inspection team and decreased their effectiveness.

Two forms were used to gather the data from the code inspections: the code inspection log and the code inspection detail log (see Fig. 1). The code inspection log described the module inspected, the participants, their preparation time, the hours of engineering effort involved, the number of lines inspected, and the number of problems identified. The defects identified were categorized by severity, and whether and how they could have been found in the absence of code inspections. The code inspection detail log identified a problem by page and line number in the module source, and provided a description of the problem and its severity. At the end of the meeting a decision was made as to the advisability of scheduling a reinspection of the module if the number of problems found was unacceptable. If the author made extensive changes to the inspected modules while fixing inspection-identified defects, this would possibly create a need for reinspection.

*We actually had two inspectors. The author acted as an inspector.



Page ____ of ____ Date ____/____/____

CODE INSPECTION LOG

TEAM _____

MODULE TO BE INSPECTED _____

ROLE	NAME	PREP TIME	MEETING TIME
Moderator			
Author			
Reader			
Inspector			
	Total Time		

TOTAL ENGR EFFORT ____ # LINES ____

NUMBER OF CRITICAL PROBLEMS

TOTAL ____ BY PROD/SYS TEST ____ BY CUSTOMER ____

NUMBER OF NONCRITICAL PROBLEMS

TOTAL ____ BY PROD/SYS TEST ____ BY CUSTOMER ____

(a)

Page ____ of ____ CODE INSPECTION DETAIL LOG Date ____/____/____

Prob no.	Page/ Line #	Problem Description	Severity

Severity: Critical Noncritical Enhancement

(b)

Fig. 1. (a) Code inspection log. (b) Code inspection detail log.

In the postinspection meetings, the moderator and the author met so that the moderator could give the collected information to the author. The author was then responsible for implementing the defect repairs. If enhancements had been identified in the inspection, these were either accepted or rejected by the author and the moderator.

Informal Code Inspections. Informal code inspections were not a part of the pilot project. Several informal code inspections were performed on later projects and as a part of the pilot project's CPE (current product engineering) or postrelease activities. There was no preparation required for these informal code inspections. The code's author would request that another engineer sit and look over a module or a submodule at the time of the meeting. There was no moderator, and no formal documentation of the process. Usually, the engineer who was asked to inspect was familiar with the overall design of the module being inspected.

Measure	Data
Pilot Project	
Lines of NCSS Code Inspected	1516
Engineering Hours of Preparation	55.9
Engineering Hours in Meetings	30.4
Number of Defects Found	21
Number of Enhancement Requests	4
Follow-On Projects (Formal)	
Lines of NCSS Code Inspected	196
Engineering Hours of Preparation	4.5
Engineering Hours in Meetings	4.0
Number of Defects Found	1
Number of Enhancement Requests	0
Follow-On Projects (Informal)	
Lines of NCSS Code Inspected	1154
Engineering Hours of Preparation	0
Engineering Hours in Meetings	5.3
Number of Defects Found	1
Number of Enhancement Requests	0

Fig. 2. Summary of code inspection statistics.

Data Collected

As mentioned earlier we tried to collect data that would enable us to evaluate the effectiveness of the code inspection process. The code inspection data collected included the number of:

- Lines of noncomment source statements (NCSS) inspected
- Engineering hours of preparation required
- Engineering hours spent in code inspection meetings
- Defects and enhancement requests found during the meeting.

Because the effort required to fix a defect is the same regardless of how it is found, no data was collected on the time taken to fix defects or implement enhancements. Also, because the implementation of the informal code inspection process was done in a way that permitted a wide variation in the statistics collected, insufficient data exists to determine clearly the effectiveness of informal code inspections. Fig. 2 summarizes the statistics from our pilot project and some follow-on projects in which

Component	Lines Inspected	NCSS	Percent Inspected
Pilot Project			
Server	961	7890	12.2%
PC	555	22346	2.5%
Both	1516	30235	5.0%
Follow-On Projects (Formal)			
PC	196	22346	1.0%
Follow-On Projects (Informal)			
Server	155	7890	2.0%
PC	999	22346	4.5%
Both	1154	30236	3.8%

NCSS = Noncomment Source Statements

Fig. 3. Major modules and percent of code inspected.

both formal and informal code inspection processes were used. Fig. 3 shows the major modules and the amount of code inspected from each module.

From the data shown in Fig. 2, the ratio of preparation time to inspection time can be calculated as 1.83 for the pilot project, and 1.13 for the formal inspections done for a follow-on project. The ratio for all of the projects was 1.76. The inspection rate for the pilot project was 200 lines of NCSS per meeting hour.* For the follow-on project it was 196 lines, bringing the overall rate to 199 lines per meeting hour. The defect finding rate was 0.243 defects per engineering hour (preparation hours plus meeting hours) for the pilot project and 0.118 for the follow-on project, bringing the overall defect finding rate to 0.233. Work done by other HP divisions showed that for a one-hour meeting, 200 lines of code is about the most that can be covered before the defect finding rate begins to decline.

The metrics for preparation time and lines per meeting can be used to estimate for future projects the amount of time required for inspections so that projects can budget in their schedules a reasonable amount of time for inspections. The defect finding rate can be used to correlate between the effectiveness of different types of testing and the complexity of modules so that the overall validation of the code design can be optimized.

During the development phase of the project, test log sheets were used to collect data about defects found during testing and the number of hours devoted to testing. This data was used to help analyze traditional testing versus code inspections.

Defects reported by customers against the pilot product during the first year after its release were also collected. The objectives of gathering this data were:

- To determine whether the defects were in modules that had been inspected
- To determine why, if the defects were in inspected modules, they were not found during the inspection
- To determine why modules with defects were not inspected
- To determine the relative costs of addressing the defects found by customers compared with the cost of putting more effort into performing code inspections.

The data collected also included the number of hours of online and offline support spent identifying and verifying the problems, the number of engineering hours spent identifying the causes, the appropriate fix for the defect, and whether the modules in which the defects were found were inspected. Additional information allowed estimation of the time required to perform code inspections on those modules not inspected.

Comparison to Testing Process

For comparison with the traditional testing process (i.e., module execution), defects for the pilot project were categorized according to the method used to find the defect and the defect cause. Fig. 4 shows this categorization for the pilot-project defects. Note the inclusion of customer-found defects. Fig. 5 shows the severity of defects found classified by severity and test type. Note that inspections found defects in each of the standard severity categories.

Figs. 4 and 5 do not reflect comment defects or defects resulting from incorrect design. Both types of defects were found and noted but not counted.

If Inspections Had not Been Done

The defects found by code inspections were analyzed to determine whether they might have been found if code inspections had not been done. While these classifications are not certain, they were our best determination based on knowledge of the product, the test environment, and the way in which the product would be used. Fig. 6 shows where in the process we think certain defects would have been found if they had not been caught by code inspections.

As a further aid in analyzing the effectiveness of code inspections as a verification tool, the defects found during

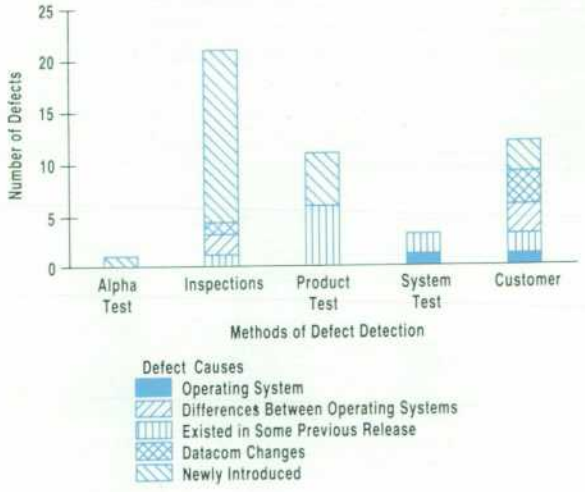


Fig. 4. Defects categorized by cause and methods used to detect the defects.

*For each formal inspection, there were 4 participants and a total of 30.4 hours were spent in meetings. Thus, 30.4 engineering hours in meetings equals 7.6 meeting hours, resulting in 200 lines per meeting hour (1516 lines/7.6 meeting hours).

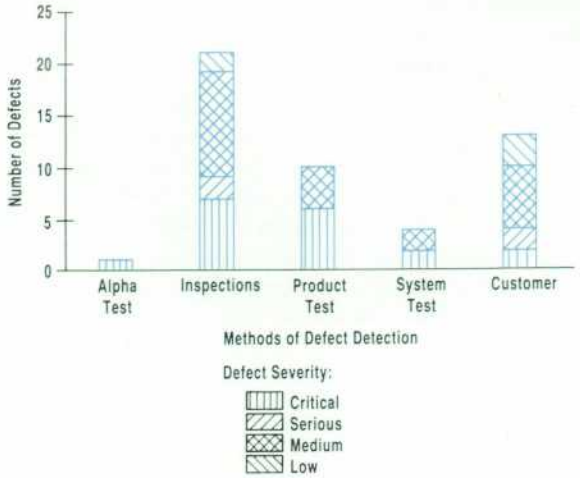


Fig. 5. Defects categorized by severity and method of detection.

inspections were divided into categories based on the following questions about each defect.

- Would the defect have been found by the existing test process?
- Could a test be devised to find those defects that would not be found by existing tests?
- Would customers find those defects that would not be found by existing tests?
- Would some of these defects not be found by any of the above methods?

Of the 21 defects found by code inspections, only four could have been found with new tests.

Defects Found by Customers

An analysis was made of the defects found by customers in the time since the pilot product was released. This allowed us to attempt to determine why the defects were not identified and fixed before the release of the product. Fig. 7 shows the distribution of defects found by customers and the reasons associated with the defects. These reasons include:

- The defect was a global error, not specific to a module or modules.
- The defect existed in a dependent product.
- The module was not inspected because it was not modified. This also indicated that our test suites were not thorough enough.
- The module was not inspected because modifications were not considered significant.
- The module was inspected, but the defect was not identified.

The defects classified in the first two categories were those for which a code inspection could not have identified the defect.

The cost of having customers find defects in the product compared to the cost of using code inspections to find these defects was measured by collecting cost information from the response center,* support engineering, R&D, and test and manufacturing. The average time of a response center call was determined and used to estimate the response center cost of handling a defect. The cost of the time spent by the support engineer was determined based on the time spent in responding to calls about defects in the product. The cost of the time spent by development engineers was based on the time spent identifying the cause of the defect, the time to fix it if necessary, and the time required to test the defect once it was fixed. Estimates of the time required to perform system testing and the manufacturing costs were also collected. The results showed that it costs approximately 100 times more to fix a defect in a released product than it costs to fix a defect during the code inspection phase.

One important cost that cannot be directly measured in currency is the loss of customer satisfaction when the customer finds a defect. Because this cost is hard to quantify, it is sometimes ignored, but the fact is that it does affect the profitability of products.

*The response center is the primary contact for customers and HP field personnel to obtain help with HP software.

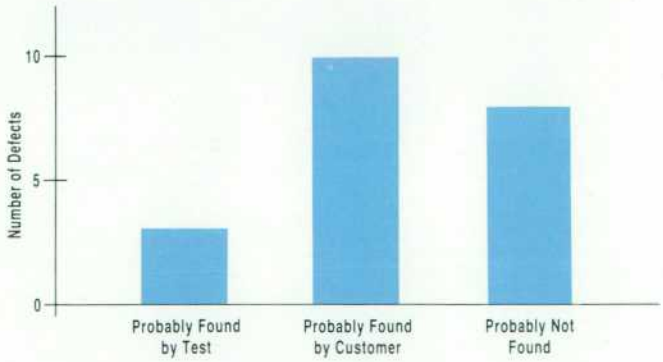


Fig. 6. Distribution of where code-inspection defects may or may not have been found if they had not been found during the code inspection process.

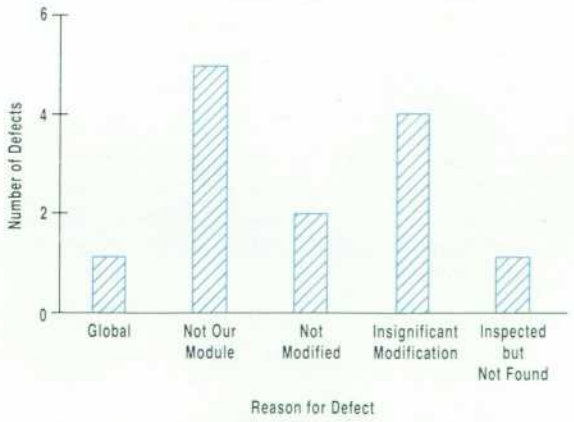


Fig. 7. Defects found by customers.

Benefits

Besides the cost savings realized by finding and fixing defects before they reach the customer, there are other benefits associated with doing code inspections. These benefits are not easy to measure, but they do have an impact on quality and productivity.

- Code inspections allow defects to be found early in the product development cycle. This has the benefit of reducing the number of product builds and certification cycles required to complete the development of a product. This benefit is difficult to quantify because there is no way to measure the build cycles that might be required for a given number of defects. However, one product build and certification cycle is a minimum of 40 hours in our environment.
- The code associated with a defect found during a code inspection is immediately identified at the inspection. When a defect is found by testing, all that is known is that something somewhere doesn't work correctly, and the additional work necessary to identify the lines of code that are at fault is unknown.
- Because code inspections require a great deal of communication among the participants, cross training and idea sharing are byproducts of the inspection process. Other engineers involved become familiar with modules they did not write, and acquire a better understanding of the entire product. Also, engineers from other projects acquire a better understanding of products other than their own.

Issues

Four issues came out of the code inspection process. These were in the area of procedures or aspects of the implementation rather than condemnations of the process in general.

First, the primary consideration in selecting inspectors was their ability to read and comprehend the programming language being inspected. A lack of understanding of the design was not considered an impediment to participation. However, this was an impediment to making the inspections as effective as possible because the inspectors could not always effectively identify situations where the implementation did not match the intended design.

This led to the second area of difficulty—the need for formal design reviews. The lack of formal design reviews results in engineers outside of the project having little familiarity with the details, or even in some cases with the overall design of a product. A lack of a design review process inhibits the effectiveness of code inspections on projects with a small number of engineers.

The third issue involves deciding when a module should be inspected. We determined that the code inspection process should be used after the developer believes that the designed functionality is correctly implemented, and before testing is done. All inspectors and readers should be familiar with the overall product design and the implementation of the module being inspected.

Finally, the defects that were missed and could have been found in the inspection process indicated that we need to find a way to improve the methods used to identify potential defect areas during the process.

Conclusion

Based on the data collected about the use of code inspections, and the data concerning the cost of finding and repairing defects after the product has been released to the customer, it is clear that the implementation of code inspections as a regular part of the development cycle is beneficial compared to the costs associated with fixing defects found by customers.

The formal code inspection process is a significant benefit in fostering quality and productivity in product development. The formal process is structured and requires documentation and accountability. These attributes make it easy to measure and improve the process. From our experience with informal inspections, it is not clear whether the process provides any benefits. However, as an aid to improving the quality of the implemented code, it is worth further investigation.

Acknowledgments

We would like to thank Jim Scaccia, Bill Toms, Bob Devereaux, Frank Heartney, Don Darnell, Guy Hall, Jim Lewis, Bruce Smith, and Peggie Silva for their participation in the code inspections that were performed. A special thanks to Sherry Winkleblack for providing an environment that made this type of experimentation possible.

References

1. M. E. Fagan, "Advances in Software Inspections," IEEE *Transactions on Software Engineering*, Vol. SE-12, no. 7, July 1986, pp. 744-751.
2. M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM System Journal*, Vol. 15, no. 3, 1976, pp. 182-211.
3. T. D. Crossman, "Inspection Teams, Are They Worth It?" *Proceedings of the Second National Symposium, EDP Quality Assurance*, Chicago, Illinois, March 1982.
4. W. T. Ward, "Software Defect Prevention Using McCabe's Complexity Metric," *Hewlett-Packard Journal*, Vol. 4, no. 2, April 1989, pp. 64-69.
5. T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, Vol. SE-2, no. 4, Dec. 1976, pp. 308-320.

Authors

October 1991

6 Component Monitoring System

Christoph Westerteicher

Chris Westerteicher was the project manager for the HP Component Monitoring System computer module and displays. As a result of his team's efforts, production of the HP Component Monitoring System has been automated and streamlined to a major extent. With parts standardization, only 300 parts are needed for the entire system. Chris joined HP's Böblingen Medical Division in 1980, shortly after earning a diploma in electrical engineering from the University of Stuttgart. He became an R&D designer of displays and interface cards for HP medical monitors, and is currently a project manager for anesthesia information systems. Chris is the author of a technical article on IC design productivity and is a member of the VDE. Born in Stuttgart, he lives in Leonberg, is married, has a daughter, and enjoys swimming, gardening, and traveling.

10 Hardware Architecture

Werner E. Heim

R&D engineer Werner Heim joined HP's Böblingen Medical Division in 1986, soon after earning his electronics engineering diploma from the University of Braunschweig in Germany. He developed computer module hardware, including the backplane, CPU, utility CPU, and EPROM, as well as the utility CPU firmware for the HP Component Monitoring System. Born in Giessen-Hessen, Germany, Werner lives in Herrenberg, Baden-Württemberg, and enjoys music and books.

Christoph Westerteicher

Author's biography appears elsewhere in this section.

13 Software Architecture

Martin Reiche

Project leader Martin Reiche was responsible for the fundamental design of the software architecture, operating system and development environment, and patient signal processing software for the HP Component Monitoring System. His team's efforts resulted in automation of all external activities and a smooth integration of each module into the monitoring system. This produced enhanced product reliability and efficiency. After joining HP's Böblingen Medical Division in 1982, he developed ECG and respiratory signal processing for the HP 78832 and 78833 neonatal patient monitors. Martin received his electrical engineering diploma in 1981 from the University of Wuppertal. Born in Wuppertal near Cologne, Germany, he lives in Gäufelden, is married, and has one child. He enjoys bicycling, music, and natural and life sciences, especially psychology.

19 Parameter Module Interface

Winfried Kaiser

Winfried Kaiser was the project leader and designer for the parameter module interface, front-end firmware, module rack, and some of the parameter modules for the HP Component Monitoring System. His efforts resulted in a front-end link that provides fast response, optimum use of available bandwidth, configuration detection, and synchronization for a wide variety of modules. After earning his engineering diploma in 1982 from the University of Karlsruhe, Winfried joined HP's Böblingen Medical Division in 1982. He has developed hardware and firmware and been a project leader for several patient monitoring systems, and is now a project manager for patient monitoring products and enhancements. Born in Lahr in the Black Forest, Winfried lives in Böblingen, is married, has one son, and enjoys traveling, swimming, and family activities.

21 Measuring the ECG Signal

Wolfgang Grossbach

Hardware R&D engineer Wolfgang Grossbach designed and developed the ECG/respiration HP M1001A and M1002A modules and mixed analog-digital application-specific integrated circuits (ASIC) for the HP Component Monitoring System. The modules and circuits produced significant reductions in cost, power consumption, size, and component count. Wolfgang joined HP's Böblingen Medical Division in 1985, shortly after receiving his electronic engineering diploma from the University of Stuttgart. Born in Salzburg, Austria, Wolfgang lives in Murr, Germany. He is married, has two daughters, and enjoys jogging, bicycling, photography, and reading.

25 Blood Pressure Module

Rainer Rometsch

Mechanical design engineer Rainer Rometsch developed the pump assembly for the noninvasive blood pressure module and plastic parts for the display front assembly in the HP Component Monitoring System. The result of his efforts is one of the smallest self-contained noninvasive blood pressure modules in the world—a unit that can be built in about two minutes. Rainer joined the R&D unit at HP's Medical Products Group Europe in 1986, and is now working on respiratory gas measurement. He is named as an inventor in a patent on noninvasive blood pressure measurement. He received an engineering diploma in 1986 from the Engineering School of Furtwangen. Born in Mühlheim/Donau (Baden-Württemberg), Rainer lives in Wildberg in the Black Forest. He is married, has two children, and enjoys working on his house and garden and restoring old BMW motorcycles.

26 Stripchart Recorder

Leslie Bank

Project manager Les Bank helped develop the two-channel stripchart recorder for the HP Component Monitoring System. The efforts of his team resulted in a breakthrough in reducing the size, cost, and power consumption of monitoring recorders. Les, a member of the IEEE, earned a BS degree in electrical engineering in 1969 from the City College of

New York and an MS degree in electrical engineering in 1973 from Northeastern University. He joined HP's Waltham Division in 1973 as a production engineer, working on patient monitoring systems. Before joining HP, Les was a design engineer with Raytheon Co. Born in New York City, he lives in Framingham, Massachusetts, is married, and has two children.

29 Human Interface

Gerhard Tivig

Gerhard Tivig was the project leader for development of the human interface for the HP Component Monitoring System. He also developed software for the alarm manager and localization tools that allow efficient generation of local language versions so the monitor can be used worldwide. Gerhard joined HP's Böblingen Medical Division in 1980, where he researched and developed display software and invasive pressure parameters for HP monitoring systems. He received an engineering diploma in 1975 from the Technical University in Bucharest, Romania, and later worked for two years as a system programmer for bedside monitor software systems at Mennen Medical Center in Israel. He is named an inventor in a European patent for the Component Monitoring System's human interface. Born in Bucharest, Gerhard lives in Böblingen. He is married, has two daughters, and enjoys family activities and traveling.

Wilhelm Meier

Wilhelm Meier was the design engineer responsible for the HP Component Monitoring System's human interface design, simulation, and implementation. He designed an intuitive, easy-to-use consistent control system for all applications and all members of the HP Component Monitoring System family. He joined HP's Böblingen Medical Division in 1982 and developed human interface software for the HP 78353, 78834, and 78352 medical monitors, gaining experience in human factors and human interface design. He is named an inventor in a European patent for the Component Monitoring System human interface. He is now responsible for improvements and enhancements to the Component Monitoring System human interface software. Wilhelm earned an electrical engineering diploma from the Technical University of Hannover in 1981. Born in Obernkirchen in Lower Saxony, he lives in Herrenberg, Baden-Württemberg, is married, and has two children.

37 Globalization

Gerhard Tivig

Author's biography appears elsewhere in this section.

40 Physiological Calculation

Steven J. Weisner

As the software project leader for data management of the HP Component Monitoring System bedside monitor, Steve Weisner was responsible for the system's external specifications and contributed to software development. His team's efforts resulted in a physiological calculation application that reduces the large amount of raw vital-signs data into derived values the clinician uses to assess a patient's condition. He joined HP's Waltham Division in 1982 and has worked as a project leader for the HP central station and as a software engineer for the HP arrhythmia monitoring system, SDN interface, and patient data management systems. His professional specialties include human interface design and clinical information management. Steve is now a software project leader for HP cardiac care systems, responsible for external specifications and user interface design. Before joining HP, he was a software engineer with Cornell University. He received a BA degree in 1976 in biology from Cornell University, and an MS degree in 1981 in biomedical engineering from the University of Wisconsin. A member of the IEEE, he is the author of technical articles in the IEEE Transactions on Biomedical Engineering and in the Proceedings of the Human Factors Society. Born in Paterson, New Jersey, he lives in Lexington, Massachusetts, has a daughter, and enjoys bicycling and sailing.

Paul Johnson

Paul Johnson, a software development engineer whose professional interests include real-time operating systems, implemented the hemodynamic, oxygenation, and ventilation physiologic calculation displays and formulas for calculating the displayed values for the HP Component Monitoring System. These calculated values are good predictors of major malfunctions or mortality in intensive care patients. Paul joined HP's Waltham Division in 1978 after receiving a BSEE degree in 1968 from Purdue University and an MSCP degree in 1986 from the University of Lowell in Massachusetts. He was a software engineer for HP's computer-aided manufacturing department, a production engineering manager, and a development engineer in R&D. Before joining HP, he was a development engineer with the Medical Division of American Optical Co. and a hardware engineer with Raytheon Corp. A six-year U.S. Navy veteran, Paul was born in Elkhart, Indiana, and lives in Groton, Massachusetts. He is married, has two children, and enjoys playing golf and listening to jazz.

44 Mechanical Implementation

Karl Daumüller

Karl Daumüller was the mechanical design project leader for the computer module, displays, and mounting hardware for the HP Component Monitoring System. He helped reduce the number of parts dramatically over previous patient monitoring designs. After joining HP's Böblingen Calculator Division in 1979, he served for two years as a process and production engineer for desktop computers and peripheral products. Karl worked as a mechanical design engineer for over eight years at HP's Böblingen Medical Division, developing the HP 7835x family of patient monitors, HP 8040 and 8041 cardiotocographs, and mounting hardware for hospital installations. Now the virtual source engineering manager for the Böblingen Manufacturing Operation, he is responsible for centralized sourcing of sheet metal and cabinet parts for all German manufacturing divisions. Karl received an engineering diploma from the Engineering School of Esslingen in 1979. Born in Stuttgart in Baden-Württemberg, he lives in Filderstadt, is married, has four children, and enjoys family and church activities and gardening.

Erwin Flachsländer

Mechanical engineer Erwin Flachsländer was responsible for the mechanical design of parameter modules for the HP Component Monitoring System. He helped design an enclosure that can be assembled and serviced without any tool. Erwin joined the R&D division of HP's Böblingen Medical Division in 1985, shortly after receiving his mechanical engineering diploma from the Engineering School of Ulm. At HP, he has worked on a TC-pO_2/CO_2 calibrator system. He is named as an inventor in a patent on a connector for the blood pressure monitor. Before joining HP, Erwin was a mechanic in manufacturing and production at two different companies. Born in Kempten, Bavaria, he lives in Böblingen, is married, has two children, and enjoys motorbiking, photography, and playing a music synthesizer.

49 Automated Test Enviornment

Dieter Göring

Software quality engineering manager Dieter Göring developed the automated software test environment for the HP Component Monitoring System. He designed the AUTOTEST tool and developed a suite of structured tests, which runs 60 hours of automatic tests and 45 hours of semiautomatic tests overnight and weekends. Dieter received his engineering diploma from the Engineering School of Furt-

wangen in 1967. After joining HP's Computer Systems Division in Böblingen, Germany in 1973, he served as an R&D engineer, project engineer for automated test systems, communications and office automation manager, and information technology manager. Before joining HP, he was an application programmer and a software project engineer. Born in Düsseldorf, he lives in Böblingen, is married, has two children, and enjoys sports, music, reading, and traveling.

52 Production and Final Test

Otto Schuster

Production engineer Otto Schuster was responsible for the production process development and design for manufacturing of the HP Component Monitoring System. He helped ensure the concurrent design of the product and its production processes—the first HP product designed in surface-mount technology in the Böblingen technology center. Otto joined HP's Böblingen Medical Division in 1979, shortly after receiving an engineering diploma in electrical engineering from the Engineering School of Esslingen. He served as a production engineer for the HP 78352, 78353, and 78354 monitoring systems. A resident of Heimsheim in Baden-Württemberg, where he was born, Otto is married, has two sons, and enjoys skiing, bicycling, and gardening.

Joachim Weller

Production engineer Joachim Weller was responsible for the design and development of front-end production test systems for the HP Component Monitoring System and for HP-UX tools for statistical process control. He also worked closely with R&D on design testability, and helped reduce manufacturing cycle time on the monitoring system. After joining HP's Böblingen Medical Division in 1984, he was responsible for the production of patient monitors and the development of a new generation of final test systems for the HP 78352, 78354, and 78356 patient monitoring systems. Joachim received an engineering diploma in 1984 from the Engineering School in Esslingen. Born in Stuttgart in Baden-Württemberg, he lives in Herrenberg, is married, has two children, and enjoys amateur radio, windsurfing, camping, and studying foreign languages.

55 Cost of Software Defects

William T. Ward

Jack Ward joined HP's Waltham Division in 1982 and has worked on the software and firmware development of critical care bedside monitors, arrhythmia analysis systems, and medical database systems. As the manager of software quality engineering, Jack is now responsible for testing each of these products for use in medical environments. He earned a BS degree in linguistics in 1972 from the University of Illinois and an MS degree in computer science in 1984 from Boston University. Before joining HP, he worked as a software support engineer for Data General Corp. The author of several articles published in technical journals, Jack teaches undergraduate and graduate courses in C, C++, and software quality at Boston University. Born in Winona, Mississippi, he lives in Brookline, Massachusetts, is married, has three children, and enjoys music, gardening, and jogging.

58 Code Inspections

Frank W. Blakely

Client/server computing and software quality improvement are the professional interests of Frank Blakely, a software engineer at HP's Applications Support Division. Frank joined HP's Information Resources Operation in 1980. He helped develop a code inspection process tool for HP's Data Management Systems Division that is now used early in the software development cycle to help improve the quality of software products and the productivity of development engineers at his division. Before joining HP, Frank was an MIS programmer at LooArt Press, Inc. and a programmer/analyst with Information Resources Ltd. He is a graduate of Colorado College, earning a BA degree in mathematics in 1973, and a graduate of the University of Oregon with an MS degree in mathematics in 1977. Frank is named as an inventor in a patent pending on HP cooperative services. Born in Colorado Springs, Colorado, Frank lives in Roseville, California, is married, and enjoys cross-country skiing, hiking, playing board games, and participating in the Placer County Fair Association.

Mark E. Boles

A software quality engineer in HP's Applications Support Division, Mark Boles is responsible for metrics collection, process improvement, and implementing processes and new methodologies for software development. He helped develop a code inspection process model for HP's Data Management Systems Division that is now used early in the software development cycle to help improve the quality of software products and the productivity of development engineers at his division. Mark joined HP's Computer Systems Division in 1982, shortly after earning a BSEE degree from San Jose State University. He became a hardware reliability engineer for environmental and reliability testing for the HP 3000 computer and process improvements, and later a software quality engineer responsible for test and productivity tools. Client-server application integration is his professional interest. Mark is a member of the American Society of Quality Control. Born in National City, California, he lives in Roseville, is married, and enjoys car restoration, snow and water skiing, and building electric trains with his three-year-old son.

69 HP Vectra 486 PC

Larry Shintaku

Project manager Larry Shintaku and his team developed the HP Vectra 486 PC accessories. Their new development processes helped HP market the first personal computer to use the Intel486 microprocessor with an EISA bus. Larry joined HP's Data Terminals Division in February, 1980, two months after receiving a BS degree in electrical engineering from Fresno State College in California. As a hardware designer, he developed the HP 2623A/2 terminal graphics subsystem, and later, as a project manager, he helped develop the expanded memory card for the HP 150 TouchScreen PC. Larry is now managing the development of the next generation of HP Vectra 486 PC products. Before joining HP, he worked in digital communications with Dantel Inc. A member of the IEEE, Larry was born in Fresno and lives in Union City. He enjoys racquetball, low-budget motion pictures, and softball.

73 EISA Connector

Michael B. Raynham

Exploring the creative links between art and technology are among the professional interests of Mike Raynham, an R&D development engineer at HP's California Personal Computer Division. Mike helped design the HP Vectra 486 PC and its Extended Industry Standard Architecture (EISA) connector. He helped achieve an extremely fast development cycle of six months from initial concept to production of the first EISA connectors, which allow EISA amd ISA I/O cards to be handled in the same connector. Mike also worked on hardware development for the HP 2116B, 2100A, and 3000 computers, the HP 2644A, 2645A, 2648A, 2647A, and 2703A terminals, and the HP 150 TouchScreen II, HP Vectra RS16/20, and HP Vectra 486 personal computers. Before joining HP in 1963 in Bedford, England, he worked as a film recording engineer with the British Broadcasting Corporation and as an apprentice with British Aerospace. He is named as an inventor in patents or patents pending for a display panel, digital encoding/decoding techniques, DRAM on-chip error correction, low-cost connectors, and IC surface-mount process defect detection designs. Mike is a member of the IEEE and acted as chair of the Desktop Futurebus+ subcommittee. He earned an H.N.C. degree in 1962 from Luton College of Technology and an MS degree in 1971 from Santa Clara University, both in electrical engineering. Born in Winnersh, England, he lives in the Santa Cruz mountains in California, is married, and has two sons. He enjoys clay sculpture, ceramic tile painting, and watercolor painting.

Douglas M. Thom

As an engineer and project manager, Doug Thom performed research and development on HP's fiber optic components. He is now a section manager at HP's California Personal Computer Division. Doug helped achieve an extremely fast development cycle of six months from initial concept to production of the first EISA connectors, which allow EISA and ISA I/O cards to be handled in the same connector. He joined HP's Optoelectronics Division in 1980, and is now listed as an inventor on a patent on fiber optic component design and an inventor on a patent pending on the EISA connector. Before joining HP, he was a consumer product designer with National Semiconductor Corp. and Fairchild Semiconductor. He earned BS degrees in 1975 in electrical and mechanical engineering from the University of California at Davis. Born in San Mateo, California, Doug lives in Woodside, California, is married, and has a son. He enjoys sailing, carpentry, architecture, cooking, and gardening.

78 HP Vectra Memory Controller

Marilyn J. Lang

Marilyn Lang joined HP's California Personal Computer Division in 1981 and worked on IC test vector generation and simulation for the HP 150 video controller ASIC. She also designed the HP-HIL port extender used in the HP Vectra RS/16, RS/20 and ES/12 PCs. Marilyn then worked on memory subsystem analysis and design for the HP Vectra 386 PC, which led to her work on the memory controller ASIC design for the HP Vectra 486 PC. Her efforts helped produce a high-performance, burst-mode capability that enhanced the competitive price/performance of the HP Vectra 486 PC. Marilyn earned a BS degree in 1975 in chemistry from the Southern Illinois University at Carbondale, an MS degree in biochemistry in 1979 from the University of Illinois at Urbana, where she also studied computer science, and an MSCSE degree in 1988 in computer science and engineering from Santa Clara University. Born in Chicago, Illinois, Marilyn lives in Milpitas, California, is married, has a daughter, and enjoys gardening, science fiction/fantasy, and classical music. She is a member of the National Gardening Association and various humane and wildlife societies.

Gary W. Lum

Project manager Gary Lum, whose professional specialties include memory technology and design, was responsible for developing the HP Vectra 486 memory controller and memory subsystem architecture. His efforts resulted in a high-performance burst-mode memory capability that helped enhance the competitive price/performance advantage of HP's 25-MHz system. Gary joined HP's Data Terminals Division in 1979 and worked as a project manager for HP Vectra PC accessory cards, on the HP Vectra PC and HP Vectra ES PC, and on IC design for the HP 150 TouchScreen II PC. A member of the IEEE, Gary was born in Syracuse, New York, lives in Cupertino, California, is married, and enjoys film and film history, photography, and gardening.

83 HP Vectra BIOS

Thomas Tom

R&D software engineer Thomas Tom joined HP's California Personal Computer Division in 1989 and developed the firmware for security features and the Micro-DIN mouse support for the HP Vectra 486 Basic I/O system (BIOS). He is now designing software to support features of HP's newest PCs. Thomas is a 1983 graduate of the California Polytechnic State University at San Luis Obispo with a degree in electrical engineering. Before joining HP, he developed real-time satellite simulation software for Stanford Telecommunications, Inc., and developed test software to evaluate integrated circuits at NEC Corp. Thomas lives in San Francisco, where he was born, and enjoys basketball, tennis, bowling, and biking.

Irvin R. Jones, Jr.

Computer and system architecture and artificial intelligence are the professional interests of Irvin Jones, a software engineer who helped develop the HP Vectra 486/25T system BIOS. His efforts helped ensure that the HP Vectra 486 PC makes the most efficient use of its Intel486 microprocessor, the EISA bus, and new memory subsystem. Since joining HP's California Personal Computer Division in 1988, Irvin also helped design the system BIOS and system utilities disk for the HP Vectra LS/12, the HP Vectra 486/33T PC, and the HP Vectra 386 PC. Before joining HP, Irvin worked as a digital designer on photocopier function cards for IBM, on the microcontroller design of professional video systems for Sony Corporation, and on a parallel computer peripheral interface for Bell Communications Research. A member of the IEEE and the Triathlon Federation, Irvin is named as an inventor of two patents pending for HP's PC BIOS. He earned a BS degree in electrical engineering from Stanford University in 1982, an MS degree in computer engineering in 1986 and an MS degree in computer science in 1988 from the University of California at Santa Barbara. Born in Dayton, Ohio, he lives in San Jose, California, and enjoys triathlon competition, playing jazz on the vibraphone and drums, and collecting comic books.

Christophe Grosthor

Real-time low-level software design and application development are the professional specialities of Christophe Grosthor. He joined HP's Grenoble Personal Computer Division in 1988 as a software engineer and designed software for the HP Vectra 486 PC BIOS and the HP Vectra 386/25T PC BIOS. He helped ensure that the HP Vectra 486 PC makes the best use of its Intel486 microprocessor, the EISA bus, and a new memory subsystem. He received an MS degree in electronics from the University of Toulouse, France, in 1986 and a software engineering degree from Ecole Nationale Superieure des Telecommunications de Bretagne in 1988. Before joining HP, Christophe worked on object-oriented compiler design as a software engineer for Interactive Software Engineering, Inc. in Santa Barbara, California. Born in Strasbourg, France, he lives in Grenoble, is married, and enjoys sports, mountain hiking, and traveling.

Viswanathan S. Narayanan

Software development engineer Suri Narayanan developed the EISA initialization procedures for the HP Vectra 486 PC BIOS. His efforts helped ensure that the HP Vectra 486 PC makes the best use of its Intel486 microprocessor, the EISA bus, and a new memory subsystem. After joining HP's California Personal Computer Group in 1988, he developed BIOS designs for the HP Vectra 386/25 PC, and is now working on future HP personal computer products. Suri received a BS degree in 1980 from the Regional Engineering College in Warangal, India, and an MS degree in electrical engineering in 1985 from the University of Wyoming. Born in Secunderabad, India, he lives in Fremont, California, is married, and enjoys gardening and playing basketball.

Philip Garcia

Phil Garcia was responsible for the EISA CMOS BIOS interface and the cache control BIOS code. He has worked on keyboard microcontroller firmware design and PC utilities design for HP Vectra PCs. After joining HP's Data Terminals Division in 1982 as a development engineer, he worked on analog design for the HP 2700 color graphics workstation and the HP 150 Touchscreen PC, and on EMI/RFI compliance design for the HP 150 and HP Vectra PC. A Stanford University graduate, he received a BAS degree in economics and electrical engineering in 1979, and an MSEE degree in analog IC design in 1981. Phil is named as an inventor in two pending patents on PC BIOS designs. Born in New York City, he lives in Saratoga, California, is married, and enjoys hiking, skiing, old movies, and museums.

92 Vectra Performance Analysis

John D. Graf

Development engineer John Graf joined HP's California Personal Computer Division in 1989, right after earning a BS degree in electrical engineering from Rice University. He then designed hardware tools to measure the performance of existing PCs, and developed mathematical and software models to evaluate and predict the performance of future architectures. These performance tools were used to design and enhance the performance of the HP Vectra 486 PC and the HP Vectra 486/33T PC. John's professional interests are focused on evaluating the performance characteristics of CPU, cache, memory, and video. Born in Baton Rouge, Louisiana, he lives in Sunnyvale, California, is married, and enjoys Cajun cooking, bodysurfing, and volunteer work in a church youth group.

David W. Blevins

As a hardware development engineer at HP's California Personal Computer Division, Dave Blevins developed the hardware for the backplane I/O monitor, a noninvasive tool that helps analyze a personal computer's subsystem workload and provides data for predictive system modeling. Dave, who joined HP's Southern Sales Region in 1982 as a customer engineer in the New Orleans sales office, was a member of the HP Vectra RS/20 development team, a CAE tools support engineer, and a hardware development engineer. He left HP in 1990 to join MINC, Inc. in Colorado Springs, Colorado, as an applications engineer. Dave received a BSEE degree in 1982 from Washington State University. Born in Middletown, Ohio, he lives in Colorado Springs, Colorado, and is married. Dave enjoys music synthesizers and computer music sequencing, high-performance motorcycles, mountain biking, and playing guitar in a local jazz-fusion group.

Christopher A. Bartholomew

Chris Bartholomew joined HP's California Personal Computer Division in 1989 soon after earning BS degrees in computer science and in electrical engineering from Texas A&M University. As an HP system performance engineer, Chris developed the disk, I/O, and BIOS performance modeling hardware and software tools that help to noninvasively analyze a personal computer's workload in these subsystems. These tools were first used to design and enhance the performance of the HP Vectra 486/25T PC and the HP Vectra 486/33T PC. Chris' professional interests include embedded programming, object-oriented programming, performance modeling, and multiprocessor architectures. He is a member of the IEEE and the IEEE Computer Society. Before joining HP, he was a systems programmer at Compaq Computer Corp. Born in Jackson, Michigan, Chris lives in Fremont, California, is married, and enjoys camping, radio-controlled airplanes, fishing, and racquetball.

The HP Vectra 486 Personal Computer

The HP Vectra 486 series of computers uses the Intel486™ microprocessor, a custom-designed burst-mode memory controller, and the HP implementation of the Extended Industry Standard Architecture (EISA).

by Larry Shintaku

The HP Vectra 486 PC was the first of HP's new generation of personal computers using the Intel486 microprocessor and the EISA (Extended Industry Standard Architecture) bus architecture. The Intel486 is a high-performance microprocessor that integrates the CPU, 8K bytes of cache, and a math coprocessor onto one chip running at a clock speed of 25 or 33 MHz.* The CPU instruction set is optimized to execute instructions and move data in fewer clock cycles than its predecessor, the Intel386 microprocessor. The EISA bus was defined by an industry consortium of which HP is an active member. The EISA bus definition objectives were to migrate the existing 16-bit Industry Standard Architecture (ISA) bus into a 32-bit bus, improve the DMA performance, and provide support for multiple bus masters while maintaining backwards compatibility with all existing ISA backplane I/O cards.

The HP Vectra 486 development objective was to deliver these two new technologies to market quickly. We were presented with several technical and product development challenges in trying to meet this objective. These challenges included:

- Defining a physical bus connector that would accommodate both EISA and ISA cards (see article on page 73)
- Incorporating all the new technical design features that EISA offers
- Developing performance enhancements targeted for the memory and mass storage subsystems.

*Recent releases of the HP Vectra 486 series include the Vectra 486/25T and Vectra/33T, the latter uses the 33-MHz version of the Intel486 microprocessor.

System Overview

The Vectra 486 uses the existing upright floor package that is used by the HP Vectra RS series (see Fig 1). Its mass storage, power, and feature options matched our customer requirements for high-end server and CAD applications. The form factors for the printed circuit boards, already defined by the Vectra RS PC package, fixed the amount of logic each board could support, the logic design and printed circuit board partitioning, and the functional, EMC, and performance objectives. The functional objectives were met by partitioning the system components so that follow-on EISA products could easily leverage core components developed by the HP Vectra 486

Fig. 1. The HP Vectra 486 personal computer.

team. The EMC objectives were met by minimizing, through design, source radiation and harmonics caused by mismatched impedances. The design required that clock speeds up to 66 MHz be distributed over several printed circuit boards and connectors. Meeting the EMC objectives was very important since they represented a potential delay in the schedule if regulatory requirements were not met. The performance objectives were met through the development of an Intel486 burst-mode memory controller and a high-performance hard disk subsystem. The burst-mode memory controller is described on page 78 and the disk subsystem is discussed on this page.

The HP Vectra 486 EISA SCSI Subsystem

HP's advanced PC mass storage products have consistently provided customers with performance and conformance to industry standards. The HP Vectra 486 PCs continue this tradition by providing a high-performance storage subsystem that is compatible with EISA and SCSI-2 (Small Computer System Interface) industry standards.

The investigation of customer needs for the first EISA PC, the Vectra 486, revealed that the highest-performance PCs were entering new application areas. Customer and application requirements resembled more those of the workstation or minicomputer user than those of an individual running a word processing application. Demanding compatibility with the IBM PC AT, customers also insisted upon high capacity, performance, and reliability for such applications as PC CAD, multiuser UNIX* operating systems, and multiclient file servers.

HP's California PC Division's (CPCD) advanced storage team responded by developing, along with its invaluable partners at HP's Disk Memory Division (DMD) and Adaptec, a new ESDI (Enhanced Small Device Interface) disk family and Industry Standard Architecture (ISA) disk controller. Each of the new 20-Mbyte/s disk drives from DMD provides up to 670 Mbytes of 16-ms average access time storage at an MTBF (mean time between failures) of 150,000 hours. Adaptec's controller not only supports the drive's data rate, but also provides a 64K-byte read-ahead cache. By continuing to read data past the user's request, the controller's cache already has additional data the user is likely to want later. At its introduction, the Vectra 486's storage subsystem provided excellent performance, capacity, and reliability while staying PC-AT software compatible.

The engineers at CPCD realized that although powerful for its time, the ISA disk subsystem's performance had approached its architectural limits. Further performance improvements could only come with fundamental design changes. Unlike ISA disk subsystems, a new architecture would take full advantage of the EISA I/O bus and other new technologies

The first products based on this new architecture appeared with the introduction of the Vectra 486/33T. Targeting once again the multiuser UNIX operating system environment and Novell Netware file server customers, the advanced storage team brought to market the PC industry's first EISA SCSI-2 storage subsystem. Contributors from all disciplines in the PC industry supplied state-of-the-art components. Hardware suppliers developed the EISA SCSI host adapter and the 440-Mbyte to 1000-Mbyte SCSI-2 disk drive family while software suppliers created the industry's first tagged queuing SCSI-2 disk drivers for the Santa Cruz Operation UNIX operating system and the Novell Netware network operating system.

Tagged command queuing is a feature of SCSI-2 that allows a peripheral to intelligently execute I/O requests from the host computer. The peripheral can, but does not have to, reorder the sequence of the I/O command stream to optimize its execution. By use of the queue tag, the peripheral can associate the I/O request with the data, thereby not requiring that the data be associated with a single pending I/O request. Fig. 1 illustrates this concept. Five I/O requests are generated all at once in the sequence shown. The peripheral device decides to reorder the requests for optimal execution and gives the completed requests back to the host adapter in the optimal order. The host adapter associates the returned data with the correct tag, and reassembles the I/O thread originated by the system.

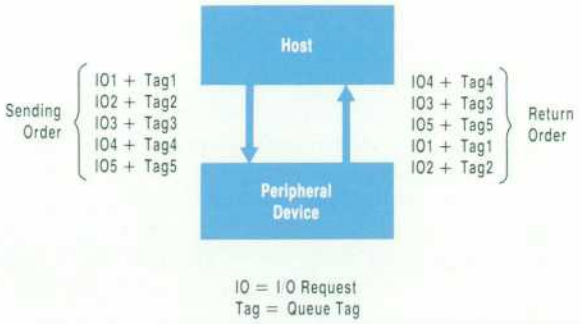


Fig. 1. Tagged queuing.

From the 32-bit bus master host adapter, which supports up to 10-Mbyte/s single-ended fast SCSI, to the 12-ms access time caching disk drive, the hardware components represent some of the best of today's technology applied to the PC environment. However, choosing the highest-performance components is only part of the development story. Tuning each subsystem component for UNIX and Netware application performance made the Vectra 486/33T more than the sum of its parts. The team optimized each of SCSI-2's performance parameters and features while staying within the industry standard. Additional HP proprietary performance tuning of the drive's 128K-byte cache further enhances system performance.

Today's PC SCSI subsystem offering is just the beginning. The architecture can support a wide variety of SCSI peripherals available industry-wide. For the first time PC customers can have access to such diverse peripherals as CD-ROM, tape, DAT, and removable magnetooptical storage. In addition, this EISA SCSI architecture allows for system performance growth as the industry continues to develop SCSI-2 to its full potential. The SCSI-2 storage subsystem architecture will meet the challenge of future customer needs for both added performance and greater peripheral connectivity.

Differentiating products that are based upon widely available industry standards is difficult. After all, an Intel486 microprocessor runs just as fast for another PC vendor as it does for HP. HP's strength lies not only in its SPU architecture but also in its ability to fulfill particular customer needs. Every Vectra 486 and 486/33T storage subsystem has been optimized to provide customers with the best performance and reliability in an EISA PC.

Mike Jerbic
Development Engineer
California PC Division

*UNIX is a U.S. registered trademark of UNIX System Laboratories in the U.S.A. and other countries.

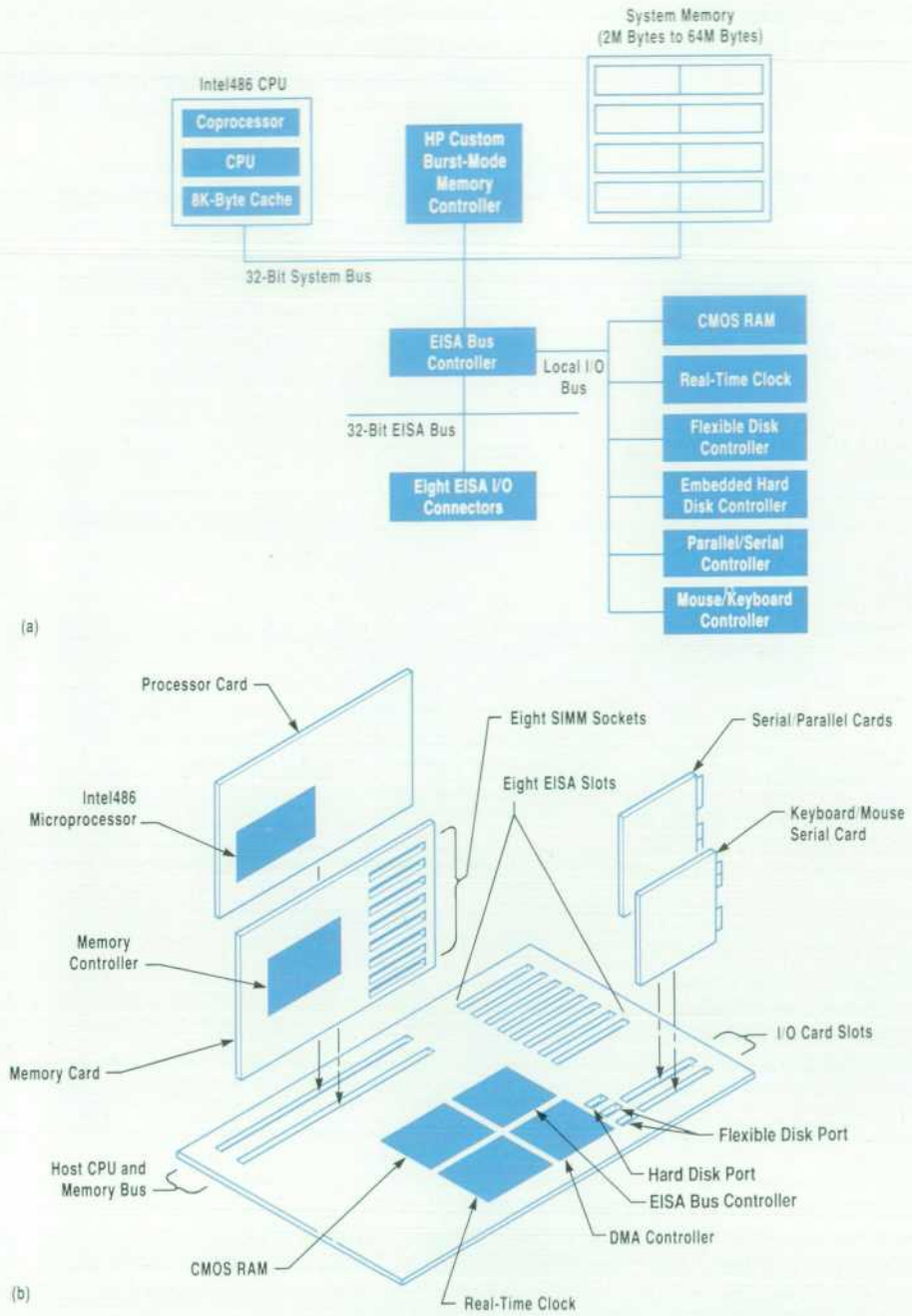


Fig. 2. (a) Logical partitioning of the Vectra 486 system. (b) Physical partitioning of the Vectra 486 system

The physical partitioning of logic (see Fig. 2a) divides the system into five printed circuit board assemblies (see Fig. 2b). The core assemblies consist of a motherboard containing the core EISA control logic and local I/O logic, and two vertical printed circuit assemblies containing the physical I/O connectors for the keyboard, mouse, and three I/O ports: two serial and one parallel. These core assemblies can be leveraged for future EISA products. The remaining two assemblies are the CPU and memory printed circuit board assemblies. The CPU board contains the Intel486 and related control logic, and the memory board contains the HP burst-mode controller and SIMM (single in-line memory module) sockets for RAM memory upgrade.

Product Development Overview

The time-to-market objectives for the HP Vectra 486 product required a new approach to the normal development process used for past products. Managing three parallel technology developments, the Intel486, the EISA bus controller chips, and the memory controller, and keeping the project on an aggressive schedule was the main challenge for the HP Vectra 486 team. To add to the challenge, two of the three critical technologies in development were outside HP (i.e., the Intel486 and the EISA bus controller). The first step was to outline the overall development approach that would meet the time-to-market objectives with a product that met our quality standards. The development process also had to be flexible enough to track

the parallel development of the Intel486 processor and the EISA chipset. The resulting development approach consisted of two main phases of execution (see Fig. 3). Our traditional product development cycle required three to four phases.

We felt we could combine the breadboard and the laboratory prototype phases and still meet all the requirements for determining feasibility, design for manufacturing, design for EMC, quality, and functional verification in the first phase. This approach took less time and cut out redundant or ineffective processes. After making the necessary changes to the design in the first phase of the project, the second phase focused on getting the manufacturing process ready for volume production and executing product qualification testing for HP environmental, regulatory, and quality requirements.

Many processes were performed in parallel to reduce the amount of development time which, in many cases, increased the risk of having to address dependency problems caused by something failing. Other important processes were put in place to address these potential development roadblocks. An example was the establishment of direct interactive technical communication links with outside companies for technical reviews and changes. This liaison saved days or weeks of development time for the HP Vectra 486 team. The team also made sure that contingency plans were made for the critical processes such as printed circuit board layout, fabrication, prototyping, and the tools of development to ensure that progress would be maintained in most circumstances.

The HP Vectra 486/33T

During the latter stages of the HP Vectra 486/25T development program, development of the Vectra 486/33T was initiated. This system, designed around the new Intel486 33-MHz microprocessor, provides higher performance at a lower cost in LAN server, multiuser, and PC CAD applications. By combining this processor technology with enhanced memory and mass storage subsystems and by building on the achievements of the Vectra 486/25T, the Vectra 486/33T program was driven by three major objectives: fast time to market, high performance, and high quality. To meet the challenges of these three objectives, the development team implemented two key strategies; the first was focused system design understanding, and the second was ongoing process improvements.

System Development

A strategy of the HP Vectra 486 implementation was to partition the system components to provide easy leverage for follow-on EISA products such as a 33-MHz system. The areas of engineering and product reuse were the package and power system, three core printed circuit assemblies, and the video adapter card. Through performance analysis of the Vectra 486/25T system, we were able to focus on the areas that would significantly contribute to our objectives. These areas included design for 33 MHz, the addition of a high-performance second-level cache, inclusion of both write and memory buffers, and integration of a new high-performance SCSI (Small Computer System Interface) hard disk subsystem.

To achieve the required performance levels for the 33-MHz system, it was determined that a second level cache (in addition to the on-chip Intel486 8K-byte cache) was necessary. Simulations also showed that significant performance gains could be achieved through the addition of a bus write buffer and a memory write buffer. Therefore, the CPU design includes a 128K-byte, 2-way set associative, write-through cache, with one level of bus write buffers. In addition, support for an optional Wietek 4167 floating-point coprocessor was added to further meet the needs of our customers requiring increased floating-point performance for their PC CAD applications.

Further performance simulation and analysis showed that the capabilities of HP's custom burst-mode memory controller, first implemented on the Vectra 486/25T, would continue to offer superior performance, with minor changes for optimizing 33-MHz operation and with the addition of a memory write buffer. Therefore, the design of the memory controller was leveraged for use in this higher-performance system. In fact, the result of the redesign of the memory PCA is a memory board that can support both the Vectra 486/25T and 486/33T, with optimal performance enhancements for both.

During the Vectra 486/33T design, the team was continuously looking for ways to improve the quality and manufacturability of the system. A significant contribution to this goal was made on the CPU board by eliminating all discrete delay lines. This was achieved through the use of delay lines implemented by traces on the printed circuit board. Using simulation and an understanding of the physical properties of the printed circuit board, the team was able to deliver delays with excellent characteristics and margins. This resulted in higher reliability, lower costs, and improved manufacturability.

Process Development

To achieve the fast time to market, the team needed to apply the lessons learned from the efforts of the Vectra 486/25T program. In addition to using the improvements made for that program, several other enhancements were required. To ensure team focus, the team constantly reviewed their decisions against the well-communicated list of "musts" and "wants". Increased levels of simulation were used along with frequent design reviews. New and improved processes were instituted for supporting the prototype systems used in test and verification and for tracking and solving defects found during these phases. The result of all of these efforts was a very efficient system verification cycle leading to a timely manufacturing release of a high-quality product.

Mark Linsley
Section Manager
California PC Division

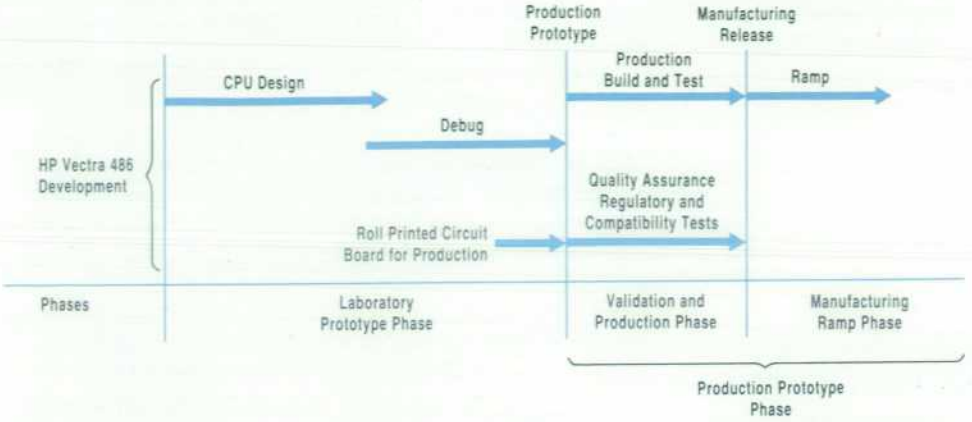


Fig. 3. The Vectra 486 development phases.

Conclusion

The Vectra 486 development team was confronted with the challenge of bringing an Intel486-based product to market almost simultaneously with the announcement of the Intel486 microprocessor. By using new development processes, the HP Vectra 486 was the first computer on the market using the Intel486 CPU and the EISA bus.

Acknowledgments

The author would like to thank the following people who helped to make the development of the HP Vectra 486 a success: Mike Perkins, Mark Hanlon, Anil Desai, Gary Lum, John Wiese, Irvin Jones, Phil Garcia, Jon Won Yuk, Suri Narayanan, Thomas Tom, Becky Smith, Dave Wilkins, Rod Young, Van Dam, Martin Goldstein, Sam Lee, Kirin Mundkur, Kamal Avlani, Kamran Casim, Gary Lin, Otaker Blazek, Wes Stelter, Marilyn Lang, Chris Anderson, Christine Doan, Kevin Owen, Sam Chau, Robert Rust, Shridhar Begur, Christophe Grosthor, Bob Johnson, and Jun Kato.

The EISA Connector

Providing backward compatibility in the EISA connector hardware for ISA I/O boards resulted in a bilevel connector design that provides pins for both bus standards in the same connector.

by Michael B. Raynham and Douglas M. Thom

One of the reasons for the rapid growth of the personal computer (PC) market is the wide variety of compatible software and hardware peripherals available for these machines. This compatibility has been provided by a de-facto industry-standard bus specification called Industry Standard Architecture (ISA). Although started with the original IBM PC system architecture, the standard has evolved to where it can be adopted by any PC manufacturer, thus providing a stable platform for software and hardware development.

The EISA (Extended Industry Standard Architecture) is a superset of the ISA 8-bit and 16-bit architecture. The important features of the EISA specification include:

- Full compatibility with the ISA standard. ISA 8-bit and 16-bit expansion boards can be installed in EISA slots.
- Support for a 32-bit address path and for 16-bit or 32-bit data transfers for CPU, DMA, and bus master devices. (A bus master is a device that drives the address lines and controls the signals for a bus cycle.)
- An efficient synchronous data transfer protocol that provides for normal single transfers and the cycle control required to execute burst cycles up to 33 Mbytes/s.
- Automatic translation of bus cycles between EISA and ISA masters and slaves.
- Support for a bus master architecture designed for intelligent peripherals. With EISA-based computers the bus controller can operate some of the lines on behalf of the bus master.
- A centralized bus arbitration scheme that supports preemption of an active bus master or DMA device. The

EISA arbitration method grants access to the bus for DMA devices, DRAM refresh, bus masters, and bus and CPU functions on a fair, rotational basis.

- Level-triggered, shareable interrupts. Edge-triggered operation ensures compatibility with interrupt-driven ISA devices. Level-triggered operation facilitates sharing of a single system interrupt by a number of devices.
- Automatic configuration of system and expansion boards. EISA expansion board manufacturers provide configuration files and product identification information so that during system initialization these boards can be automatically configured into a system (see page 84).

More detailed information about the EISA bus can be found in references 1, 2 and 3.

Engineers from HP's personal computer group were involved in defining the physical and electrical design of the I/O bus, the board connectors, and the logic controlling bus timing for the EISA bus specification. Their most obvious contribution was the "double-decker" EISA connector. This connector has two levels of pins. The first level maintains ISA compatibility and the second level adds the pins for the EISA bus specification. This article will describe the EISA connector and some aspects of the development partnership that led to the development of the connector and I/O card hardware.

Background

The EISA connector was an important part of the implementation of the EISA bus standard. At the time we started this project there was no connector available that met the general electrical and mechanical characteristics required for EISA. Some solutions were proposed but they were discarded because they were not competitive in size and electrical performance. The IBM Microchannel* bus architecture had already doubled the pitch of contacts from 0.100-inch to 0.050-inch centers on their connectors, and it was felt that the EISA solution must use this contact density to be competitive.

The technical responsibilities for the proposed EISA bus design were divided among a small group of the original EISA consortium companies. The responsibilities for the definition, development, and sourcing for the EISA connector were given to Hewlett-Packard and Compaq Computer Corp.

Because the EISA connector was the first physical evidence of the EISA hardware, it became important from a public relations standpoint that the design not only be backward compatible with ISA, but also be perceived as technically superior (e.g., higher-performance, well designed, etc.).

The availability of production connectors was a serious concern because once the design was finalized the potential demand for connector hardware would be very high. To ensure that a high-volume supply would be available, and to manage the technical risks, it was decided to recruit at least two major connector manufacturers to develop and produce the connector. HP and Compaq Computer Corp. recruited Burndy Corporation and AMP Incorporated into the EISA consortium to participate in the design.

*Microchannel is the bus architecture developed for the IBM Personal System/2 computers.

Organizational Challenges

The connector project was managed primarily by a joint team of HP and Compaq engineers representing the EISA consortium. The team attracted connector manufacturers using the number of customers within the consortium to convince the manufacturers of the magnitude of the business opportunity for EISA connectors. The preliminary design requirements were established by HP and Compaq Computer Corp. as part of the EISA technical specification. This technical specification, which was revised and published periodically by the consortium, was the single specification that all connector vendors used to develop their specific connector designs. The periodic revision of the specification proved very valuable in maximizing the collective technical contributions of the connector vendors. All potential vendors could obtain a set of technical requirements by joining the EISA consortium. These vendors could also recommend technical ideas for the design, which, if adopted, would become part of the specification. All technical contributions incorporated into the specification became the intellectual property of the consortium, and therefore, became available to all members. This process produced a very robust and thorough connector specification by using the collective efforts of all participants, some of whom were direct competitors. Fig. 1 shows the design and development information flow during this process.

The connector's technical specification was a performance-based specification. Except for the basic mechanical dimensions, all parameters were specified based on electrical, environmental, mechanical, or process performance. This performance-based approach allowed each vendor to provide subtle but significant design features in their final

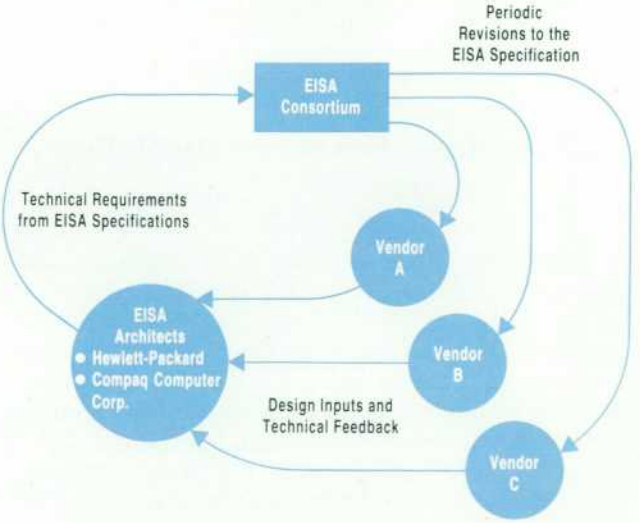


Fig. 1. Information flow during the design and development of the EISA connector. All connector manufacturers received the EISA bus specification and provided feedback to the EISA connector architects without interfacing to other manufacturers. This provided the best possible technical design without compromising vendor confidentiality.

EISA Configuration Software

One of the specifications of the EISA standard defines the process for configuring EISA slots into a computer system. When the EISA consortium was being formed, Compaq Computer Corp. started the initial development of the software for configuring EISA slots. Soon after development began, two HP engineers were assigned to work with Compaq in the development effort.

The configuration software detects the presence of accessory cards inserted into the EISA slots of the computer and provides a process to configure the cards into the system. The configuration process begins with the configuration program reading a configuration file for each of the accessory cards installed in the EISA slots. The configuration file contains information about the card that enables the program to determine the optimum settings for any switches or jumpers on the card. Once the program has determined the required configuration of the accessory cards, it identifies any manual switch settings or changes that may be necessary, and instructs the user to make them. The system configuration information is then written to nonvolatile memory where it is stored and available to the BIOS (Basic I/O System) each time the computer boots up.

HP contributed heavily to the usability features of the configuration software by using a fully equipped camera studio in our usability department to observe people using the configuration utility.

Some of the testing showed that nonprocedural interfaces, such as a windows environment, didn't work in the installation process as well as a procedural interface. (A procedural interface presents a series of steps—procedures—that guide the user. A nonprocedural interface simultaneously presents a number of tasks from which the user must select the next step.) The initial version of the configuration utility used a windows-like interface. The later versions of the configuration utility were changed to use a procedural interface. In addition, help screens have been improved, and some of the processes have been combined into a single task. We also improved the code to make it run faster, eliminating a perception by some users that the system was hung up.

The usability testing continues, and the latest version of the configuration software has a much improved user interface. This new interface is fully procedural, and tests have shown that even the most inexperienced users can effectively configure an HP Vectra computer.

More about the configuration files and the EISA slot initialization process can be found in the article on page 83.

Tony Dowden
Learning Products Engineer
California PC Division

connector design. This preserved a healthy competitive environment among the connector vendors and allowed them to market their individual features and benefits.

Customer and Vendor Relations

The existence of the consortium provided the technical benefits mentioned above and it also freed HP, Compaq, and other consortium members to establish the necessary customer and vendor relationships that would eventually be necessary to manufacture products. Nondisclosure agreements were established between HP and several connector manufacturers. This allowed HP to negotiate supply contracts and characterize their business needs independently of any HP competitors. This provided the necessary business and product planning isolation between HP and all other competitors.

During the development process it was a challenge to document and manage the flow of information between all parties. Fig. 2 shows how this was done. Each PC manufacturer was able to negotiate a supply of connectors without disclosing volume, pricing, or new product schedule to potential competitors. There was no exchange of information between connector vendors, and each PC vendor had independent access to the connector manufacturers.

EISA Connector Issues

The key issues surrounding the development of the EISA connector were maintaining ISA electrical and mechanical compatibility at a competitive cost, and excellent market perception for the final product.

Compatibility. The compatibility issue meant that the existing ISA or PC AT boards had to be supported both electrically and mechanically in the new scheme. The new scheme also had to support a new EISA board that used the EISA 32-bit burst mode bus. These constraints caused rejection of solutions that required:

- Increasing the height of the worldwide PC AT product packages by 0.3 inch
- Investigating how many PC AT plug-in cards worldwide have components in the 1/8-inch space above the connector
- Adding the EISA expansion as a separate outrigger or tandem connector.

Electrical Performance. The additional EISA signal lines were specified by the consortium, including power, ground, and spares. This meant adding approximately 90 pins to those already present on the ISA connector. The way in which they were added was important because the goal was not only to provide for the additional EISA pins, but also to improve the RF performance of the ISA section to work with TTL bus logic having typical logic transition times of 2 ns. Improving RF performance meant that the connector impedance had to match the typical multilayer printed circuit board trace impedance of 60 ohms, and multiple-line switching crosstalk to a victim line had to be less than 20% at 2 ns.[4] Crosstalk performance is largely determined by the ratio of the number of signal pins to the number of ground pins and the isolation provided by the EISA printed circuit board ground plane. Therefore, the EISA connector had to have a lower

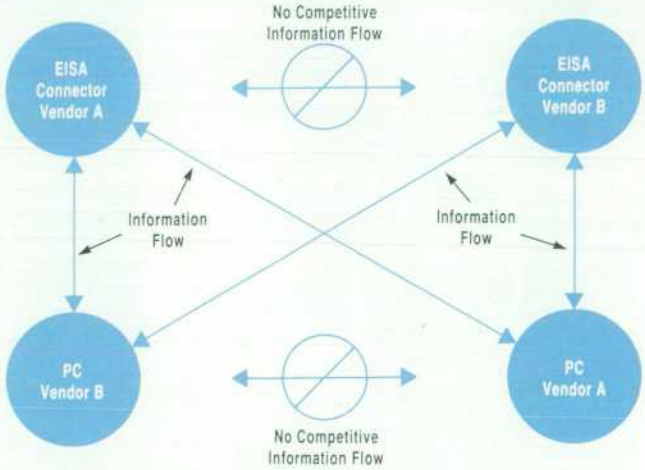


Fig. 2. Information flows between PC manufacturers and connector manufacturers. The goal here was to enforce confidentiality between the connector manufacturers and each PC vendor they worked with.

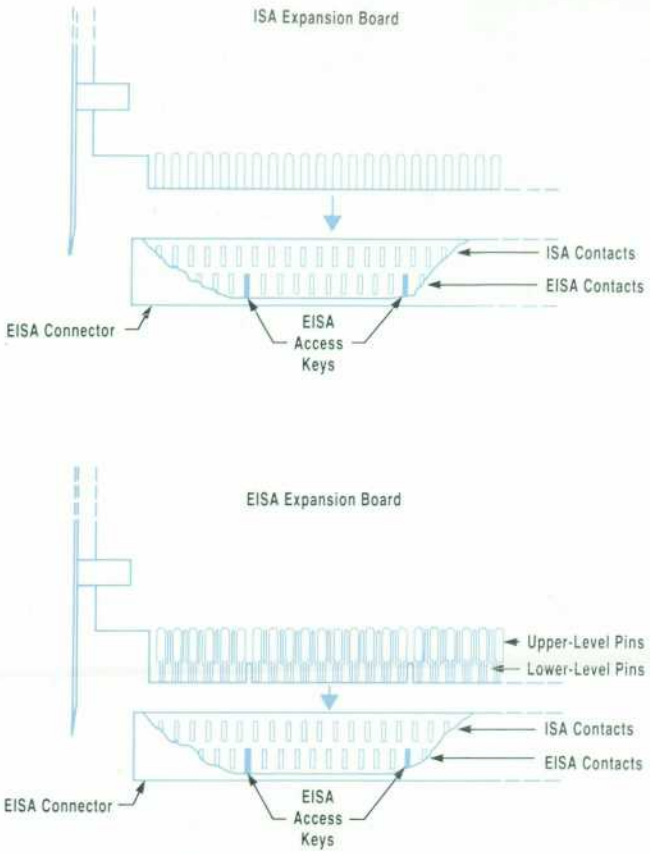


Fig. 3. Portions of ISA and EISA expansion boards, showing the I/O pins on each board and a cutaway view of the EISA connector.

signal-to-ground pin ratio than an ISA bus because the ISA and EISA signals together form a high-performance bus. Ground planes were assumed to be present in the motherboard and EISA printed circuit boards, and the current capacity of the ISA contacts had to be 3A per contact for ISA power pin compatiblity.

Mechanical Performance and Market Perception. A positive public perception was important to the acceptance of the new EISA standard. The connector design needed to maintain the reference features, seating planes, and insertion force of ISA boards. This was key to the overall mechanical design and it also communicated ergonomic backward compatibility to the user. For this reason it was decided that the EISA connector should have the same dimensions as the ISA connector.

EISA Connector Solution

The solution that meets all of the objectives is an extension of an idea used from the very first scheme proposed—the double-decker (or bilevel) connector. Instead of adding the EISA signals in front of, on the side of, or underneath (by increasing the height) of the ISA connector, the additional signals were added below the level of the existing signal pins (see Figs. 3, 4, and 5). Incidentally, this solution was arrived at simultaneously by HP and Burndy Corporation.

At HP this solution evolved from investigating how to add grounds to the ISA connector section for use with EISA cards. It was determined that the additional grounds could be located on a lower level than the ISA contacts. Since the ground contacts had to be as reliable as the signal contacts, the EISA signals were also located on the lower level (see Fig. 3).

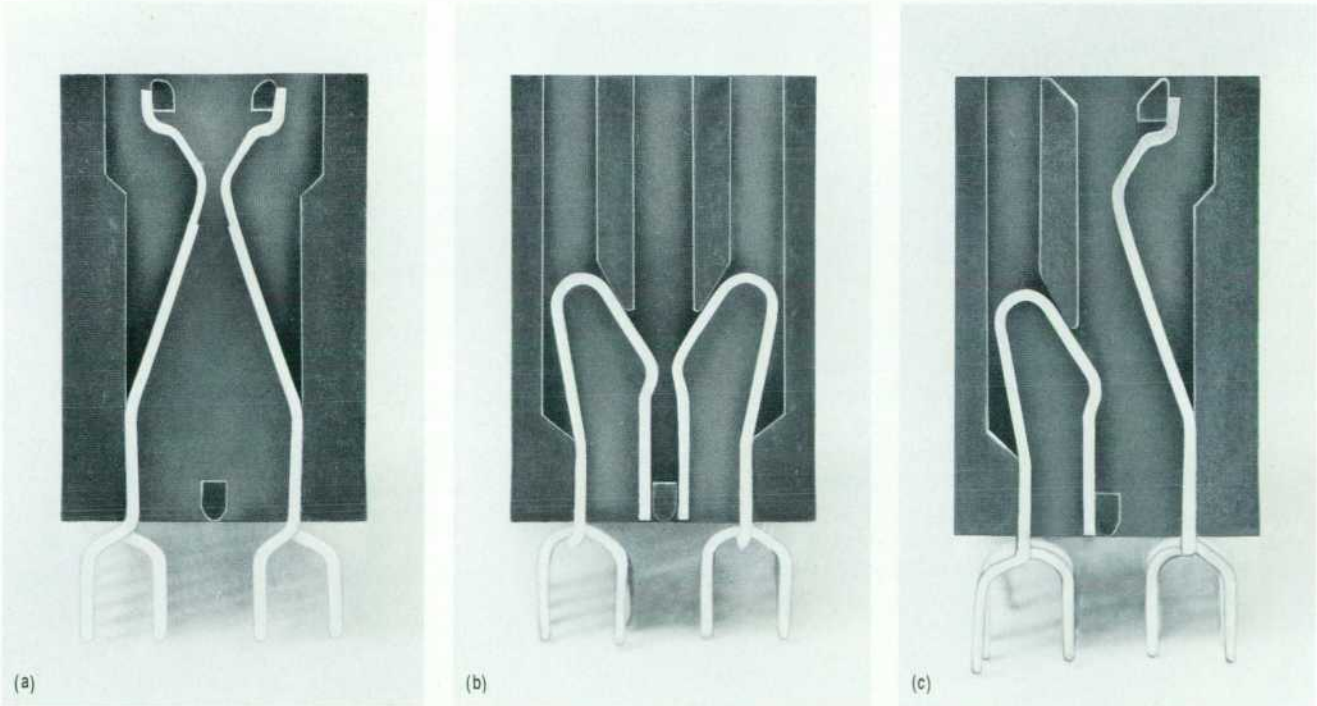


Fig. 4.* (a) Cross-sectional view of the upper-level contact of the EISA connector. (b) Cross-sectional view of the lower-level contact of the EISA connector. (c) Cross-sectional view of both contact levels.

Fig. 5.* Inside view of one half of an EISA connector. The EISA access key prevents ISA boards from being inserted to the depth of the EISA contacts.

*The pictures of the EISA connector sections shown in Figs. 4 and 5 were made from connectors manufactured by Burndy Corporation.

Since this design allows EISA signals (including grounds) in the same motherboard space as an ISA system and the connector remains the same height, the signal-to-ground pin ratio for the ISA signals is effectively reduced to 8:1. Improved isolation for the 8.33-MHz BCLK (backplane clock) is provided by adjacent RF grounds. Two of the ground pins are at BCLK so that the gold finger pads are on opposite sides of the printed circuit board. Thus these pins can be directly connected to the plug-in board ground plane with a low-inductance connection.

In addition, the internal ground planes of the plug-in board under the gold fingers, which play a key role in determining overall connector electrical performance, can extend almost to the surface of the motherboard. These help provide electrical isolation between the two halves of the connector, single-line crosstalk between adjacent pins of 5% to 7% at 1-ns edge transition times, and a controlled 55-ohm to 65-ohm signal impedance.[4] An added benefit of the dual-level contact structure is that although the number of contacts doubled, the insertion force only increased from 28 pounds for the ISA connector to 35 pounds for the EISA connector. The signal density of each level is the same as the ISA connectors (20 per inch), thereby minimizing the impact on printed circuit board manufacturing requirements.

Conclusion

Through a joint effort with other members of the EISA consortium, we designed a connector that meets all the technical design requirements necessary for industry acceptance. Given the number of companies and parties involved, we achieved an extremely fast development cycle of six months from start of this project to the production of the first connectors.

Acknowledgments

We would like to thank the HP team consisting of Bob Johnson, Peter Guckenheimer, Geoff Moyer, Carl Steele, Guy Gladden, Bob Campbell, Kim Tanaka, and John Neuneker. We should also recognize the effort and contributions of the Compaq design team, and the engineers of Amp Inc. and the Burndy Corporation.

References

1. *Extended Industry Standard Architecture*, Revision 3.10, 1989, BCPR Services, Inc.
2. L. B. Glass, "Inside EISA," *BYTE*, November 1989, pp. 417-425.
3. T. Dowden, *Inside the EISA Computers*, Addison Wesley, 1990.
4. *AMP Designer Digest no. 53*, AMP incorporated., pp. 6-8.

The HP Vectra 486 Memory Controller

The memory subsystem architecture and the memory controller in the HP Vectra 486 personal computer provide a high-performance burst-mode capability.

by Marilyn J. Lang and Gary W. Lum

During the investigation phase for the HP Vectra 486 personal computer, in-house performance tools confirmed that the memory system was a key to overall system performance (see article on page 92). Selecting an optimal memory and controller architecture for a high-performance memory subsystem was a major design consideration for the HP Vectra 486 design team.

While performance was considered important to the success of the HP Vectra 486, it was but one of many important factors to consider for the memory controller design. The PC server market (a target for the HP Vectra 486) continues to demand more memory, yet entry level systems require a small starting memory and incremental memory size. There is also an emerging need to simplify the installation and configuration of memory by both customers and dealers. We were also anticipating future Intel486 microprocessor speed upgrades, and wanted a memory architecture that could support these upgrades with minimal changes. And, of course, we were striving to deliver, at a competitive price, a system that included the EISA standard.

From these requirements, the memory controller objectives became the desire to:

- Meet the HP Vectra 486 schedule and cost structure
- Provide competitive performance for 25-MHz systems
- Have a large and logical memory upgrade scheme
- Provide a design for supporting higher-speed Vectra 486 systems.

With these objectives, the design team began investigating relevant technologies that would help determine the optimal feature set. Three main areas were focused on: the Intel486's burst-mode capability, the 4M-bit DRAM, and the emerging 36-bit SIMM (single in-line memory module) standard for PCs.

Investigations

The Intel486, with its on-board 8K-byte cache, uses burst mode to fill a cache line from an external memory system. Burst mode, long used in larger computer systems but new to personal computers, is a more efficient method of transferring data. Rather than transferring only a single piece of data for each address generated, burst mode allows multiple pieces of data (typically four dwords*) to be transferred for each address. Since subsequent addresses need not be generated, fewer cycles are required to move information, and bandwidth increases. Supporting burst mode, on the other hand, requires more complexity than traditional memory or cache controllers.

Using our available performance tools, the Intel486 burst-mode capability was matched with various memory architectures, ranging from a simple, single-bank memory array to a cached, multiple-bank configuration. The single-bank memory array was quickly dropped, because it was not a competitive solution. The key finding from this analysis was that for 25-MHz systems, by using the burst-mode capability in the Intel486, a DRAM memory controller communicating directly to the Intel486 could compare quite favorably with a moderately sized external memory cache. This was particularly true for cache controllers that only supported burst mode between the Intel486 and the cache (or did not support burst mode at all). When the cost of the cache was factored in, the interleaved, bursting memory controller was the clear preference for the Vectra 486.

The 4M-bit DRAM was scheduled for production about the same time the Vectra 486 was to be released. Although the 4M-bit DRAM would provide the highest memory density available, it was considerably more costly than the 1M-bit DRAM, which had been in production for several years. Being able to support both densities would allow us to exploit both the 1M-bit and 4M-bit advantages. Standard memory configurations could be built with the cost-effective 1M-bit DRAMs, while large memory arrays could use the 4M-bit. Furthermore, as the 4M-bit DRAM progressed down the production cost curve, we could move quickly to it when prices became attractive. By working closely with some of our key memory vendors, we were able to secure prototype and production volumes of 4M-bit DRAMs for the Intel486.

Previous HP personal computers had used SIMMs, and the general feedback from our customers and dealers was very positive. A SIMM is a small printed circuit board with memory installed on it (typically surface mounted). An edge connector on the SIMM allows a customer to install it easily into an available connector. The typical SIMM organization is nine bits wide (eight data bits and a parity bit) and the edge connector has 26 pins. During Intel486 development a new SIMM organization was beginning to get attention—36 bits wide with a 72-pin edge connector—which allows a full dword (32 bits plus parity) to be on a single SIMM. This SIMM also supports presence detect, which encodes the size and speed of the module on four of the 72 bits, and allows the module

*32 bits

characteristics to be read directly from the SIMM. The new SIMM was already available in 1M-byte and 2M-byte densities. Both densities use 1M-bit and 256K-bit DRAMs, but at the time none used the 4M-bit DRAM. Working with our key memory vendors, we were able to establish standard 4M-byte and 8M-byte SIMMs.

From these investigations and other discussions, the Intel486 memory controller feature set was defined to include:

- Intel486 burst-mode support
- 2M-byte to 64M-byte memory array size
- Minimum memory upgrade size of 2M-bytes
- Support for 1M-byte, 2M-byte, 4M-byte, and 8M-byte SIMMs
- Support for shadowing or remapping of 16K-byte memory blocks
- Full support for EISA devices, including bus masters.

Since many of the features we wanted to include involved new technologies, no commercial memory controllers were available that supported our feature set. Furthermore, a short investigation concluded that using an existing memory controller with additional surrounding logic to support the new features would not meet our cost or performance goals. We decided that the best design approach was to develop a new controller using an ASIC to implement the memory controller.

Memory System Architecture

The memory system is completely contained on a 5.6-inch by-13.3-inch memory board, and uses a proprietary connector on the Vectra 486 motherboard. The memory system sits directly on the 25-MHz Intel486 bus.

Allocating board space for the memory controller, the DRAM drivers, and other support logic, a maximum of eight SIMMs can be accomodated on the board. When populated with 8M-byte SIMMs, this allows a maximum memory size of 64M bytes. This is four times what previous HP personal computers had supported.

In burst-mode operations, the Intel486 is capable of accepting one dword each processor clock cycle. At 25 MHz, this means an ideal memory system would be able to deliver one dword every 40 ns. Since we were using 80-ns DRAMs, a simple 32-bit memory array was clearly not sufficient to meet our performance goals. Two possible architectures were investigated: a 128-bit-wide memory array and a 64-bit-wide memory array. With a 128-bit memory array, all four dwords would be fetched on the initial Intel486 memory access, and one dword output on each of the four clock cycles. For the 64-bit memory array, two dwords would be fetched using the Intel486-generated address, and two more dwords fetched using an address generated by the memory controller. The additional address generation requires another clock cycle, so the 64-bit memory array provides four dwords in five clocks, rather than four clocks. Although this was slower than ideal, the 64-bit-wide memory system allowed a minimum system configuration and upgrade increment of 2M bytes, rather than the 4M bytes required in the 128-bit architecture. We decided the 64-bit-wide memory array provided the best overall solution for the Vectra 486.

Fig. 1 shows the block diagram of the Vectra 486 memory system. The 36-bit SIMMs are organized in pairs, creating the 64-bit-wide memory array. SIMMs 1, 3, 5, and 7 contain the lower-order dword, while SIMMs 2, 4, 6, and 8 contain the higher-order dword. Each SIMM pair must be of the same SIMM density, but different density pairs are allowed in the memory array. The memory array is fur-

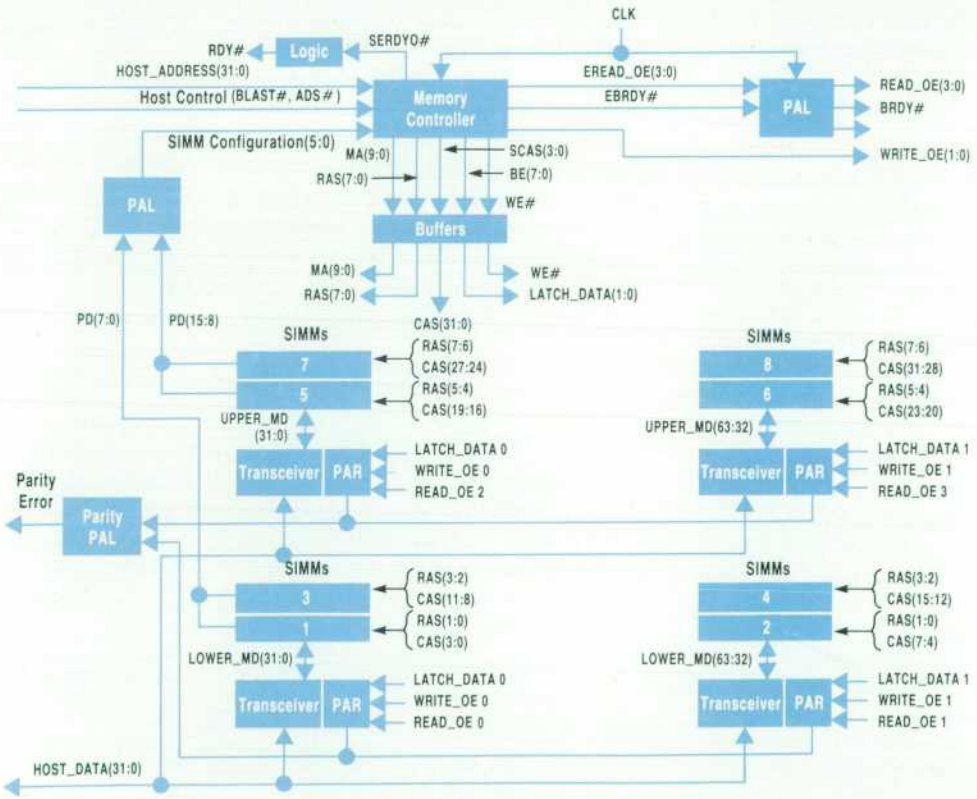


Fig. 1. The Vectra 486 memory subsystem.

ther divided into upper and lower memory halves (UPPER_MD and LOWER_MD) to reduce the maximum capacitance on each memory data line. Although this increased part count on the board and loading on the system host bus, it improved timing margins in the the most critical system timing paths.

Data transceivers are used to move data between the Intel486 and the memory array, and sit directly on the system host data bus (HOSTDATA(31:0)). Since the 64-bit memory system requires two memory accesses for each Intel486 burst access, latching data transceivers are used to output data from the first fetch while the second 64 bits are read.

The generation of memory addresses and control signals by the memory controller is complicated by the organization of the SIMMs. The 1M-byte and 4M-byte SIMMs are organized as a single block of memory (or memory bank), 256K deep by 36 bits wide and 1M deep by 36 bits wide respectively. Each memory bank has one row address strobe and four column address strobes (one for each byte). The 2M-byte and 4M-byte SIMMs, however, are organized as two banks of memory. The 2M-byte SIMM contains two 1M-byte banks, and the 8M-byte SIMM contains two 4M-byte banks. These two-bank SIMMs have two row address strobes (one per bank) and four shared column address strobes (to select one of four bytes in both banks). A SIMM socket can contain either a one-bank or a two-bank SIMM.

To correctly control the one-bank or two-bank SIMMs, the memory controller generates row address strobes and row addresses to the array based on the memory bank configuration. Each SIMM pair contains either one or two banks, depending on the SIMM installed. Eight row address strobes (RAS(7:0)) are generated directly from the memory controller, two for every SIMM pair. For a 2M-byte or 8M-byte SIMM the memory controller uses both row address strobes. For a 1M-byte or 4M-byte SIMM only one address strobe is used. The row address appears on MA(9:0) when the row address strobe goes active.

The memory controller also takes advantage of the page mode capability of the SIMMs, and keeps the row address strobe asserted in each memory bank. If a subsequent memory access falls within an active page (has the same row address as a previous access to the bank), the much faster page mode access is performed.

The column address strobe and column addresses to the array are generated from the four column address strobes from the memory controller (SCAS(3:0)), providing one strobe per SIMM pair. Because the Intel486 can operate on a single byte of data, each byte in the array is made individually accessible. Each SIMM has four column address strobes, so 32 strobes (CAS(31:0)) are generated for the eight SIMMs by combining SCAS(3:0) with eight byte enable signals (BE(7:0)). BE(7:0) is also used to generate the direction controls (READ_OE and WRITE_OE) and latch signal (LATCH_DATA) to the data tranceivers.

Parity is also handled on a byte basis. Because of memory controller pinout and timing, parity generation and detection are implemented using PALs and random logic. Another PAL is used as a SIMM presence detect encoder, which reads four presence detect (PD) bits from the first SIMM of each pair and encodes them into six SIMM_CONFIGURATION bits. This encoding specifies several different possible memory configurations, including combinations of 1M-byte and 4M-byte SIMMs, or 2M-byte and 8M-byte SIMMs. When used with the EISA configuration utility, the presence detect capability allows the user to configure memory from the screen.

To accommodate the Intel486's 33-MHz timing (which was not available during the design phase of the project), the READ_OE signals to the data tranceivers are generated one clock early and pipelined through an external registered PAL. This scheme ensured that the read path was as fast as possible. It also gave us some flexibility in host bus timing, in case of changes in CPU timing.

Memory Controller Architecture

Fig. 2 shows a block diagram of the Vectra 486 memory controller. There are seven major blocks in the memory controller. The configuration registers contain address range, remap and shadow regions, and other memory configuration information typically set by the BIOS at power-on (see the article on page 83). The 8-bit XD bus, a data bus available on all PCs, is used to access all memory controller registers because fast access is not a high priority at power-on time.

The memory configuration information, along with the SIMM configuration information from the presence detect pins on each pair of SIMMs, is used by the address block to determine if the current memory cycle on the host address bus is in the memory controller's address range. If it is, the address block will also determine which memory bank is selected, whether it is a page hit or miss (whether the current row address is the same as an active page), and the appropriate DRAM row and column addresses (MA(9:0)).

Memory cycles that appear on the host bus are generated either from the CPU or from a backplane device such as an EISA bus master. Two independent state machines, the CPU state machine and the EISA/ISA/Refresh state machine, monitor the state of each device. The CPU state machine is actually two interlocked state machines. One machine monitors the host bus and when it sees a memory request, it starts a second state machine. The second machine generates the appropriate CPU_CYCLE_CNTL signals (page hit or miss, dword write, or one, two, or four dword read). The CPU state machine is fully synchronous with the Intel486 processor clock.

The EISA/ISA/Refresh state machine generates control signals for all other cycles. This machine supports EISA burst read or write cycles, EISA- and ISA-compatible DRAM refresh, and all ISA cycles. Because ISA is an asynchronous bus, the EISA/ISA/Refresh state machine is a semi-synchronous state machine, and uses BCLK (the backplane clock), and external delay lines to generate the BACKPLANE_CYCLE_CNTL signals.

The CPU_CYCLE_CNTL and BACKPLANE_CYCLE_CNTL signals are generated on every memory cycle. Each set of signals includes the DRAM timing relationships that optimize the respective CPU or backplane device bus cycle. HLDA (hold

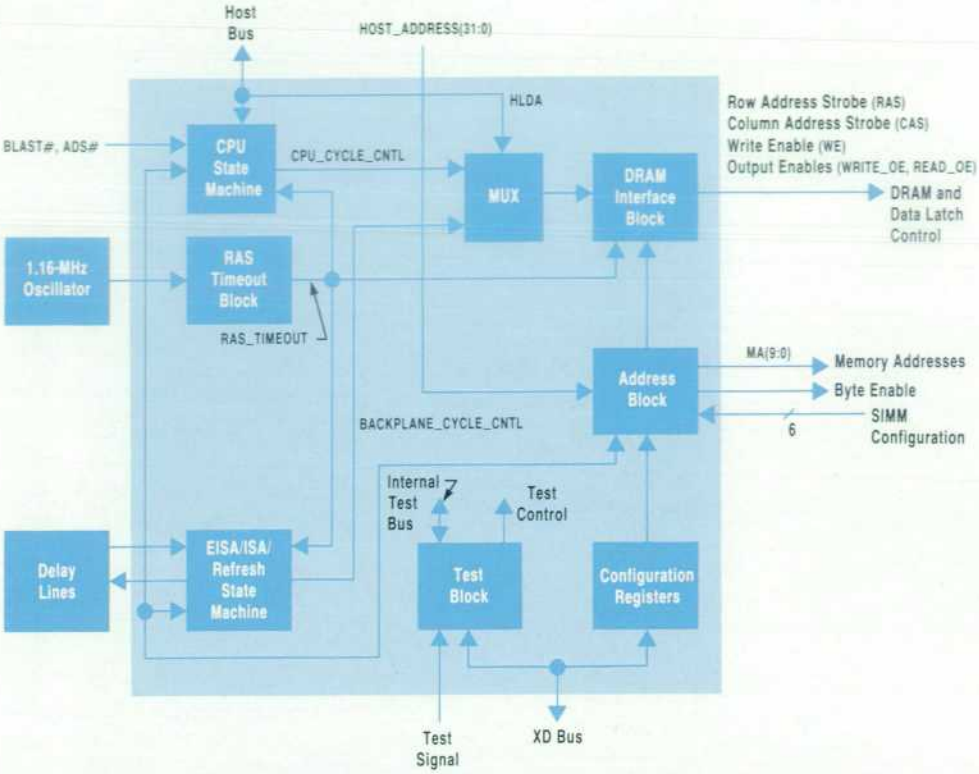


Fig. 2. The Vectra 486 memory controller.

acknowledge) is used as the select signal to a multiplexer to determine the correct set of signals. Once the correct CYCLE_CNTL is selected, the corresponding DRAM control signals RAS, CAS, and WE are generated for each bank via the DRAM interface block. The byte, word, and dword addressability of the memory array is also handled by the DRAM interface block, which generates the appropriate data transceiver control signals (READ_OE and WRITE_OE). For the Vectra 486, all memory reads are 64 bits while memory writes can be one byte, one word (two bytes), or one dword (four bytes).

The row address strobe timeout clock is used for DRAM timing. The maximum time a page can be open (RAS active) is 10 μs. Since it is possible to exceed this limit during an EISA burst cycle, continuous page hits, or a long Intel486 idle time, it is necessary to monitor the time each bank is active. Eight timeout counters, one for each bank, monitor the active page time. Counters are enabled when the row address strobe is active, reset when the row address strobe goes inactive, and clocked by an external 1.16-MHz oscillator. When the timeout limit is reached, RAS_TIMEOUT is generated. The CPU state machine and the EISA/ISA/Refresh state machine will then finish the current memory cycle and allow the DRAM interface block to disable the timed-out DRAM page. In some instances it is possible to disable a page without incurring any clock penalties because a page hit on one bank can be done while turning off a timed-out bank.

The test block is used to debug and test the memory controller chip. An external test pin puts the memory controller into the test mode. In test mode, external address lines are used to select which signals and state machine states are put on the internal test bus. The internal test bus contents are available via the XD bus.

Burst Mode Read

All Intel486 memory requests are initiated by placing the memory address on the host address bus, setting appropriate control lines (i.e. memory read or write) and strobing ADS#. Fig. 3 shows some of the key timing for a burst-mode read cycle for four dwords. One of the control lines, BLAST# (burst last) is asserted if the Intel486 requests a burst-mode cycle. If the memory system is incapable of supporting burst mode, it will return a single dword and assert RDY# (ready). If the memory system can support burst mode, it will assert BRDY# (burst ready) and return two or four dwords depending on the type of Intel486 request. The Intel486-generated memory address is used to fetch the first two dwords, and a second address (incremented by two dwords) is generated by the memory controller to complete the four-dword burst read.

Returning a burst-mode request entails several operations within the memory system. For simplicity, we assume a DRAM page hit (for a page miss, additional cycles are required to generate a row address strobe and row address). When the Intel486 requests a burst cycle, it will output an address for each of the four dwords in the burst. These addresses (and respective data) follow a particular sequence, depending on the initial address supplied by the Intel486. The memory controller uses only the initial address because the subsequent addresses from the Intel486 would not meet our system timing. The memory controller will latch the initial address and generate the identical sequence earlier in the burst cycle.

There are four possible address sequences, determined by the state of HOST_ADDRESS(3:2):

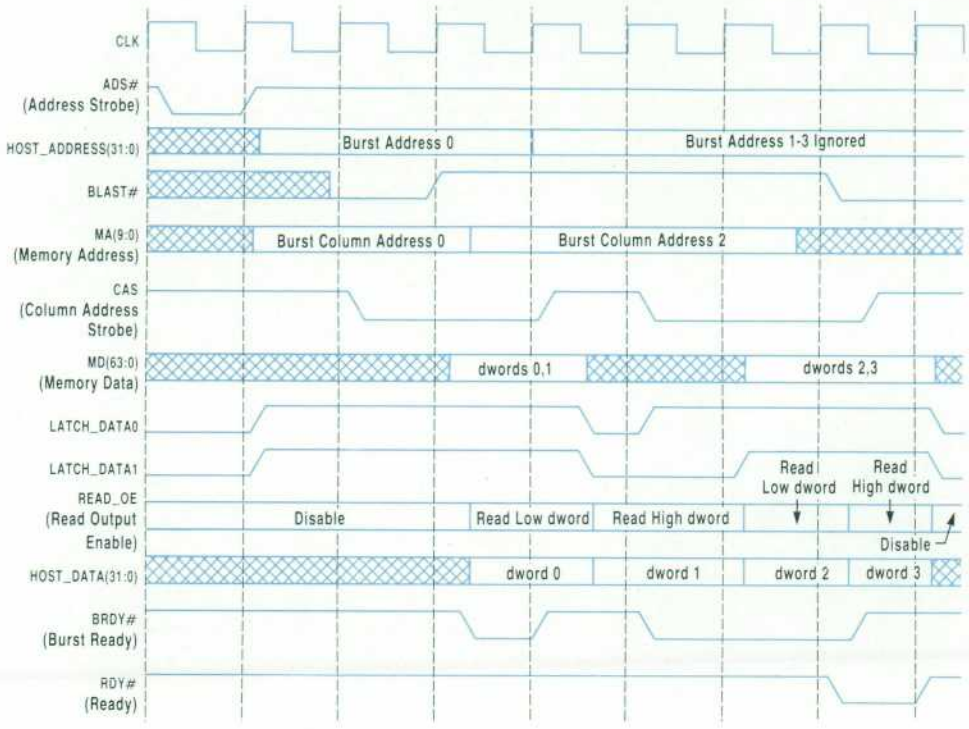


Fig. 3. The timing relationships between the signals involved in doing a burst read hit of four dwords.

Address Sequence	dword 0 address	dword 1 address	dword 2 address	dword 3 address
1:	xx 00	xx 01	xx 10	xx 11
2:	xx 01	xx 00	xx 11	xx 10
3:	xx 10	xx 11	xx 00	xx 01
4:	xx 11	xx 10	xx 01	xx 00

xx = HOST_ADDRESS(31:4)

00, 01, 10, 11, = HOST_ADDRESS(3:2) or A3 A2

The memory controller will generate the correct address sequence by toggling A2 on each dword. The third and fourth dwords differ in A3, so the second memory read has a column address that differs from the first only in one bit.

To improve burst-mode timing, rather than waiting for BLAST# to be asserted (which may come relatively late in the cycle), the memory controller assumes every memory read is a burst-mode read, and begins generating CAS, READ_OE and BRDY# signals. The memory controller will return BRDY# with the first dword of every read cycle. The memory controller will then use BLAST# (now valid) to determine if the request was for a burst read. If it was not, a RDY# will be generated, the second dword read ignored, and the cycle terminated. If it is a burst read, then CAS is precharged in preparation for a second memory read, the first and second dwords are latched in the data transceivers, and the second dword is output. BRDY# is returned for the second dword on the next clock cycle, at which time the second memory read begins and the first data latch is opened to receive data for the third dword.

One clock later, both data latches are open, and the third and fourth dwords are put on the host data bus in consecutive clock cycles. The memory controller completes the burst-mode read by generating a SERDYO# (shared early ready) signal. This signal is input to a logic block in the Vectra 486 memory subsystem which forms the RDY# signal to the Intel486 (see Fig. 1). In the Intel486 a burst mode read cannot be prematurely terminated, so once a burst sequence has started, all four dwords must be read.

Conclusion

The memory controller design began at the same time as the HP Vectra 486 SPU (system processing unit), and remained the critical path component for most of the development schedule. The project team successfully met the HP Vectra 486 schedule objective by delivering a fully functional first-pass memory controller chip. This chip revision was used for the HP Vectra 486/25T production until introduction of the HP Vectra 486/33T memory controller version. Fig. 4 shows one of the memory bench-

	System External Cache Size			
	Vectra 486 None	Vendor A 128K-Byte Cache	Vendor B 64K-Byte Cache	Vendor C 128K-Byte Cache
Lotus Benchmark (Relative Performance)	1.00	1.03	.97	1.03
Integer Sort (K Stones)	778.89	763.55	782.83	828.88

Fig. 4. Memory benchmarks run on the Vectra 486 and other cached Intel486 25-MHz machines.

marks run on the HP Vectra 486 and other cached 25-MHz Intel486-based machines.

Acknowledgments

Key to the success of the HP Vectra 486 memory controller were the other members of the design team: Sridhar Begur, Stuart Siu and Deepti Menon. Wes Stelter was responsible for the memory board design, and provided much assistance during initial chip debug. Carol Bassett led the vendor selection investigation and the writing of the data sheet. Bob Campbell contributed to the initial architecture, while Mark Brown provided project management during the initial definition and architecture phase. Wang Li and the HP Circuit Technology Group deserve special recognition for their execution and delivery of prototype and production chips.

Bibliography

1. *i486 Microprocessor Data Book*, Intel Corporation, 1991.
2. *82350 EISA Chipset Data Sheet*, Intel Corporation, 1989.

The HP Vectra 486 Basic I/O System

An Intel486 processor, the EISA bus standard, and a new memory subsystem all required enhancements to the Basic I/O System to ensure that the HP Vectra 486 made the best possible use of these new features.

by Viswanathan S. Narayanan, Thomas Tom, Irvin R. Jones Jr., Philip Garcia, and Christophe Grosthor

The Basic I/O System (BIOS) is the lowest-level software interface between the hardware and the operating system in the HP Vectra 486 personal computer. The BIOS consists of a power-on self-test and function support for the DOS operating system. The power-on self-test performs testing and initialization of the various components of the system and loads the operating system. The rest of the BIOS supports functions to access the various DOS devices. This article describes the development process and the features incorporated into the HP Vectra 486 BIOS to support the Intel486 microprocessor and the Extended Industry Standard Architecture (EISA).

BIOS Source Base

The Vectra 486 BIOS code was heavily leveraged from the source code of the Vectra ES, RS, and QS personal computer series, which support the HP-HIL (human interface link) BIOS extensions (EXBIOS). The EXBIOS support was stripped off and support for EISA, the micro-DIN mouse, and other enhancements were added to create the Vectra 486 BIOS (see Fig. 1).

To maximize BIOS leverage for future systems, team members focused on keeping a large part of the new source files reusable. A common collection of reusable software modules ensures a more compatible and easily upgradable software system. This commonality ensures that during development, potential compatibility problems only have to be addressed once, and when a compatibility problem in a released product is fixed in a common routine, the fix is done once and automatically goes into all subsequent software releases.

The BIOS development of code was shared between the engineers at HP's California Personal Computer Division in Sunnyvale, California and HP's Grenoble Personal Computer Division in France. The configuration for transferring files back and forth between the two groups is

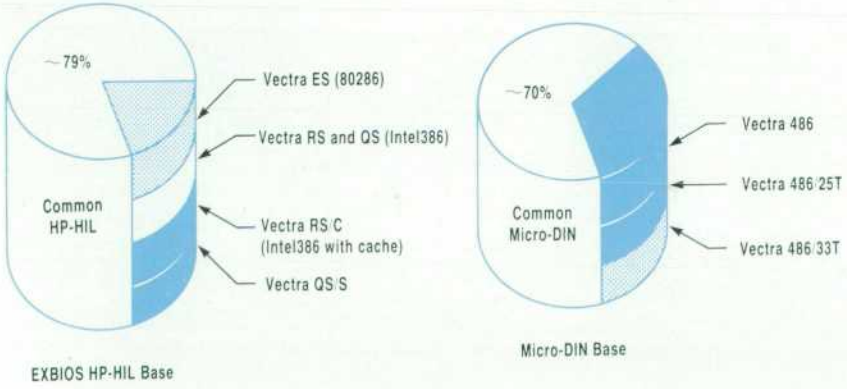


Fig. 1. HP Vectra BIOS source code bases.

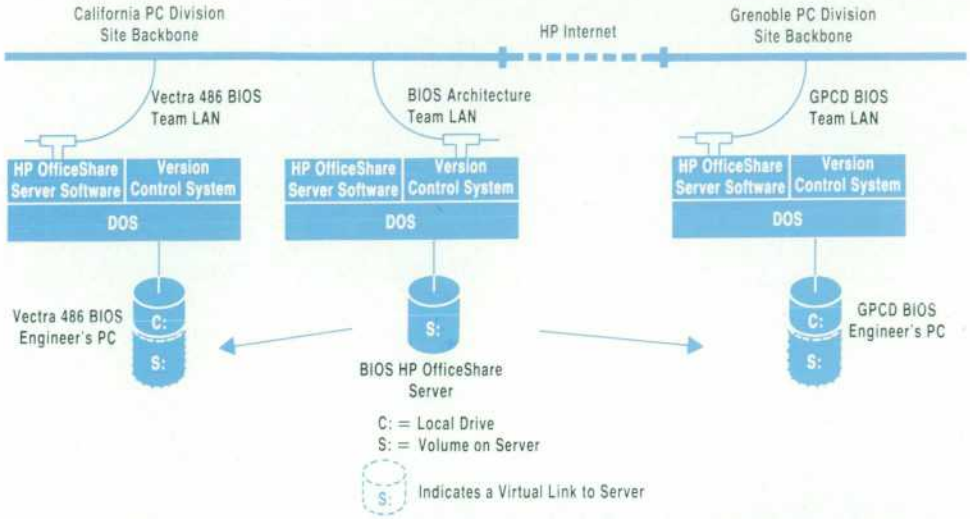


Fig. 2. Communication network between the BIOS engineers in California and the BIOS engineers in Grenoble, France.

shown in Fig. 2. This code sharing created issues related to ensuring file security and tracking changes to the code. For this reason the BIOS source base is managed by a software revision control system. The source files are structured into common and machine-specific directories. The machine-specific files contain code that handles the initialization requirements of different chip sets and different processors and processor speeds. EISA and ISA differences are also handled by the code in these files.

EISA Initialization

One of the most important features of the EISA architecture is its ability to detect the I/O expansion boards inserted in the system's motherboard slots. The configuration utility easy config generates information for each EISA or ISA card installed in a system expansion card slot. When the user is satisfied with the system configuration with either the defaults presented by easy config or after making any desired changes, the configuration is stored in nonviolate RAM.

The configuration files for each board contain function data structures for each slot that provide information on the DMA initialization, IRQ (interrupt request) trigger, memory information, and I/O initialization. easy config resolves I/O initialization, memory conflicts, and identification for the individual expansion boards in each slot.

EISA Configuration Support

Support for storage and retrieval of EISA configuration information is provided by 8K bytes of nonvolatile RAM and by system BIOS support routines. The EISA configuration utility easy config uses these routines to clear nonvolatile RAM, store EISA configuration information (on a slot-by-slot basis), and retrieve information for all functions of a slot (brief format) or for one function (detailed format). Fig. 3 shows some of the processes involved in retrieving data from or storing data to the nonvolatile RAM containing configuration data.

The system BIOS power-on software also retrieves the configuration data to initialize the hardware in each slot. After the system boots, other system drivers, utilities, or the operating system may also store and/or retrieve configuration data (or any other data) from nonvolatile RAM.

To accommodate various operating environments the BIOS routines that interface to the nonvolatile RAM can operate in the Intel486's real or protected modes. In real mode, 16-bit segments and offsets are used to address a 1M-byte address space. In protected mode, segment registers become selectors into descriptor tables which with offsets allow for 16-bit to 32-bit addressing (up to 4 gigabytes).

To save space, input data is compressed by the caller before it is stored in nonvolatile RAM by the BIOS routines. When configuration data is retrieved from memory it is expanded by the BIOS routines before being passed to the caller. Expanding the output data involves padding variable-length data fields and blocks into fixed lengths. Slot configuration data consists of a variable number of variable-length function blocks that describe each function of a card in an EISA or ISA slot. The function blocks consist of fixed and variable-length fields and variable repetitions of fixed and variable-length subfields. These fields consist of descriptive text information and memory, interrupt, DMA, and I/O resource and configuration data. Free-form data can also be stored in some of these fields. The slot configuration data is stored sequentially by slot

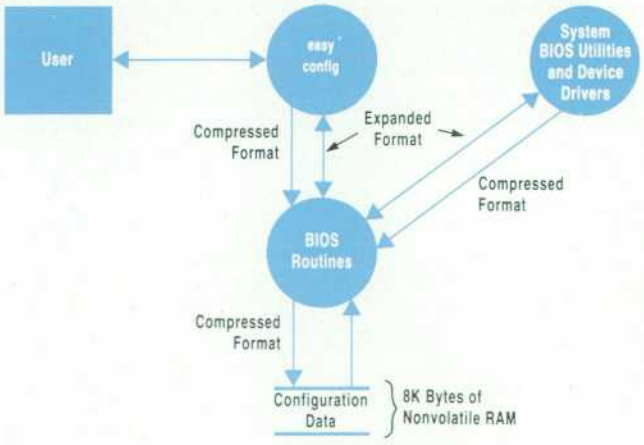


Fig. 3. Storing and retrieving configuration data to and from the nonvolatile RAM. Data is compressed when it is placed in memory and expanded when it is retrieved.

number (including empty slots) until the last physical or virtual slot is reached. The minimum size of a slot's configuration data is zero (empty) and the maximum size can be as long as the remaining available space in nonvolatile RAM.

To access nonvolatile RAM data efficiently (in terms of speed and space), a table approach is used. A table of pointers that point to slot configuration data blocks is allocated dynamically and grows inward from one end of the nonvolatile RAM. The data space for slot configuration blocks is also allocated dynamically and also grows inward but from the opposite end of nonvolatile RAM (see Fig. 4).When the pointer table and data space meet, the nonvolatile RAM is full. This technique saves memory space and allows for a single look-up to reach any data block.

Power-on Initialization

When the system is rebooted the BIOS initializes one EISA or ISA slot at a time and one function at a time using the configuration information stored in nonvolatile RAM. The initialization proceeds in two steps; error checking is performed first and then the slot is initialized.

Error Checking. The system ROM BIOS begins the initialization only if the nonvolatile memory's checksum is good. The BIOS also has to check whether the correct card is installed in the right slot before it initializes the card in that slot. The BIOS checks for the following combinations in each slot.

- A slot could be defined as empty according to the configuration data, but the user may have plugged a card into the slot.
- A slot could be defined to have a particular identifier according to the configuration data but may be read as empty.
- A slot could be defined to have no readable identifier according to the configuration data but BIOS reads an identifier from the slot.
- An identifier read from the slot may not agree with the identifier in the configuration data.

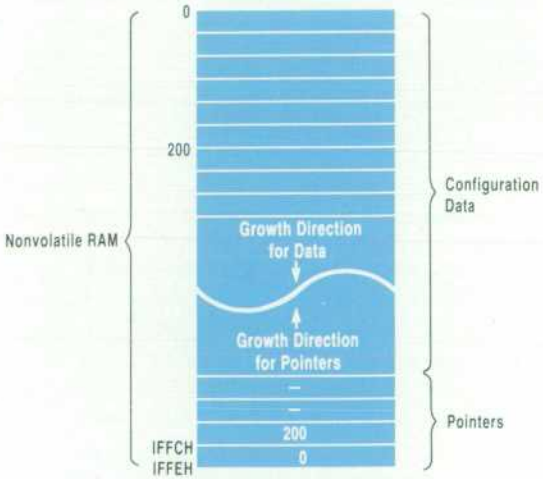


Fig. 4. Organization of pointers and slot configuration data in nonvolatile RAM.

An identifier for a slot is checked by reading certain slot-specific I/O ports as defined by the EISA specifications. After verifying that the slot that needs to be initialized has the correct card in it, the BIOS start the initialization for that slot. Fig. 5 shows the error checking process performed by the BIOS during initialization.

Slot Initialization. As in error checking, slot initialization data comes from the configuration data in nonvolatile memory. The configuration data for a slot is retrieved as a block of data, and there could be many blocks of data for a particular slot. Fig. 6 shows the flow for slot initialization.

Slot initialization starts with the BIOS code reading a block of data from nonviolatile memory for a particular slot. It checks to see if there are any DMA initializations for that slot. If DMA is not shared, then the BIOS initializes the extended DMA registers defined for that slot. Next the code checks to see if the slot has any IRQs that need to be set as edge- or level-triggered. It then sets up the cache map for noncacheable regions as defined for that slot. The code then continues with the I/O initializations if any. Once this sequence is complete the code continues with the next function for the slot until all the functions are completed for that slot.

The BIOS provides a feature that allows the user to make blocks of memory cacheable or not. This is very useful for boards that have memory-mapped I/O. The BIOS builds a cache map in which each bit defines the cache on/off state of a particular segment (each segment is 64K bytes). A function in the configuration information for a slot can define the start address of the memory and the length of memory for which caching needs to be turned on or off. The BIOS initially sets all segments' caching to be on. It then checks for segments of memory for which the caching needs to be turned off and then turns caching off for the segments that are within the memory length specified. The cache map is updated and is later used in the boot process to initialize a 64K-bit static RAM, which the hardware uses in its cache on/off logic. Each bit of the static RAM represents a segment, allowing 64K segments (or four gigabytes) to be represented (see Fig. 7).

The BIOS then initializes the various I/O ports as defined in the configuration data. The I/O can be 8-bit, 16-bit, or 32-bit reads or writes. The configuration data also defines the mask for the particular I/O port. Thus, the I/O port is read, the data is masked (ANDed) with the mask value, ORed with the bits that need to be set, and written back to the I/O port.

Finally the BIOS enables the board in the initialized slot. Any time the initialization fails, the BIOS makes sure that the system can boot from a flexible disk and that the video is initialized correctly. This is done so that the machine is in a minimum working state so that the user can execute easy config and reconfigure the system.

Variable Speed Control

For backwards compatibility, it is sometimes necessary to reduce the speed of the PC. This is particularly true for copy-protected software applications that are speed sensi-

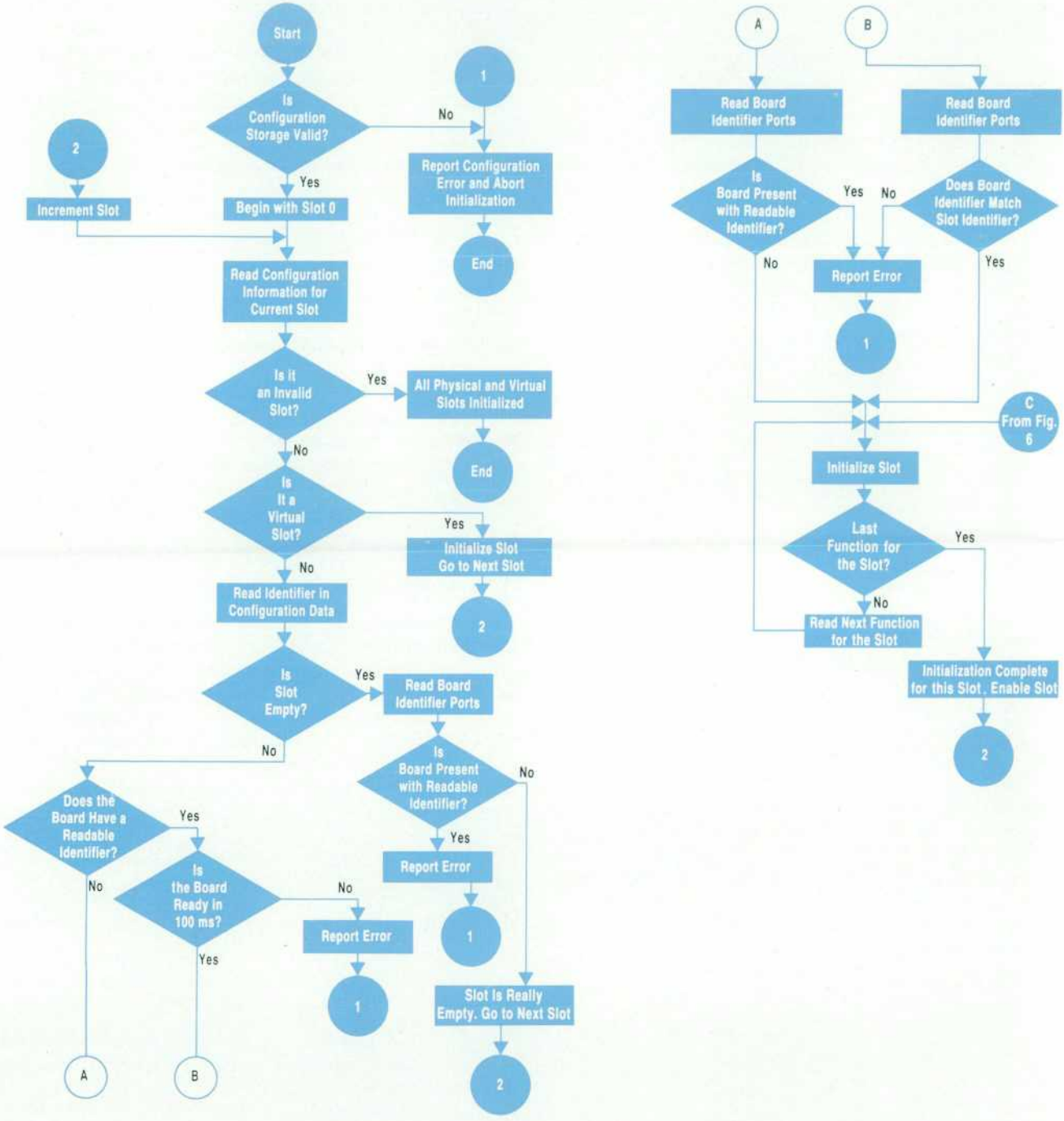


Fig. 5. Error checking during power-on initialization.

tive. The Vectra 486 can reduce its speed for all operations, or only for flexible disk operations. The system BIOS is responsible for this control. To change speeds the BIOS programs the duty cycle of a square wave generated by a hardware timer which modulates the Spd_Hold_Req (hold request) input of the microprocessor (see Fig. 8a).

If the microprocessor did not have an internal cache then it would effectively be idle while it relinquishes the bus during a hold request. Its effective speed would thereby be reduced by the modulation factor. Since the Intel486 has an internal cache it will continue execution, even when in a Hold state, until a cache miss occurs, when it must wait for the bus. Therefore, to control the microprocessor's effective speed accurately when it is reduced from its maximum (unmodulated) value, it is necessary to disable and flush the processor's internal cache. With its internal cache empty, the processor will halt execution (because of cache misses) until the modulated Spd_Hold_Req signal is deasserted. The BIOS programs an I/O port which disables and flushes the internal cache via the Intel486 control lines. This avoids having to use the Intel486 control registers to disable the cache. These con-

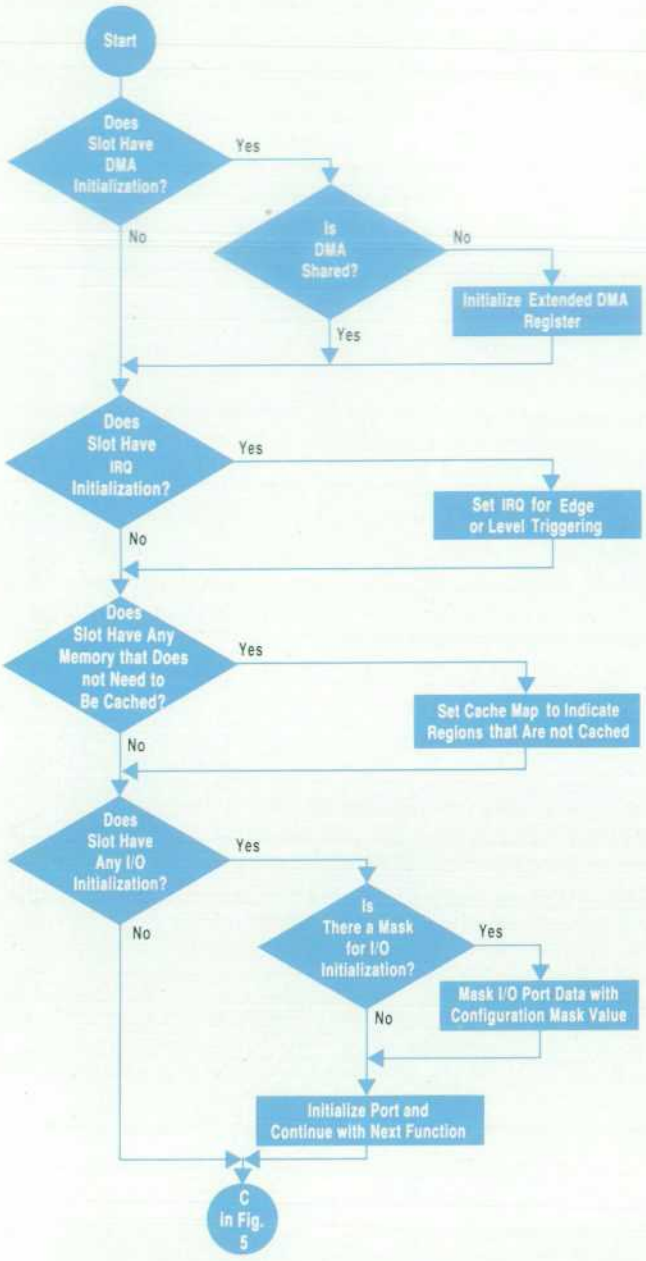


Fig. 6. Flow for slot initialization.

trol registers could be in use by other software applications and might be disrupted by the actions of the BIOS.

If the speed is restored, or after a flexible disk access when at autospeed* the BIOS reprograms the cache control I/O port and the duty cycle of the square wave (see Fig. 8b). Therefore, the control state of the cache is maintained after resumption of maximum speed without interfering with resources (Intel486 control registers) that applications may depend upon.

Micro-DIN and Security Features

The input system consists of three components: the input devices, BIOS functions, and the Intel 8042 keyboard controller. The 8042 keyboard controller communicates with the keyboard and an auxiliary device in a bidirectional, serial format with a synchronized clock generated by the input device. The auxiliary device may be any type of serial input device compatible with the 8042 keyboard controller interface. Some of these are: mouse, touchpad, trackball, and keyboard.

The 8042 keyboard controller receives the serial data, checks the parity, translates keyboard scan codes (if requested), and presents the data to the system as a byte of data. It also provides a password security mechanism to support the network server mode and application software.

Additional security features of the Vectra 486 PC are the power-on password and the mechanical keylock. Both schemes are designed to prevent unauthorized access to the PC. The BIOS provides the software to support the power-on password feature whenever the Vectra 486 is powered on.

The password function can be configured via the easy config utility to request a password either when the PC is powered on, or only when a user needs to use the input devices. If the PC is configured to request a password, the BIOS will display a graphical key symbol to prompt the user for the password. If the user types in the correct password, the PC will continue with its initialization. Otherwise, the BIOS allows three attempts for the user to type in the password before halting the CPU. If the user knows the correct password, the BIOS will allow the user to change or delete the password during the power-on sequence.

When the password is set up to allow limited access, which is also known as the network server mode, all micro-DIN input devices are disabled via the 8042. A PC configured as an unattended file server would typically install the password in the network server mode. In the network server mode, if the BIOS detects a diskette in drive A, it will prompt the user for the installed password

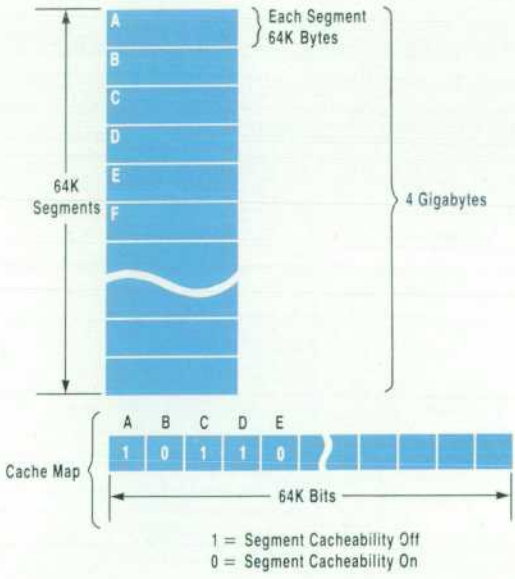


Fig. 7. Cache mapping scheme.

*At autospeed the system operates at its highest speed unmodulated and switches to an effective speed of8 HHz (modulated only when it is accessing a flexible disk.

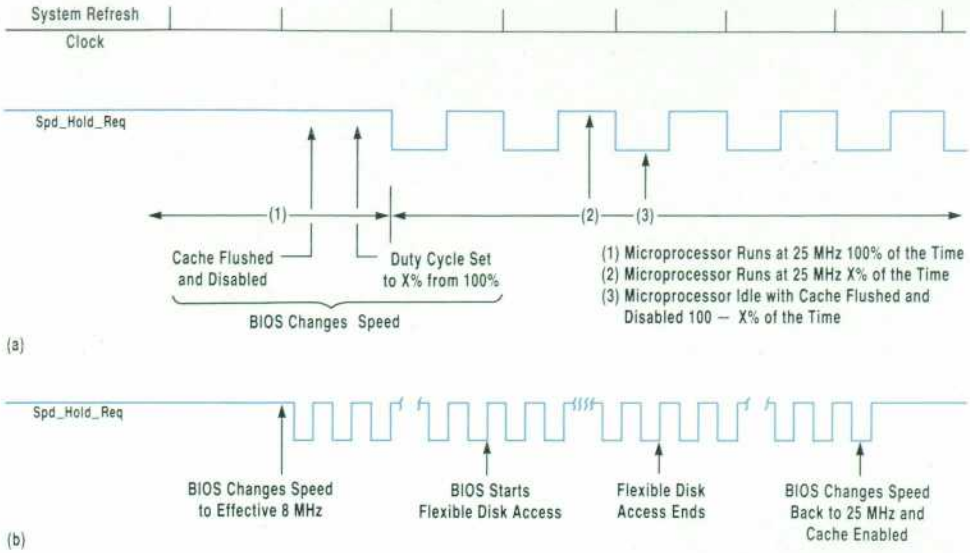


Fig. 8. Timing relationships involved in speed control. (a) For fixed speeds. (b) For autospeed.

because an unauthorized user may be trying to gain access to the PC via a bootable diskette.

The mechanical keylock, used for locking the input devices, can be used in conjunction with the power-on password to provide maximum security. If the keylock is in the locked position and the password function is installed to request the password at power-on, then the user will have to unlock the keylock before typing the password. But if the password function is installed in the network server mode, the keylock doesn't have to be unlocked to type in the password until a user needs to use the input devices.

Since the user may occasionally forget the installed password, the BIOS supports a DIP switch within the Vectra 486 that disables the password. The BIOS uses this switch to allow the user to erase the password without any knowledge of the installed password. This switch is also used to forbid the installation of a password by an unauthorized user. To access the switch, the user must unlock a mechanical keylock to open the PC.

Shadowing and Remapping

The system memory of a Vectra 486 is partitioned into three areas: base memory, reserved memory, and extended memory. The base memory is within the physical address range from 0 to 640K bytes. The reserved address space is within the physical address range from 640 bytes to 1M bytes. Lastly, the extended memory area is all memory above 1M bytes. This memory architecture is known as the PC AT system memory architecture.

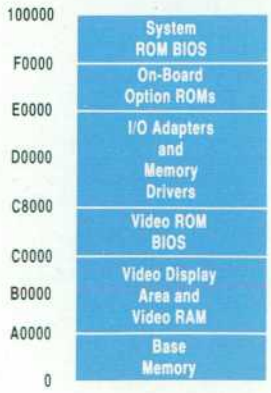


Fig. 9. Reserved memory organization.

Most software applications typically use the base area and some use the extended memory area. The reserved memory is set aside for special system functions and is generally not available for typical software application use. The reserved memory is organized to support the main functional components of a microcomputer (see Fig. 9). The video display area and the video RAM can occupy the lowest portion of the reserved address space, A0000 to BFFFF. The video ROM BIOS can begin at C0000 and typically ends at C7FFF. The address space that begins at C8000 and ends at DFFFF is reserved for special I/O adapters and memory drivers. E0000 to EFFFF is used for onboard option ROMs or backplane I/O ROMs (located in the I/O slots for ISA or EISA cards). F0000 to FFFFF is reserved for the Basic Input/Ouput System (system ROM BIOS).

Since the introduction of this architecture, the cost of memory devices has declined while the density and speed of the components have increased. Processor speeds have increased far beyond the speed that any programmable read-only memory device can effectively support. With the advent of 32-bit bus architectures, systems can physically address four gigabytes of memory, which can be used to support larger, more complex software applications.

Better system performance can be obtained with efficient management of the reserved memory. The Vectra 486 makes use of two memory management schemes: ROM BIOS shadowing and memory remapping. ROM BIOS shadowing is a method used to speed up ROM memory access so that portions of reserved memory that are frequently used can be accessed as quickly as possible. Memory remapping permits unused reserved memory to be used as extended memory.

Shadowing. BIOS and video ROM BIOS routines and data are stored in EPROM (electrically programmable read-only memory). This type of memory is considerably slower than dynamic random access memory (DRAM). Since

the BIOS and video ROM routines are frequently used by the system, contents of the ROM BIOS and video ROM are copied into memory having a faster access time. This technique is known as shadowing.

The conventional organization of the reserved address space, in Fig. 9, shows the locations of the system RAM and BIOS ROMs. In the Vectra 486, as in other microcomputing systems, the conventional organization of reserved memory is enhanced to accommodate some additional system RAM which is located at the same address locations as the system BIOS and video ROMs (see Fig. 10). This memory is called shadow RAM. Another advantage of shadowing is that memory fetches for the Intel486 can be accomplished four times faster because the EPROM is an 8-bit device and system DRAM consists of 32-bit devices.

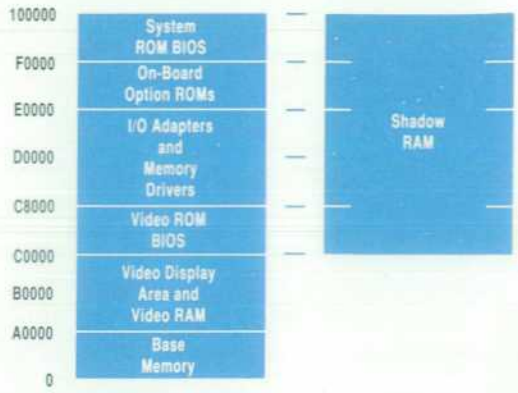


Fig. 10. Shadow RAM.

The conventional approach to shadowing is to copy the contents of the system ROM BIOS and video ROM BIOS to some temporary location. The ROM is then disabled and the shadow RAM is enabled. THE BIOS information, which currently resides in the temporary storage area, is copied into the shadow memory at the same memory address locations from which the information was originally retrieved. Following the shadowing process, any data that was originally in ROM will be accessed from the corresponding location in the faster shadow RAM. This approach requires that the state variables of the memory controller indicate whether the ROM or the shadow RAM is being accessed and if write access to the shadow memory is permitted. These state variables prevent data corruption by ensuring that either the ROM or the shadow RAM is enabled at any time, but not both, and that, once copied, the contents of the shadow memory cannot be inadvertently overwritten.

The disadvantage of the conventional shadowing method is that system memory control states are wasted. The Vectra 486 overcomes this problem by eliminating the ROM and shadow RAM enable variable. The write protect and shadow RAM enable variables are combined into a single state variable. On power-up, the default state of the system will read BIOS (system and video ROM) data from ROM and write this data to the BIOS address space in the shadow RAM. Whenever the shadow RAM is enabled, data read from the BIOS address space will be read from the shadow memory and all write operations to addresses within the BIOS address space are ignored. The shadowing method used in the Vectra 486 system results in a tremendous savings of hardware and valuable system ROM BIOS code space. The additional step of copying BIOS data is eliminated since data can be copied directly from ROM into the shadow RAM.

Remapping. It is often the case that following completion of the shadowing process, portions of the RAM in the reserved memory area are not used. Since typical software applications are not designed to be able to access reserved memory for general storage purposes, the free portions of reserved RAM remain unused. A software application can directly access reserved memory, but without prior knowledge of the configuration of the system, the application could unknowingly cause a device to malfunction by corrupting sensitive data. One approach to this problem is to incorporate an expanded memory manager into the system configuration. An expanded memory manager manages the free reserved memory by allowing the application to use the space as additional base memory, and makes the system appear as if the amount of base memory has been expanded. The disadvantage of using an expanded memory driver is that it is used during run time. This mode of operation degrades system performance. The expanded memory manager also requires memory space for storage of the software routines, thereby leaving less memory for the application to use.

Another conventional method is to remap portions of reserved memory to the top of the physical address space of the system. The disadvantage of this method is that the memory location to which the free memory is moved sometimes does not border on existing memory locations and results in creating a noncontiguous memory structure. Most applications cannot make use of fragmented memory. Also, the conventional remapping scheme is a machine-specific feature, and therefore, all software applications must be customized to take advantage of the remapped memory.

The Vectra 486 solution to memory remapping uses the system configuration information in nonvolatile RAM to instruct the memory controller how to organize the system memory. As an EISA machine, the Vectra 486 has an autoconfiguration program which identifies system components and allocates system resources to obtain maximal system performance.

Memory remapping in the Vectra 486 is a two-step process. The first step is to find the largest contiguous chunk of free reserved memory that can be remapped. The video RAM space (A0000 to BFFFF) is generally not used because the video cards and the embedded subsystems are currently made with their own RAM. On-board option ROMs, which physically reside at E0000 to EFFFF, are rarely used, and the video BIOS address space, C0000 to C8000, fragments the reserved memory area. The way to create the largest contiguous section for reserved memory is to shadow the video BIOS in shadow memory

at E0000, provided that the system does not contain on-board option ROMs and if I/O considerations of the video BIOS support portability. This paradigm creates a 256K-byte (A0000 to DFFFF) chunk of memory that can be remapped (see Fig. 11). The 32K-byte portion between E8000 and EFFFF is unused. The system memory controller is told what area of reserved memory is to be remapped. It must also be noted that systems that use the DOS shell rely heavily on the system BIOS and video BIOS routines, so maximally, only 256K bytes of memory is available for remapping. Systems that use OS/2 and the UNIX* operating system can maximally remap all of reserved memory because these operating systems replace system BIOS and video BIOS and supply all of their own drivers.

The second step in the remapping process is to determine where the physical existence of memory ends. The system BIOS knows this information following the system memory test procedure during the system power-on self-test. This address is passed to the system memory controller. Following the first step, the memory controller has the necessary information for memory remapping. The system memory controller then does the proper address translation.

BIOS shadowing and reserved memory remapping are powerful system features that enhance system performance and make better use of system resources. BIOS shadowing is a common feature among all machines currently on the market. Its primary advantage is to bring more parity between processor speed and ROM access times. To implement this functionality in an efficient manner saves hardware and code space which translates into a cost savings to the customer.

*UNIX is a registered trademark of UNIX System Laboratories Inc. in the U.S.A. and other countries.

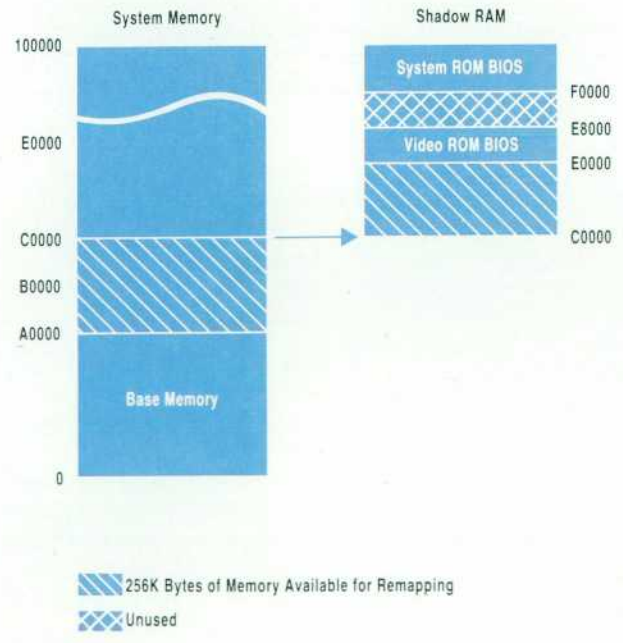


Fig. 11. Remapping.

System Memory Initialization

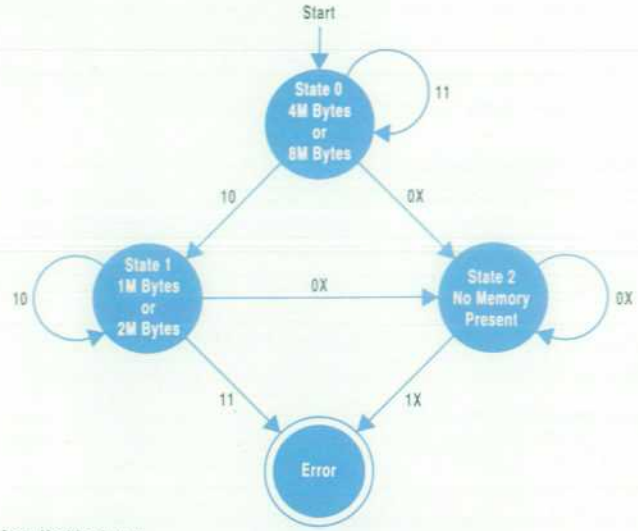
Finite state machines implemented in software can be a very powerful tool. For a system's BIOS, a software finite state machine is ideal for component test situations in which scratchpad memory is not available. This technique is used in testing the system memory configuration in the Vectra 486.

The memory subsystem of the Vectra 486 is a two-way interleaved, linear memory architecture. The system memory board has four memory banks. Each bank can hold two memory modules. The two memory modules are two-way word interleaved. The banks of memory are organized in a linear fashion. See the article on page 78 for more information about the Vectra 486 memory subsystem.

The memory modules are packaged in single in-line memory modules and come in 1M-byte, 2M-byte, 4M-byte, and 8M-byte varieties. The 1M-byte and 4M-byte modules are single-density modules. The 2M-byte and 8M-byte modules are double-density modules. Each memory bank on the system memory board must contain a pair of memory modules that are the same size and have the same density type. Moreover, density restrictions require that all memory banks contain memory modules of the same density type. The linear structure of the memory subsystem requires that the amount of memory in a bank be less than or equal to the amount of memory in a bank that logically precedes it. The exception to this rule is the first bank because no other memory bank precedes it.

Before system memory can be tested, the memory subsystem configuration must be verified. The power-on self-test procedure in the system BIOS is responsible for this task. The use of system resources must be kept to a minimum because system memory is not available at this stage in the system power-on initialization process. A software finite state machine is ideal in this situation, since only the registers within the processor are available. The finite state control is guided by the memory module identification encoding (each memory module has information encoded within it that specifies the size and density type of the module). The memory state machine evaluates the identification for each memory bank and verifies that the current memory configuration (linearity, uniform bank densities, etc.) is valid.

The software finite state method is very effective when considering that each of the four banks can have one of five types of memory modules. The number of possible memory configurations is 4^5, or 1024 possible configurations. Of the 1024 possible configurations, 28 are valid. If module density errors are found first, then the linearity check can be done with a software finite state machine with four states. Fig. 12 shows the finite state machine for testing the Vectra 486 memory configuration.



State Machine Inputs
SIMM Presence Detect Bit = {1→SIMM Present, 0→SIMM Not Present}
SIMM Part Size = {1→2M-Byte or 8M-Byte SIMM, 0→1-Mbyte or 4-Mbyte SIMM}
X = Don't Care

SIMM Presence	SIMM Size	Current State	Next State	Comments
0	X	0	2	
1	0	0	1	
1	1	0	0	
0	X	1	2	
1	0	1	1	
1	1	1	Error	4M-Byte or 8M-Byte Part Found after a 1M-Byte or 2M-Byte Part
0	X	2	2	
1	X	2	Error	SIMMs Found after an Empty Bank

Fig. 12. The finite state machine for Vectra 486 memory configuration testing.

Defect Tracking

Because BIOS source code was shared among independently managed projects in different locations, BIOS problems had to be tracked not only for the Vectra 486 product but also for several other HP personal computer products as will.

To keep track of prerelease problems across the various projects and to provide a means of collecting more global process improvement metrics, a custom dBASE IV® application was written for use by all of the BIOS teams. Problems reported on the Vectra 486 were sent by testers to a special electronic mail account to be entered into the database. The use of a standard problem form allowed the collection of valuable statistical as well as problem-specific information. Information about BIOS problems in common files was shared directly with each team by exchanging database files. The flexibility of the PC database application allowed each team to adapt the database application to their needs without affecting the ability to share data. This was important, because several BIOS teams could have been involved in resolving any one problem.

As each problem was investigated and resolved, status information was entered into the database. Detailed information, such as which code module contained the problem and when the problem was introduced, was readily available. The typical weekly status report contained a simple summary of the active problems, the problem owners, and their current status. One BIOS team member acted as the bug manager and helped keep everyone informed of the progress being made to resolve a problem.

BIOS Qualification and Test

The BIOS qualification effort for the Vectra 486 project was an improvement over previous BIOS development efforts. Because of major revisions to the BIOS in the Vectra 486 and the need to produce quality software in minimum time, a special BIOS qualification team was formed. This team consisted of four engineers and a software technician whose main job was to verify that the BIOS specifications were correct. In the past, the job of qualification of the BIOS was left up to the developer of the BIOS code. For the Vectra 486, it was felt that qualification would be more thorough if the persons developing and executing the tests were not the individuals developing the BIOS code.

To make best use of our limited resources, two types of test strategies were developed: white box and black box testing. Black box testing used a high-level language program, such as C, to verify the functionality and quality of the BIOS. The C functions invoked DOS functions, BIOS functions, and I/O registers to test the BIOS. This was the standard method of testing most programs. However, when testing the BIOS, we did not always have the luxury of relying on an operating system such as DOS because much of the BIOS functionality had to be tested during the machine initialization, and was inaccessible to high-level programs.

An alternate method was developed using some new approaches and working with special development tools to perform white box testing. New features such as EISA initialization, memory initialization, and shadowing routines were tested using this approach. This method forced two engineers to read and understand the actual code: the original designer and the one developing the test. This task alone required an in-depth understanding of the BIOS modules on an instruction-by-instruction basis. The concept of having two people intimately understand each module is not new, but typically resource limitations make such an arrangement a luxury.

There were many advantages to this type of testing. For one thing, it allowed us to simulate some of the system errors. For example, if the user had an invalid memory configuration, this error could be simulated without even changing the memory inside the computer. Furthermore, all the valid memory configuration could be tested via this program. There were well over 1000 memory configurations that could be tested through one automated program. Normally, this process would involve a technician physically changing the configuration each time. Another advantage was that the BIOS could be tested without the hardware. This proved to be very helpful since the BIOS was being developed before the hardware was available. With this method, two tasks could be done concurrently. Once the hardware was available, the tests could also be executed on the hardware.

*DBASE IV is a registered U.S. trademark of Ashton-Tate Corp.

The test strategy that was developed by the BIOS qualification team enabled the team to perform tests on the BIOS that would normally not have been done. Once the tests were developed, they could be added to the test suite for regression testing and other BIOS related tests.

Conclusion

The HP Vectra 486 BIOS development effort was a major milestone in HP's Personal Computer Groups software development history from the perspective of both new PC technology advances supported and new processes introduced. Support for new technologies and features like the EISA architecture and an advanced memory controller was incorporated into BIOS with high quality while meeting system schedules.

New or enhanced processes with their associated tools were incorporated to meet customer needs and HP business requirements. A customized source version control process and tools allowed efficient, multi-site, simultaneous PC BIOS development with maximum code reuse. Brand new defect tracking and component qualification processes and tools allowed quality BIOS development concurrent with PC hardware development.

Acknowledgments

The BIOS group would like to thank the Vectra 486 hardware team, the easy config utility team, the quality assurance group, the test development group, and other individuals throughout the division who participated in the Vectra 486 development. In particular we would like to thank Joyce Higa, Ruth Lim, Dave Wilkins, Roderick Young, Jongwon Yuk, Van T. Dam, Tewelde Stephanos, Paul Schlegelmann, Becky Smith, Dirk Massen, Martin Goldstein, and Chin Pin Wu. A special thanks to our project manager Anil R. Desai and section manager Tom Battle for their continued support. The support of all these people has made the Vectra 486 project a success and helped us deliver a high-performance and high-quality machine to the end user.

Performance Analysis of Personal Computer Workstations

The ability to analyze the performance of personal computers via noninvasive monitoring and simulation allows designers to make critical design trade-offs before committing to hardware.

by David W. Blevins, Christopher A. Bartholomew, and John D. Graf

Today's high-performance personal computers are being used as file servers, engineering workstations, and business transaction processors, areas previously dominated by large, costly mainframes or minicomputers. In this market, performance is of paramount importance in differentiating one product from another. Our objective at HP's Personal Computer Group's performance analysis laboratory is to ensure that performance is designed into HP's offering of personal computers. To achieve this, analysis of a personal computer's subsystem workloads and predictive system modeling can be used to identify bottlenecks and make architectural design decisions. This article describes the tools and methodologies used by HP engineers to accomplish performance analysis for personal computers.

The toolset currently being used at the performance analysis laboratory consists of specialized hardware and software. Typically, the hardware gathers data from a system under test and then the data is postprocessed by the software to create reports (see Fig. 1). This data can also be used to drive software models of personal computer subsystems.

Hardware-Based Tools

The two hardware-based performance analysis tools shown in Fig. 1 are the processor activity monitor (PMON) and the backplane I/O activity monitor (BIOMON). Both tools are noninvasive in that they collect data without interfering with the normal activity of the system under test.

Processor Activity Monitor

The processor activity monitor is a hardware device that monitors a personal computer's microprocessor to track low-level CPU activity. The PMON is sandwiched between the computer's CPU and the CPU socket (see Fig. 2). The PMON monitors the processor's address and control pins. For each CPU operation, the PMON will track the dura-

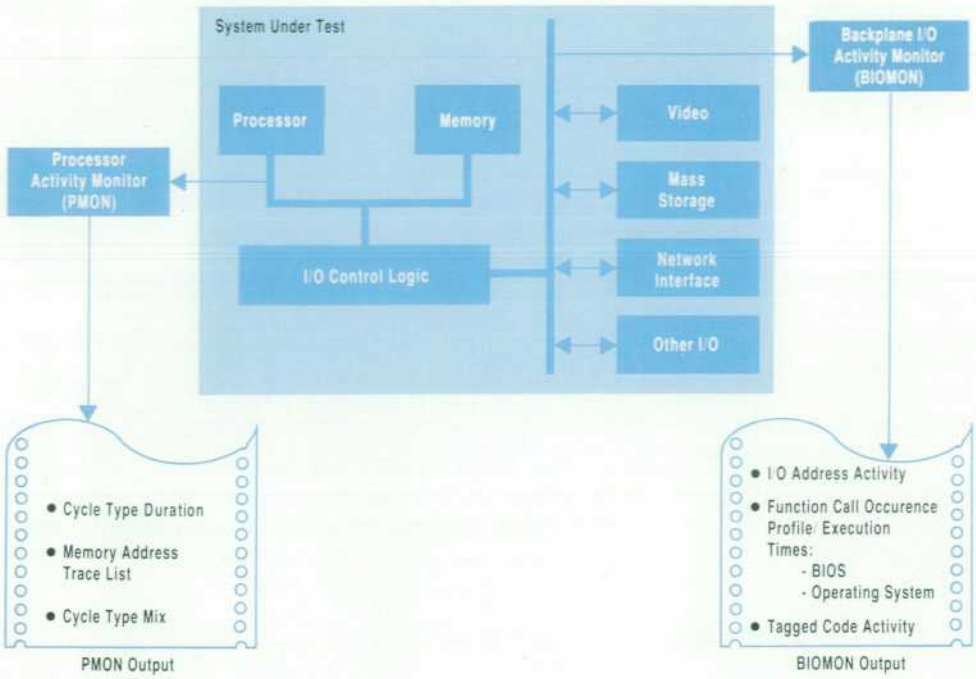


Fig. 1. The hardware-based performance analysis tools PMON and BIOMON connected to the system under test.

tion and address of the operation and output the results to the data capture device.

Gathering statistics on the activities of a personal computer's microprocessor can be very useful in making design decisions about the arrangement of the support circuitry (e.g., cache and main memory, I/O bus interface, and bus lines). In addition, trace files that detail the CPU's requests to the memory system can be used to drive software simulations of various cache memory arrangements as well as more comprehensive CPU and memory or system simulations.

Two data capture devices are commonly used in conjunction with the PMON. The first, an HP 16500 logic analyzer configured with optional system performance analysis software, generates two main types of data. One is a histogram that shows the occurrence mix of a user-defined subset of the possible CPU cycle types (Fig. 3a). The performance analysis software averages 1000 samples of cycles from the PMON on the fly, giving a randomly sampled profile of processor activity throughout the duration of a performance benchmark. The second type of data provided by the HP 16500 is real-time calculation of the minimum, maximum, and average time intervals between the beginning and end of user-defined events (Fig. 3b). The performance analysis software averages the interval calculations on the fly over a large number of samples to give, for example, the average interarrival time of writes to video memory in a CAD application.

The other data capture device used with the PMON is a less intelligent but higher-capacity logic analyzer. This instrument has a 16-megasample-deep trace buffer (as opposed to the 1000-sample deep buffer in the HP 16500).

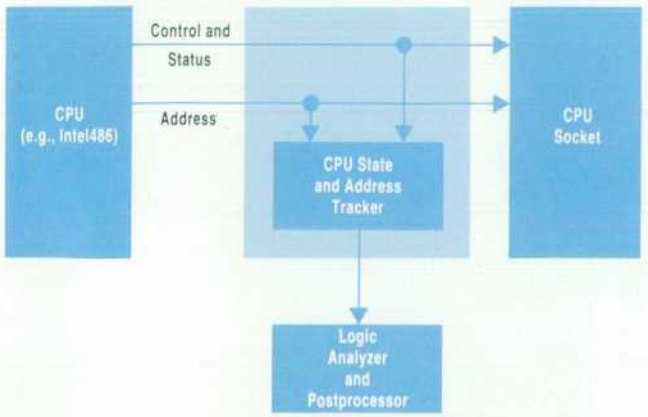


Fig. 2. Processor activity monitor external connections.

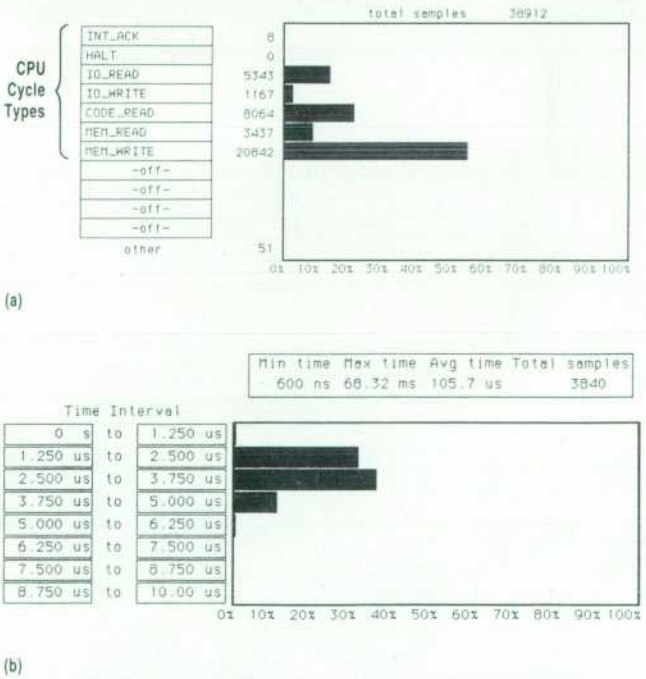


Fig. 3. Sample histograms from an HP 16500 logic analyzer. (a) The occurrence mix of a subset of Intel486 CPU cycle types. (b) Interarrival times of writes to video memory.

Address Range	Number of Occurrences	% of Totals	Total CPU Clocks	% of Time	Average CPU Clocks
Interrupt Vectors	3249	1.34%	38297	1.55%	11.79
STD-BIOS Data	1153	0.48%	12340	0.50%	10.70
DOS	23670	9.80%	249800	10.09%	10.55
Application	197143	81.59%	2028358	81.91%	10.29
Video RAM	248	0.10%	5890	0.24%	23.75
BIOS ROM	1967	0.81%	21284	0.86%	10.82
Extended Memory	14208	5.88%	120240	4.86%	8.46

Fig. 4. PMON address range summary report.

Cycles from the PMON are captured in this buffer in real time and the data is later archived to a host computer's hard disk. The buffer typically holds four to five seconds of continuous bus cycle activity generated by a 25-Mhz Intel486 microprocessor running an MS-DOS® application. The data can then be used to drive software simulations or processed to create summary reports, such as an address range summary of how the processor's address space is used by operating systems and application software (see Fig. 4).

Backplane I/O Activity Monitor

The backplane I/O activity monitor, or BIOMON, also captures information from a personal computer's hardware, but instead of the CPU activity, the I/O activity on the ISA (Industry Standard Architecture) or EISA (Extended Industry Standard Architecture) backplane is monitored (Fig. 5). The BIOMON consists of two backplane I/O cards: the qualify and capture card and the monitor card. The qualify and capture card resides noninvasively in the SUT (system under test) and is connected via a ribbon cable to the monitor card, which is located in another personal computer called the monitor system. The monitor system receives, stores, and processes the I/O events captured on the SUT's backplane.

During operation the qualify and capture card is loaded with capture enable flags for each of the I/O addresses whose activity is to be monitored on the SUT backplane.

Once the qualify and capture card is set up, I/O address accesses on the SUT's backplane cause an event information packet (address, data, etc.) to be transferred to a first-in, first-out (FIFO) holding buffer, allowing for asynchronous operation of the SUT and the monitor system. The FIFO is unloaded by transferring each event information packet to the monitor system's extended memory. At the end of event capture, this trace of I/O events can be either stored to hard disk or immediately postprocessed for analysis.

One very powerful use of BIOMON is the performance analysis of marked code, which is code that has been modified to perform I/O writes at the beginning and end of specific events within a software routine. The frequency of occurrence and execution time for each marked software event can then be analyzed under different configurations to find existing or potential bottlenecks and the optimum operating environment.

As an example, the performance analysis laboratory has developed a special installable software filter that writes to specific I/O addresses at the beginning and end of DOS and BIOS (Basic I/O System) interrupts. For our purposes, a write to I/O port 200 (hexadecimal) denotes the beginning of an interrupt, and a write to port 202 denotes the end of an interrupt. The trigger address comparator is told to capture data for I/O addresses 200 and 202, and any normal application using DOS or BIOS functions is run on the SUT. The resulting trace can be postprocessed to show which DOS and BIOS routines were used by the application, how many times each one was called, and how long they executed (Fig. 6a). Other information such as the interarrival time between events, exclusive versus inclusive service time for nested events, and total time spent in various application areas can also be extracted (Fig. 6b). Analysis of this information can assist the software engineer in optimizing frequently-used functions in DOS and the BIOS.

This technique can also be used to analyze protected-mode operating systems such as OS/2 and UNIX.** However, because of their nature, these environments must

*MS-DOS is a U.S. registered trademark of Microsoft Corp.

**UNIX is a registered trademark of UNIX System Laboratories in the USA and other countries.

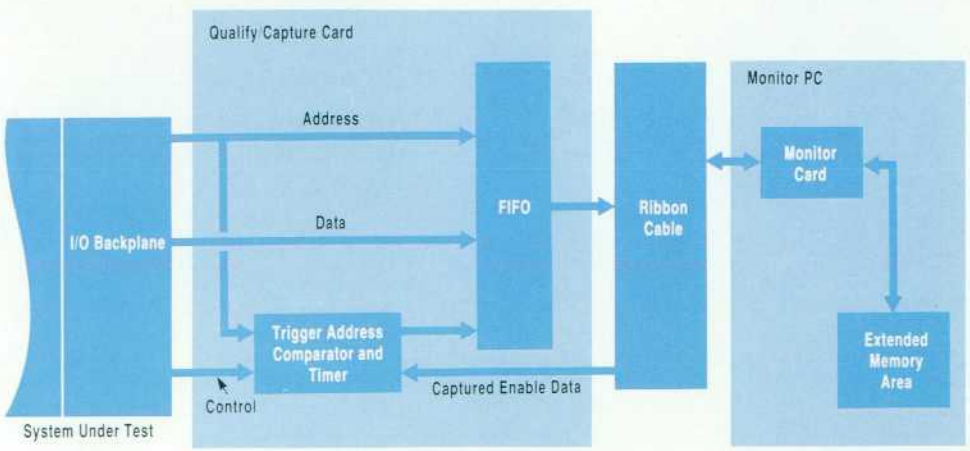


Fig. 5. The backplane I/O monitor components.

have tags embedded into the operating system code. (Protected-mode operating systems do not allow a user to arbitrarily write to specific I/O locations.)

Another use of the BIOMON is to trigger on reads and/or writes to I/O locations associated with accessory cards such as disk controllers, serial and parallel interfaces, video cards, and so on. For instance, the interarrival rates of data read from a disk controller could be examined to determine the actual data transfer rate attained by the disk mechanism or drive controller subsystem. Additionally, by monitoring the disk controller's command registers, an application's disk I/O can be fully characterized.

Software-Based Tools

The software-based tools used by the performance laboratory allow simulation of different memory architectures.

Cache Simulator

The cache simulator is a trace-driven simulation based on the Dinero cache simulator from the University of California at Berkeley.[1] The simulator takes as its input a list of memory accesses (trace file) and parameters describing the cache to be simulated. These parameters include cache size, line size, associativity, write policy, and replacement algorithm. The cache simulator reads the memory accesses from the trace file and keeps statistics on the cache hit rate and the total bus traffic to and from main memory. When the entire trace file has been read, the simulator generates a report of the cache statistics.

A trace file is generated by connecting the PMON to a CPU and storing all the memory accesses on the CPU bus to the high-capacity logic analyzer described above. The data collected from the analyzer can later be downloaded to a host personal computer and archived to hard disk. To get useful data from the simulator, however, the input trace file must be long enough to "prime" the simulated cache. The first several thousand memory accesses in the trace file will be misses that fill up the initially empty cache. The simulator will report artificially low hit rates, because in a real system the cache is never completely empty. If the trace file is significantly longer than N_p*, priming effects are minimized. When simulating a 128K-byte, 2-way associative cache external to the Intel486, N_p is approximately 40,000.[2] The high-capacity logic analyzer mentioned above is able to store 16 million memory accesses from the Intel486 via the PMON. A trace file containing 16 million accesses results in a priming error of less than 1% in the hit rate calculation (assuming a hit rate of approximately 90% for the 128K-byte, 2-way cache).

Memory Subsystem Simulator

The memory subsystem simulator, a program written in C++, is a true event-driven simulation that keeps track of time rather than just statistics. It builds on the cache simulator by integrating it into a more comprehensive model that simulates access time to memory. It accepts a parameter file that includes cache parameters, DRAM and SRAM access times, and other memory architecture parameters. It also reads in a PMON trace file, although this one must contain all accesses (not just memory), and their durations so that the simulator can keep track of time. The result is essentially a running time for the input trace file, along with statistics on all aspects of the memory subsystem.

This simulator can be used for making design trade-offs within a memory subsystem, such as cache size and organization, DRAM speed, interleave, page size, and write buffer depth. Fig. 7 shows the sample results of a memory subsystem simulation of relative memory performance versus external cache size for a 33-MHz Intel486 running a typical DOS application. By simulating various design alternatives in advance, the design engineer can arrive at a memory architecture that is tuned for optimum performance before committing to hardware.

Conclusion

The performance analysis laboratory of HP's Personal Computer Group has developed a suite of hardware and software tools to aid in the design process. The hardware

*N_p equals the number of memory accesses in the trace file needed to fill the cache.

Interrupt	Number of Occurrences	% of Total	Total Time	% of Total	Average Time	Minimum Time	Maximum Time
13:02	1,976	1.12%	24,630.00 ms	66.26%	12,470.00 μs	569.39 μs	57,880.00 μs
15:90	13,133	7.42%	71.58 ms	0.19%	5.45 μs	4.80 μs	34.33 μs
15:91	13,133	7.42%	63.90 ms	0.17%	4.87 μs	4.56 μs	7.56 μs
76:00	13,133	7.42%	209.24 ms	0.56%	15.93 μs	14.89 μs	21.37 μs

(a)

Area	Total Time	% of Time
Video	2,612.00 ms	15.76%
Disk	12,732.00 ms	76.81%
Serial	12.23 ms	0.07%
Parallel	0.00 ms	0.00%
Keyboard	134.95 ms	0.81%
Timer	151.48 ms	0.91%
Other	934.12 ms	5.64%

(b)

Fig. 6. Reports derived from postprocessing BIOMON trace file data. (a) Interrupt level summary report. (b) Application level summary.

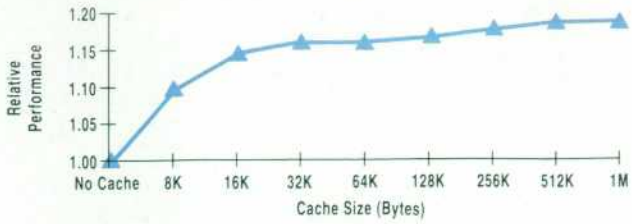


Fig. 7. Sample results of a memory subsystem simulation of relative memory performance versus external cache size for a 33-MHz Intel486 running a typical DOS application.

tools give design engineers insight into the low-level performance of existing systems, and the software tools use the data produced by the hardware tools to predict the performance of future architectures.

The performance tools were used extensively in designing the HP Vectra 486, and more recently the Vectra 486/33T. The tools helped show that a burst memory controller (described on page 78) was a better price/performance solution than an external memory cache for the 25-MHz Vectra 486, and that an external cache was a necessity for the 33-MHz Vectra 486/33T. The tools also helped predict the performance gain of memory write buffers in a Vectra 486 system. This resulted in the addition of write buffers to the Vectra 486/33T memory architecture.

Acknowledgments

The original PMON was designed by Carol Bassett and Mark Brown. Later versions were implemented by Steve Jurvetson. BIOMON's design owes thanks to several people. Its predecessor was designed by Bob Campbell and Greg Woods. Subsequent help came from Ali Ezzet, Chris Anderson, and John Wiese. Greg Woods coded the installable filter and the original postprocessing program. Jim Christy provided additional software help.

References

1. M. D. Hill, "Test Driving Your Next Cache," *MIPS*, Vol. 1, no. 8, August 1989, pp. 84-92.
2. H. S. Stone, *High Performance Computer Architecture*, Second Edition, Addison-Wesley, 1990.

HEWLETT-PACKARD
JOURNAL
October 1991 Volume 42 • Number 4
Technical Information from the Laboratories of Hewlett-Packard Company
Hewlett-Packard Company, P.O. Box 51827
Palo Alto, California, 94303-0724 U.S.A.
Yokogawa-Hewlett-Packard Ltd., Suginami-Ku Tokyo 168 Japan
Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive, Mississauga, Ontario L4V 1M8 Canada

HEWLETT PACKARD

5091-2505E