HEWLETT-PACKARD

JOURNAL

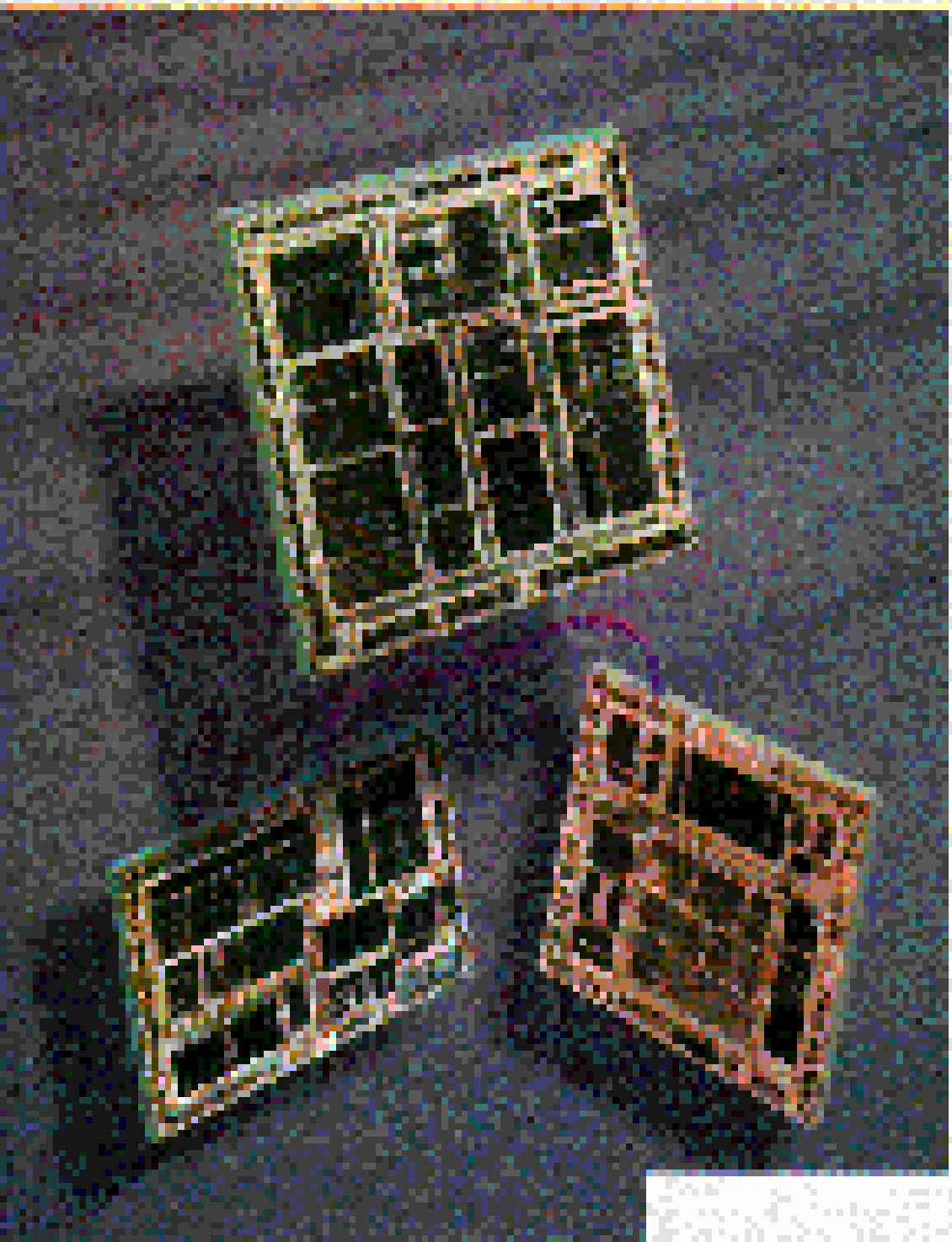


table of contents

April 1995,
Volume 46, Issue 2

Articles

A Low-Cost, High-Performance PA-RISC Workstation with Built-In Graphics, Multimedia, and Networking Capabilities

Designing as a set the three VLSI components that provide the core functions of CPU, I/O, and graphics for the HP 9000 Model 712 workstation balanced performance and cost and simplified the interfaces between components, allowing designers to create a system with high performance at a low cost.

by Roger A. Pearson

Designing a workstation entails defining various functional blocks to work together to provide a set of features at a desired level of performance at the lowest possible cost. Often, many parts of the design are leveraged from previous designs, and only new functionality is designed from scratch. This approach may save development costs, but could result in a product that is more costly to build.

When one component of the system design has performance that can't be taken advantage of, whether because of architecture limitations or other components' performance limitations, then the system design suffers by having to carry the cost of that unused performance. By designing with the total system in mind, so that all components of the design are optimized to work together with no wasted performance, cost can be minimized. The designers of the HP 9000 Series 700 Models 712/60 and 712/80 took this approach to offer a high-performance combination of graphics, multimedia, and networking capabilities at new low prices. The objectives of the new design included:

- Providing the high performance of a PA-RISC workstation at the lowest possible cost
- Improving the performance and capabilities of multimedia functions through simple extensions to the instruction set
- Enabling an extensive set of communication features through low-cost option cards
- Designing for high-volume manufacturing.

Instrumental in meeting these objectives was the decision to design three new custom VLSI chips together, as a set, to achieve new levels of price/performance for the core functions of CPU, I/O, and graphics.

Overview

Three new VLSI chips provide most of the functionality of the Model 712 workstation. The PA 7100LC CPU chip interfaces directly to the cache and main memory. The LASI (LAN/SCSI) chip does most of the core I/O needed for entry-level workstations. The graphics subsystem consists of the graphics chip and the frame buffer VRAMs. All three chips communicate through the GSC (general system connect) bus. Fig. 1 shows a block diagram of the Model 712 system.

The Models 712/60 and 712/80 are very similar and differ only in their cache sizes and cache speeds and in the main system clock speeds.

The Processor

The compute power of the Model 712 system is provided by the PA-RISC PA 7100LC processor,[1,2] which is packaged in a 432-pin ceramic PGA. The CPU design was optimized for the Model 712 and includes the following features:

- Superscalar CPU
- 1K-byte instruction buffer
- Multimedia support
- Cache control for up to 2M bytes of external cache
- ECC (error correction coding) memory controller

The clock frequencies of the Model 712/60 and the Model 712/80 are 60 MHz and 80 MHz respectively. The PA 7100LC is described in more detail in the article on page 12.

Cache

The PA 7100LC CPU uses an external cache. An external cache allows system designers to change the size of the cache easily to meet their performance and cost goals. Furthermore, off-chip cache provides all the performance necessary, without limiting the CPU frequency.

The external cache is 64K bytes on the Model 712/60 and 256K bytes on the Model 712/80 and is logically split into equal halves for the instruction and data caches. Combining the caches saved pins on the CPU. To further reduce costs, industry-standard SRAMs (static RAMs) are used. Table I shows the SRAMs used in the Model 712 systems.

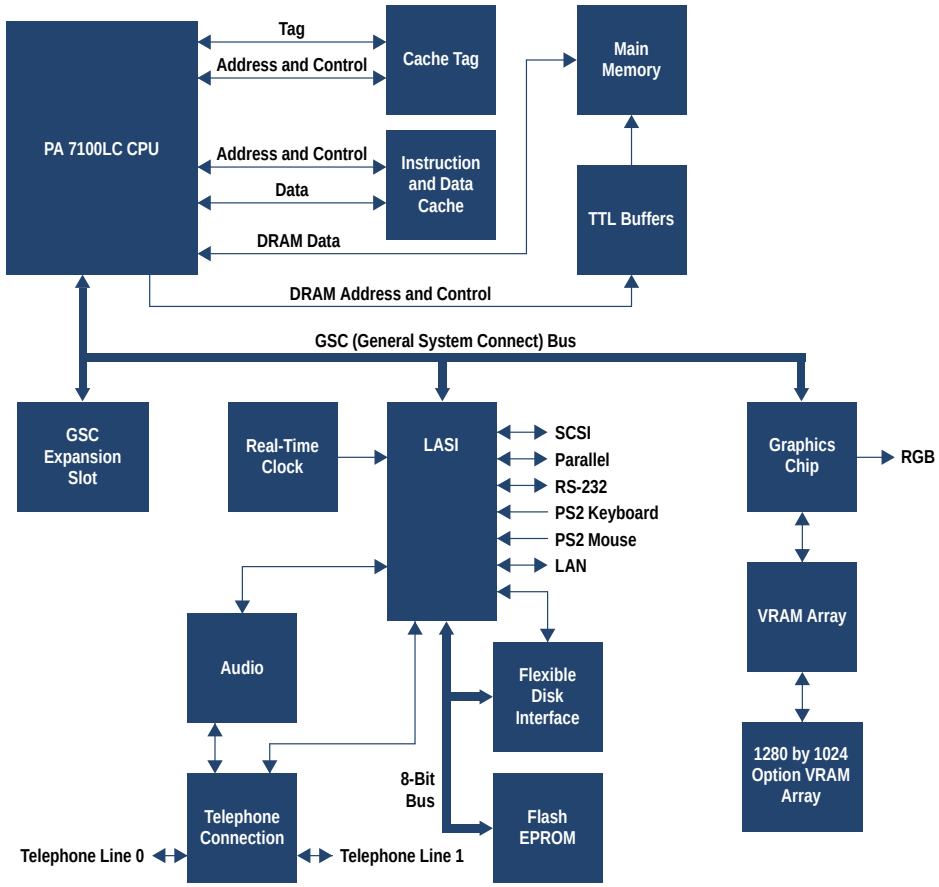


Fig. 1. Block Diagram of the HP 9000 Model 712 hardware.

Table I
Static RAMS Used in the Model 712 Systems

Model	Function	Size	Speed	Quantity
712/60	Tag	8K bytes	12 ns	4
	Data	8K bytes	12 ns	6
	Data	8K × 9 bits	12 ns	2
712/80	Tag	32K bytes	10 ns	4
	Data	32K bytes	10 ns	6
	Data	32K × 9 bits	10 ns	2

Main Memory

The main memory for the Model 712 systems has been engineered to provide high performance with industry-standard 70-ns SIMMs (single inline memory modules). Currently supported SIMMs are available in 4M-, 8M-, 16M-, and 32M-byte sizes. Four slots are available and must be filled in pairs for a maximum of 128M bytes.

The Model 712's main memory design minimizes the average cache miss penalty. The main memory controller returns double words (eight bytes, since a word is four bytes) back to the CPU. Each cache line is made up of four double words. When there is a cache miss, the one double word of the four in the cache line that was missed is referred to as the critical word. To minimize the miss penalty, the double word containing the critical word is sent back to the CPU first, followed by the remaining three double words.

Bandwidth is maximized by using fast page mode when consecutive accesses reside on the same page. This is often the case when large blocks of memory are accessed and is very common in windowed graphics systems.

The General System Connect Bus

The general system connect, or GSC, is the local bus that connects the three VLSI devices and the optional I/O card. The GSC bus is designed to provide maximum bandwidth for memory-to-graphics transfers. The bus has 32-bit multiplexed address and data lines to minimize the number of signals. Other features of the bus include:

- Operation at half the CPU frequency (30 or 40 MHz)
- Support for 1-, 2-, 4-, 8-, 16-, or 32-byte transactions
- Central arbitration
- Parity generation and checking.

Normally, bus transactions are terminated by a turnaround state that allows drivers to be turned off before the drivers for the next transaction are turned on. To improve graphics performance, the bus supports back-to-back writes to the same device without the turnaround state. This improves throughput on transfers of large blocks of data from main memory to graphics.

During transfers from memory to I/O, it is sometimes necessary to lock the CPU out of memory (e.g., when semaphores are used). To facilitate this, the GSC bus provides a locking mechanism, which prevents the CPU from accessing memory (to service a cache miss, for example).

Graphics

The graphics subsystem consists of a graphics chip and four on-board VRAMs (video RAMs), which provide a 1024-by-768-

pixel frame buffer with a depth of eight planes at a refresh rate of 72 Hz. An optional high-resolution VRAM board increases resolution to 1280 by 1024 pixels.

The graphics chip was designed with the other system components to provide high performance at a minimal cost. For more information on the graphics chip, see reference 3 and the article on page 43.

Built-in I/O

The Model 712 features a number of built-in I/O devices that are intended to address the needs of the majority of users.

Support for these functions is provided largely by the LASI I/O VLSI chip. LASI is a highly integrated chip that provides a significant reduction in system cost and increased reliability. The chip is packaged in a 240-pin MQUAD package. The LASI chip is described in more detail in the article on page 36 and in reference 4.

The following sections briefly describe the LASI chip's built-in capabilities.

IEEE 802.3 LAN. LASI contains an Intel 82C596 megacell which was ported to work with HP's IC process. The LAN transceiver, which was not practical to include on LASI, is loaded on the printed circuit board. The transceiver interfaces to both the AUI (attachment unit interface) and Ethertwist media.

SCSI. The Model 712 uses an 8-bit single-ended SCSI interface for the optional internal hard drive and external peripherals. The SCSI-2 interface is implemented entirely within LASI through a megacell that was designed by HP and NCR. A netlist for the NCR 53C710 was imported into HP's design environment. The design was then tuned to work in HP's IC process.

By keeping the SCSI bus stub length to a minimum on the printed circuit board and on the connection to the optional internal drive, SCSI termination on the internal side is greatly simplified. Short stub lengths allow the bus to be terminated on the printed circuit board, whether the optional internal drive is present or not. This saves cost by obviating the need for special terminators which would otherwise have to be enabled or disabled (manually or electrically), depending on the presence or absence of the optional internal drive.

Audio. 16-bit CD-quality audio playback and record capability is provided by the audio circuitry, which consists of a Crystal Semiconductor CS4216 CODEC and supporting circuitry. The LASI chip also includes the serial interface to the CS4216. Headphone, microphone, and line-in connectors are located on the rear panel. Standard sampling rates include 8, 44.1, and 48 kHz.

Real-Time Clock. A real-time clock is designed into the LASI chip. Battery backup keeps time while the workstation is powered down.

PS/2. There are two PS/2 connectors on the rear panel that allow connection to a low-cost industry-standard keyboard and mouse. The PS/2 interface circuitry is integrated into the LASI chip.

RS-232. An RS-232 interface has also been designed into the LASI chip. The Model 712 buffers the signals with a MAXIM 211 to provide an RS-232 serial port. LASI buffers inbound and outbound data with 16-byte FIFOs, at baud rates from 50 to 454 kbits/s.

Parallel. The LASI chip also provides a parallel port conforming to the Centronics industry standard.

Flexible Disk Support. A Western Digital WD37C65C flexible disk controller interfaces LASI to an optional internal personal-computer-style flexible disk drive.

Flash EPROM. An 8-bit bus on the LASI chip is demultiplexed by two 74CHT374 latches to provide the address and data lines necessary to address the two 128K-byte flash EPROMs that contain the boot firmware. The flash EPROMs are also used to store configuration parameters, eliminating the need for an EEPROM and its associated cost.

I/O System Support. LASI provides a number of miscellaneous I/O system support functions, including:

- Clock generation. LASI derives all the necessary clocks required by the I/O circuitry from the main system clock. It does so by using simple divide-by-n counters and two digital phase-locked loops.
- System arbitration support. LASI arbitrates GSC bus requests from the I/O devices within LASI, as well as from the CPU and optional expansion card.
- Interrupt support. LASI also provides and manages external interrupt capability for the various I/O devices.

Optional I/O

For those users who need functionality beyond that provided by the built-in I/O, the Model 712 includes two personality slots that can be configured with a variety of other I/O functions. The first of these slots is referred to as the expansion slot and includes a connection to the GSC bus. The second slot provides a connection to the serial audio stream, and is intended for telephone functions. This slot is called the telephony slot.

Expansion Cards. Expansion cards are optional cards that connect directly to the GSC bus to provide a variety of other I/O functions.

Since LASI has a configurable address space and can be configured as an arbitration slave, many of the expansion cards rely on a second LASI chip to implement much of their functionality.

The following optional expansion cards are provided for the Model 712:

- Second serial port. The second serial port card uses its own LASI chip and support circuitry identical to that on the system board to provide an additional RS-232 port.
- Second LAN AUI and second serial interface. This card also uses a LASI chip and circuitry similar to that on the system board to add an additional IEEE 802.3 LAN with an attachment unit interface (AUI) and a second RS-232 interface.
- X.25 and second serial interface. A Motorola 68302 multiprotocol processor interfaced to the 8-bit bus of a slave LASI provides X.25 networking to a 25-pin X.21bis port for speeds of 1.2 kbits/s to 19.2 kbits/s. The second RS-232 serial interface is implemented in the same fashion as the other cards.
- Second display. A second display can be added to the system with the second display card. This card duplicates the

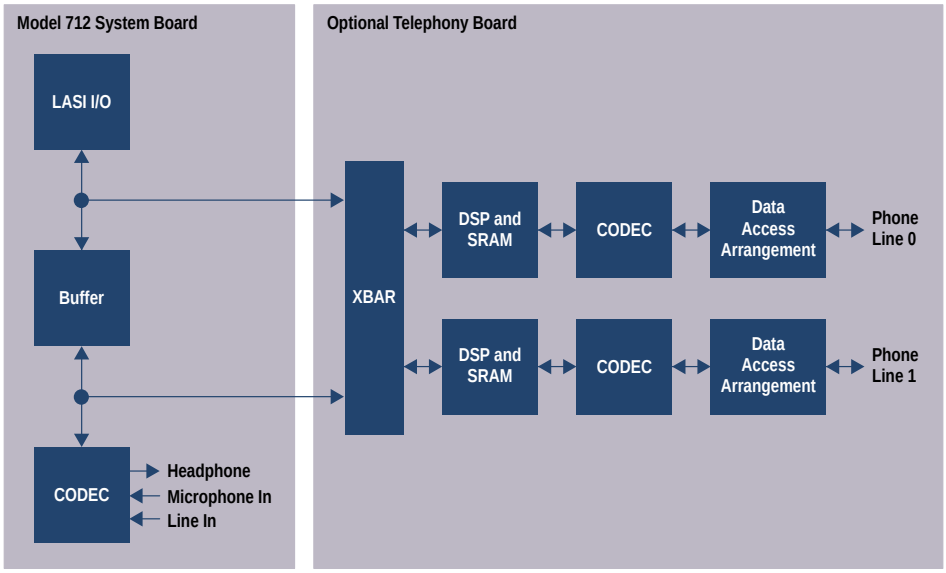


Fig. 2. Block diagram of the Model 712 audio and telephony circuits.

graphics functionality that is already built into the system board by replicating the graphics chip and its supporting circuitry.

- Token Ring/9000. The Token Ring/9000 card provides IEEE 802.5 LAN functionality through the use of a Texas Instruments token ring controller chip and a custom ASIC that provides the GSC interface. Unshielded and shielded twisted pair connections are provided at data rates from 4 Mbits/s to 16 Mbits/s.
- Second display and second LAN AUI/RS-232. This option combines the features of the second graphics display and the second LAN AUI/RS-232 options. Since the circuitry for this option would not fit on a single expansion slot card, some of the circuitry resides on a daughter card that is connected to the expansion slot card. The daughter card gets power and mechanical support through the telephony connector, so when this option is installed, the telephony option is not available.

Telephony. The telephony card installs in the telephony slot and provides two lines of telephone access. Each of the lines can be configured to support voice, data modem, or fax modem.

The system board's headset and microphone serve as the human interface for voice telephony, and an interface chip on the telephony card called XBAR links the system board's audio circuitry to the telephony functions (see Fig. 2).

This arrangement allows recording and playback during telephone conversations. It also supports digital mixing of microphone, line-in, telephone, and prerecorded audio. Caller-ID decoding is supported, as are DTMF (dual-tone multifrequency) encoding and decoding, and dual-line conferencing.

The XBAR chip serves to route information between the LASI I/O chip, the audio CODEC, and the DSP blocks in a variety of programmable ways. Data is transferred to and from the system board through two serial data paths. Two additional serial paths send and receive data to and from the DSPs. Two 8-bit parallel ports are used by the DSPs during the DSP boot process. XBAR has a few other functions, including receiving incoming phone rings and controlling phone line hook status.

Each DSP subsystem consists of an Analog Devices ADSP2101 processor and 32K by 24 bits of external 20-ns

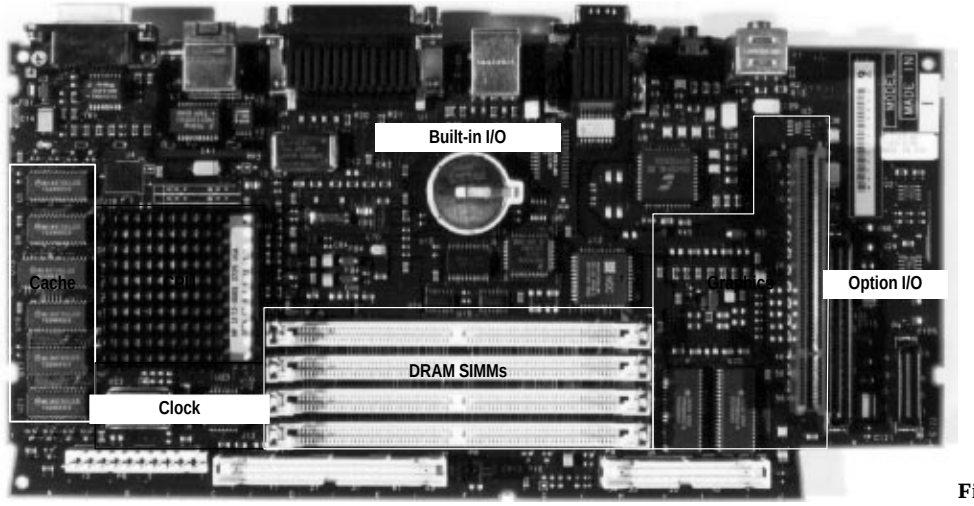


Fig. 3. The Model 712 system board.

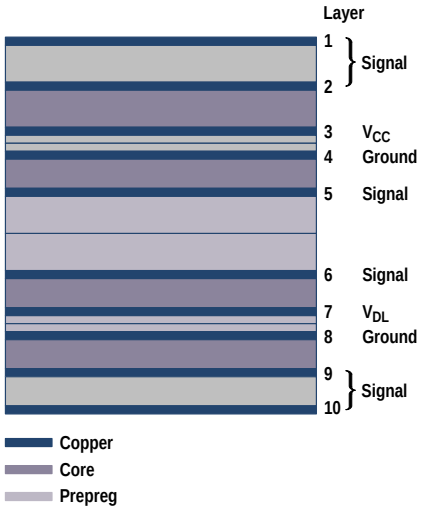


Fig. 4. The Model 712 system board construction.

SRAM for DSP programs and data. Each processor has two serial ports, one for XBAR and the other for the Analog Devices AD28mps01 analog front end (phone CODEC). Each phone CODEC connects to a standard two-wire telephone line through a Silicon Systems Incorporated 73M9002 data access arrangement, which provides the isolation circuitry required by communications regulatory agencies.

The telephony card is described in more detail in the article on page 69.

Printed Circuit Board Design

The Model 712 system contains a single printed circuit board called the system board. Fig. 3 shows a photograph of the system board. The system board supports all the functionality of the Model 712 system except for the optional boards and peripherals.

The system board is 10 layers deep, and has 0.005-inch traces and spaces. It measures 11.4 inches by 5.6 inches and uses double-sided surface mount technology.

The board construction shown in Fig. 4 was designed with the printed circuit board vendor to ensure that the least costly materials were chosen to obtain the necessary electrical parameters. Although it is designed to exhibit specific trace impedances, the blank printed circuit board is not a controlled-impedance design, which saves cost. The finished board size is optimized to make the best use of standard subpanel sizes used by the printed circuit board vendor. Although the board does use 0.005-inch traces and spaces, these minimum geometries are used only when necessary. Whenever possible, less aggressive routing is used to help with board yield and to keep down the cost of the board.

The design of the blank printed circuit board presented a number of technical challenges and some cost-saving opportunities.

Performance Challenges. The clock and cache layouts presented some very special challenges in designing the printed circuit board.

Fig. 5 shows a simplified block diagram of the clock circuit used in the Model 712. All ECL circuitry is powered from the V_{cc} supply, and all clock receivers in the VLSI are designed to operate at these shifted ECL voltage levels. This saves the cost of additional supply voltages and level translators. The master clock is first buffered, and multiple copies are routed to the receiving VLSI. This way, the delay to each device can be independently controlled to minimize clock skew and maximize system performance. Clocks are all routed on inner layers, where propagation delay is better controlled because of the trace's stripline nature. The clocks are driven as differential pairs and are routed to each other to minimize differential noise generation and susceptibility. The clock circuitry also features an interesting termination scheme. This pi-termination network is designed to approximate the same load as other more traditional termination schemes. However, it has the advantage of using zero supply current and fewer parts.

Fig. 6 shows a conceptual representation of how the cache is routed. The cache line is routed to minimize cache address drive delay. This arrangement also cuts down on the number of vias and maintains an unbroken ground plane. Address lines are routed from the CPU to the first via split on inner layers, where the impedance is close to half that of the outer layers. This is to better match the impedance of the traces on the two outer layers, which are essentially in parallel.

EMC and EMI Control. In addition to more traditional methods of EMC and EMI control, the Model 712 system board uses features built into the blank printed circuit board to mimic

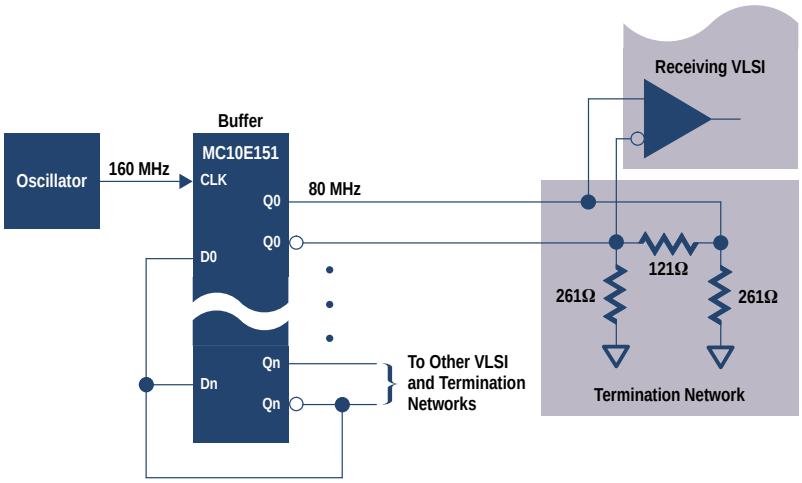


Fig. 5. The clock circuit used in the Model 712 system.

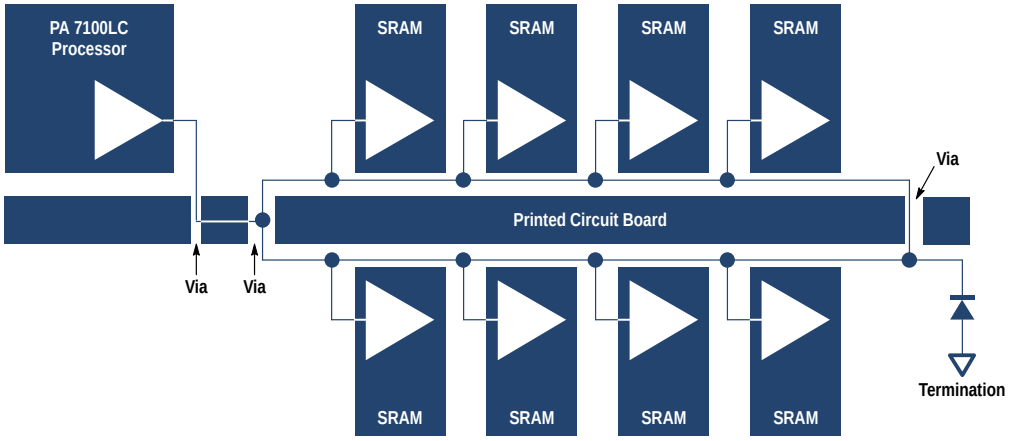


Fig. 6. A conceptual representation of the cache layout in the Model 712.

the functionality of equivalent discrete designs. However, since they are built into the printed circuit board their benefits are essentially free.

Small spark gaps are placed near many of the connectors to help control ESD. These spark gaps are simply very small trace segments separated at minimum geometries to provide a shunt path for ESD energy from signal to ground.

To control RFI, the printed circuit board makes use of a number of buried capacitors. Buried capacitors are essentially small capacitors whose plates are all or part of the printed circuit board's signal or ground layers. The dielectric material of the printed circuit board serves to separate the plates of the capacitors. Each power plane is effectively bypassed to ground by placing a ground plane in close proximity to it. Furthermore, some signals are also bypassed to ground with small buried capacitors to shunt unwanted RFI energy to ground.

Conclusion

By taking the approach of designing from the ground up, the Model 712 hardware designers have optimized each part of the design to work together to provide outstanding performance at very low cost. Designing the VLSI components as a set balanced performance and cost and also simplified the interfaces between the devices. By building in the features wanted by most customers and making less common features available only on low-cost option boards, the system cost is minimized for most customers.

The Model 712 system performance is summarized in Table II.

Table II
Model 712 Performance

Specification	712/60	712/80
SPECint92	58.1	84.3
SPECfp92	85.5	122.3
MFLOPS(DP)	12.8	30.6
AIM APR II	44.5	73.8

Acknowledgments

This paper would not have been possible without the help of Rob Horning, Mike Diehl, Jeff Hargis, Paul Tucker, Steve Scheid, and Howell Felsenthal.

The design of the Model 712 hardware was a team effort, and many people are to be thanked for its success. Special thanks to the firmware team including Jeff Kehoe and Doug Feller, whose innovations helped keep the hardware simple. Thanks to the R&D team in Fort Collins, Colorado including Jim McLucas and James Murphy. Thanks to the R&D team in Cupertino, California including Alan Wiemann, Wayne Ashby, Sharon Ebner, Maria Lines, Danny Lu, Rob Snyder, Steve La Mar, Robert Lin, Daniel Li, Rayka Mohebbi, Pat McGuire, Jean Lundeen, Jeff Swanson, and Paul Rogers. Thanks also to the teams who designed the VLSI chips that were crucial to the success of the project including the PA 7100LC design team, the LASI design team, and the graphics chip design team, led by Paul Martin. Thanks to Spence Ure's manufacturing team for their insights. Thanks to the marketing team of Barry Crume, Steve Johnson, and Evan James for providing focus. Finally, thanks to the guidance of Cliff Loeb and Joe Fucetola, and to the vision of Denny Georg.

References

1. Patrick Knebel, et al, "HP's PA7100LC: A Low-Cost Superscalar PA-RISC Processor," *Compcon Digest of Papers,* February 1993, pp. 441-447.
2. Steve Undy, et al, "A Low-Cost Graphics and Multimedia Workstation Chip Set," *IEEE Micro,* April 1994, pp. 10-22.
3. C. Dowdell and L. Thayer, "Scalable Graphics Enhancements for PA-RISC Workstations," *Compcon Digest of Papers,* February 1992, pp. 122-128.
4. Tom Spencer, et al, "A Workstation I/O System on a Chip," *Compcon Digest of Papers,* February 1994.

A Low-Cost, High-Performance PA-RISC Workstation with Built-In Graphics, Multimedia, and Networking Capabilities

Designing as a set the three VLSI components that provide the core functions of CPU, I/O, and graphics for the HP 9000 Model 712 workstation balanced performance and cost and simplified the interfaces between components, allowing designers to create a system with high performance at a low cost.

by Roger A. Pearson

Designing a workstation entails defining various functional blocks to work together to provide a set of features at a desired level of performance at the lowest possible cost. Often, many parts of the design are leveraged from previous designs, and only new functionality is designed from scratch. This approach may save development costs, but could result in a product that is more costly to build.

When one component of the system design has performance that can't be taken advantage of, whether because of architecture limitations or other components' performance limitations, then the system design suffers by having to carry the cost of that unused performance. By designing with the total system in mind, so that all components of the design are optimized to work together with no wasted performance, cost can be minimized. The designers of the HP 9000 Series 700 Models 712/60 and 712/80 took this approach to offer a high-performance combination of graphics, multimedia, and networking capabilities at new low prices. The objectives of the new design included:

- Providing the high performance of a PA-RISC workstation at the lowest possible cost
- Improving the performance and capabilities of multimedia functions through simple extensions to the instruction set
- Enabling an extensive set of communication features through low-cost option cards
- Designing for high-volume manufacturing.

Instrumental in meeting these objectives was the decision to design three new custom VLSI chips together, as a set, to achieve new levels of price/performance for the core functions of CPU, I/O, and graphics.

Overview

Three new VLSI chips provide most of the functionality of the Model 712 workstation. The PA 7100LC CPU chip interfaces directly to the cache and main memory. The LASI (LAN/SCSI) chip does most of the core I/O needed for entry-level workstations. The graphics subsystem consists of the graphics chip and the frame buffer VRAMs. All three chips communicate through the GSC (general system connect) bus. Fig. 1 shows a block diagram of the Model 712 system.

The Models 712/60 and 712/80 are very similar and differ only in their cache sizes and cache speeds and in the main system clock speeds.

The Processor

The compute power of the Model 712 system is provided by the PA-RISC PA 7100LC processor,[1,2] which is packaged in a 432-pin ceramic PGA. The CPU design was optimized for the Model 712 and includes the following features:

- Superscalar CPU
- 1K-byte instruction buffer
- Multimedia support
- Cache control for up to 2M bytes of external cache
- ECC (error correction coding) memory controller

The clock frequencies of the Model 712/60 and the Model 712/80 are 60 MHz and 80 MHz respectively. The PA 7100LC is described in more detail in the article on page 12.

Cache

The PA 7100LC CPU uses an external cache. An external cache allows system designers to change the size of the cache easily to meet their performance and cost goals. Furthermore, off-chip cache provides all the performance necessary, without limiting the CPU frequency.

The external cache is 64K bytes on the Model 712/60 and 256K bytes on the Model 712/80 and is logically split into equal halves for the instruction and data caches. Combining the caches saved pins on the CPU. To further reduce costs, industry-standard SRAMs (static RAMs) are used. Table I shows the SRAMs used in the Model 712 systems.

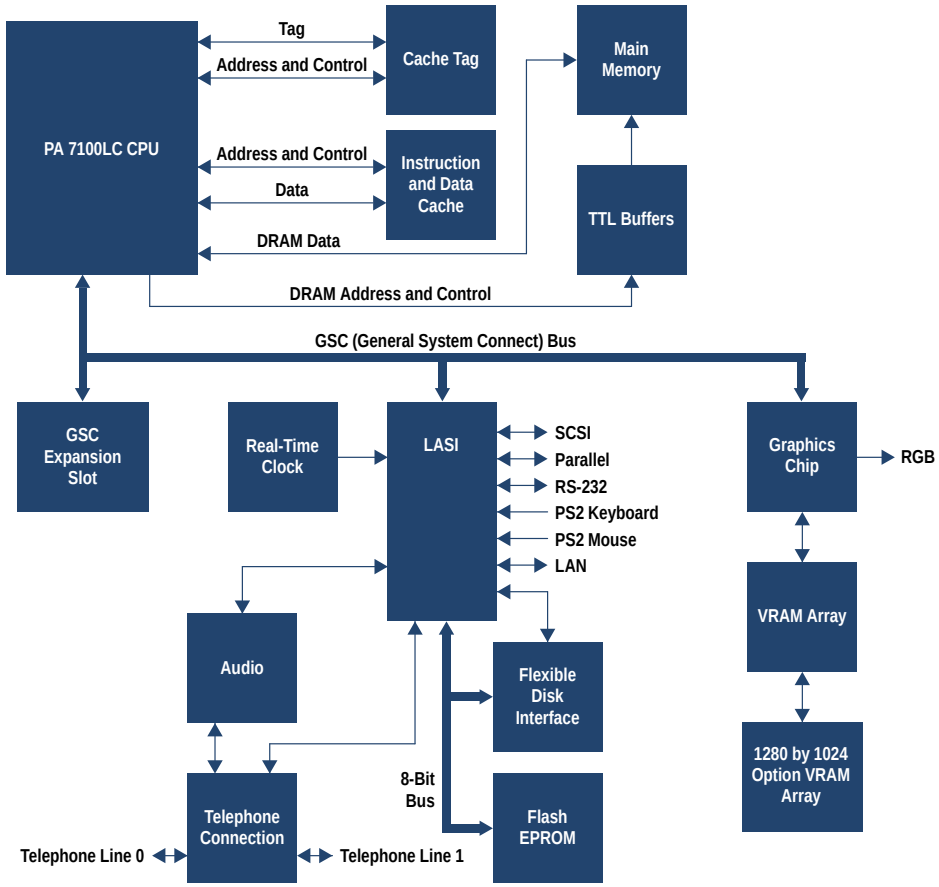


Fig. 1. Block Diagram of the HP 9000 Model 712 hardware.

Table I
Static RAMS Used in the Model 712 Systems

Model	Function	Size	Speed	Quantity
712/60	Tag	8K bytes	12 ns	4
	Data	8K bytes	12 ns	6
	Data	8K × 9 bits	12 ns	2
712/80	Tag	32K bytes	10 ns	4
	Data	32K bytes	10 ns	6
	Data	32K × 9 bits	10 ns	2

Main Memory

The main memory for the Model 712 systems has been engineered to provide high performance with industry-standard 70-ns SIMMs (single inline memory modules). Currently supported SIMMs are available in 4M-, 8M-, 16M-, and 32M-byte sizes. Four slots are available and must be filled in pairs for a maximum of 128M bytes.

The Model 712's main memory design minimizes the average cache miss penalty. The main memory controller returns double words (eight bytes, since a word is four bytes) back to the CPU. Each cache line is made up of four double words. When there is a cache miss, the one double word of the four in the cache line that was missed is referred to as the critical word. To minimize the miss penalty, the double word containing the critical word is sent back to the CPU first, followed by the remaining three double words.

Bandwidth is maximized by using fast page mode when consecutive accesses reside on the same page. This is often the case when large blocks of memory are accessed and is very common in windowed graphics systems.

The General System Connect Bus

The general system connect, or GSC, is the local bus that connects the three VLSI devices and the optional I/O card. The GSC bus is designed to provide maximum bandwidth for memory-to-graphics transfers. The bus has 32-bit multiplexed address and data lines to minimize the number of signals. Other features of the bus include:

- Operation at half the CPU frequency (30 or 40 MHz)
- Support for 1-, 2-, 4-, 8-, 16-, or 32-byte transactions
- Central arbitration
- Parity generation and checking.

Normally, bus transactions are terminated by a turnaround state that allows drivers to be turned off before the drivers for the next transaction are turned on. To improve graphics performance, the bus supports back-to-back writes to the same device without the turnaround state. This improves throughput on transfers of large blocks of data from main memory to graphics.

During transfers from memory to I/O, it is sometimes necessary to lock the CPU out of memory (e.g., when semaphores are used). To facilitate this, the GSC bus provides a locking mechanism, which prevents the CPU from accessing memory (to service a cache miss, for example).

Graphics

The graphics subsystem consists of a graphics chip and four on-board VRAMs (video RAMs), which provide a 1024-by-768-

pixel frame buffer with a depth of eight planes at a refresh rate of 72 Hz. An optional high-resolution VRAM board increases resolution to 1280 by 1024 pixels.

The graphics chip was designed with the other system components to provide high performance at a minimal cost. For more information on the graphics chip, see reference 3 and the article on page 43.

Built-in I/O

The Model 712 features a number of built-in I/O devices that are intended to address the needs of the majority of users.

Support for these functions is provided largely by the LASI I/O VLSI chip. LASI is a highly integrated chip that provides a significant reduction in system cost and increased reliability. The chip is packaged in a 240-pin MQUAD package. The LASI chip is described in more detail in the article on page 36 and in reference 4.

The following sections briefly describe the LASI chip's built-in capabilities.

IEEE 802.3 LAN. LASI contains an Intel 82C596 megacell which was ported to work with HP's IC process. The LAN transceiver, which was not practical to include on LASI, is loaded on the printed circuit board. The transceiver interfaces to both the AUI (attachment unit interface) and Ethertwist media.

SCSI. The Model 712 uses an 8-bit single-ended SCSI interface for the optional internal hard drive and external peripherals. The SCSI-2 interface is implemented entirely within LASI through a megacell that was designed by HP and NCR. A netlist for the NCR 53C710 was imported into HP's design environment. The design was then tuned to work in HP's IC process.

By keeping the SCSI bus stub length to a minimum on the printed circuit board and on the connection to the optional internal drive, SCSI termination on the internal side is greatly simplified. Short stub lengths allow the bus to be terminated on the printed circuit board, whether the optional internal drive is present or not. This saves cost by obviating the need for special terminators which would otherwise have to be enabled or disabled (manually or electrically), depending on the presence or absence of the optional internal drive.

Audio. 16-bit CD-quality audio playback and record capability is provided by the audio circuitry, which consists of a Crystal Semiconductor CS4216 CODEC and supporting circuitry. The LASI chip also includes the serial interface to the CS4216. Headphone, microphone, and line-in connectors are located on the rear panel. Standard sampling rates include 8, 44.1, and 48 kHz.

Real-Time Clock. A real-time clock is designed into the LASI chip. Battery backup keeps time while the workstation is powered down.

PS/2. There are two PS/2 connectors on the rear panel that allow connection to a low-cost industry-standard keyboard and mouse. The PS/2 interface circuitry is integrated into the LASI chip.

RS-232. An RS-232 interface has also been designed into the LASI chip. The Model 712 buffers the signals with a MAXIM 211 to provide an RS-232 serial port. LASI buffers inbound and outbound data with 16-byte FIFOs, at baud rates from 50 to 454 kbits/s.

Parallel. The LASI chip also provides a parallel port conforming to the Centronics industry standard.

Flexible Disk Support. A Western Digital WD37C65C flexible disk controller interfaces LASI to an optional internal personal-computer-style flexible disk drive.

Flash EPROM. An 8-bit bus on the LASI chip is demultiplexed by two 74CHT374 latches to provide the address and data lines necessary to address the two 128K-byte flash EPROMs that contain the boot firmware. The flash EPROMs are also used to store configuration parameters, eliminating the need for an EEPROM and its associated cost.

I/O System Support. LASI provides a number of miscellaneous I/O system support functions, including:

- Clock generation. LASI derives all the necessary clocks required by the I/O circuitry from the main system clock. It does so by using simple divide-by-n counters and two digital phase-locked loops.
- System arbitration support. LASI arbitrates GSC bus requests from the I/O devices within LASI, as well as from the CPU and optional expansion card.
- Interrupt support. LASI also provides and manages external interrupt capability for the various I/O devices.

Optional I/O

For those users who need functionality beyond that provided by the built-in I/O, the Model 712 includes two personality slots that can be configured with a variety of other I/O functions. The first of these slots is referred to as the expansion slot and includes a connection to the GSC bus. The second slot provides a connection to the serial audio stream, and is intended for telephone functions. This slot is called the telephony slot.

Expansion Cards. Expansion cards are optional cards that connect directly to the GSC bus to provide a variety of other I/O functions.

Since LASI has a configurable address space and can be configured as an arbitration slave, many of the expansion cards rely on a second LASI chip to implement much of their functionality.

The following optional expansion cards are provided for the Model 712:

- Second serial port. The second serial port card uses its own LASI chip and support circuitry identical to that on the system board to provide an additional RS-232 port.
- Second LAN AUI and second serial interface. This card also uses a LASI chip and circuitry similar to that on the system board to add an additional IEEE 802.3 LAN with an attachment unit interface (AUI) and a second RS-232 interface.
- X.25 and second serial interface. A Motorola 68302 multiprotocol processor interfaced to the 8-bit bus of a slave LASI provides X.25 networking to a 25-pin X.21bis port for speeds of 1.2 kbits/s to 19.2 kbits/s. The second RS-232 serial interface is implemented in the same fashion as the other cards.
- Second display. A second display can be added to the system with the second display card. This card duplicates the

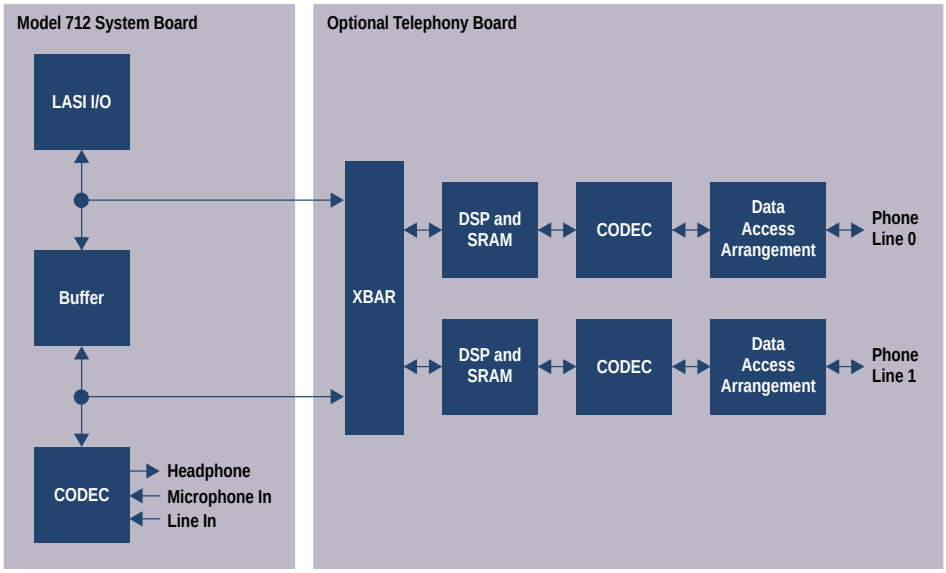


Fig. 2. Block diagram of the Model 712 audio and telephony circuits.

graphics functionality that is already built into the system board by replicating the graphics chip and its supporting circuitry.

- Token Ring/9000. The Token Ring/9000 card provides IEEE 802.5 LAN functionality through the use of a Texas Instruments token ring controller chip and a custom ASIC that provides the GSC interface. Unshielded and shielded twisted pair connections are provided at data rates from 4 Mbits/s to 16 Mbits/s.
- Second display and second LAN AUI/RS-232. This option combines the features of the second graphics display and the second LAN AUI/RS-232 options. Since the circuitry for this option would not fit on a single expansion slot card, some of the circuitry resides on a daughter card that is connected to the expansion slot card. The daughter card gets power and mechanical support through the telephony connector, so when this option is installed, the telephony option is not available.

Telephony. The telephony card installs in the telephony slot and provides two lines of telephone access. Each of the lines can be configured to support voice, data modem, or fax modem.

The system board's headset and microphone serve as the human interface for voice telephony, and an interface chip on the telephony card called XBAR links the system board's audio circuitry to the telephony functions (see Fig. 2).

This arrangement allows recording and playback during telephone conversations. It also supports digital mixing of microphone, line-in, telephone, and prerecorded audio. Caller-ID decoding is supported, as are DTMF (dual-tone multifrequency) encoding and decoding, and dual-line conferencing.

The XBAR chip serves to route information between the LASI I/O chip, the audio CODEC, and the DSP blocks in a variety of programmable ways. Data is transferred to and from the system board through two serial data paths. Two additional serial paths send and receive data to and from the DSPs. Two 8-bit parallel ports are used by the DSPs during the DSP boot process. XBAR has a few other functions, including receiving incoming phone rings and controlling phone line hook status.

Each DSP subsystem consists of an Analog Devices ADSP2101 processor and 32K by 24 bits of external 20-ns

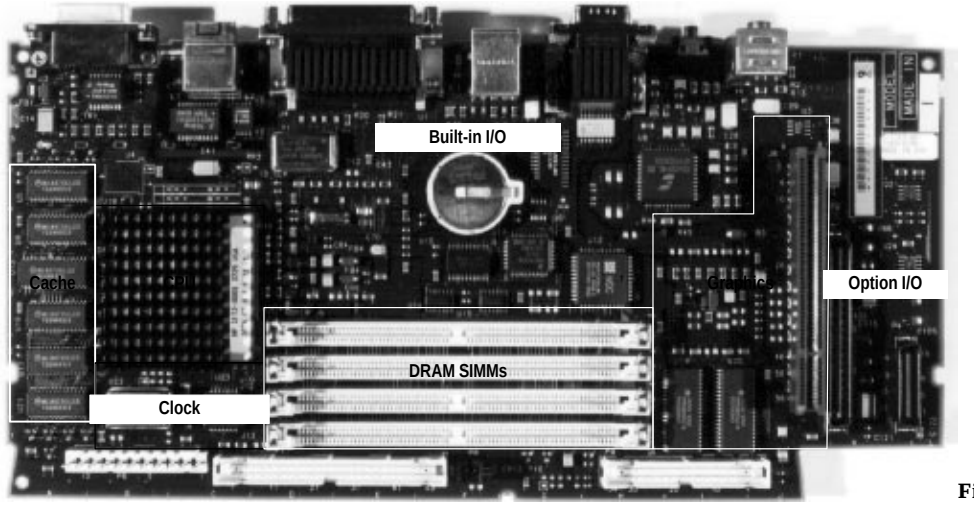


Fig. 3. The Model 712 system board.

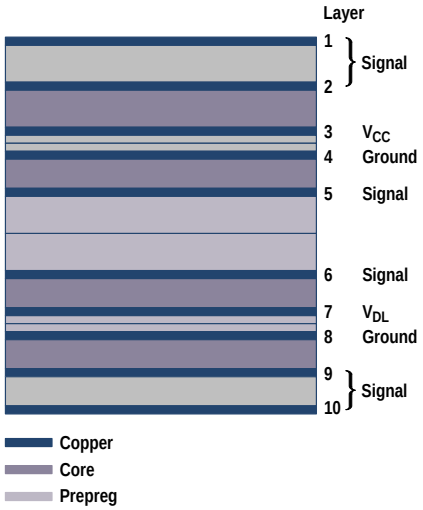


Fig. 4. The Model 712 system board construction.

SRAM for DSP programs and data. Each processor has two serial ports, one for XBAR and the other for the Analog Devices AD28mps01 analog front end (phone CODEC). Each phone CODEC connects to a standard two-wire telephone line through a Silicon Systems Incorporated 73M9002 data access arrangement, which provides the isolation circuitry required by communications regulatory agencies.

The telephony card is described in more detail in the article on page 69.

Printed Circuit Board Design

The Model 712 system contains a single printed circuit board called the system board. Fig. 3 shows a photograph of the system board. The system board supports all the functionality of the Model 712 system except for the optional boards and peripherals.

The system board is 10 layers deep, and has 0.005-inch traces and spaces. It measures 11.4 inches by 5.6 inches and uses double-sided surface mount technology.

The board construction shown in Fig. 4 was designed with the printed circuit board vendor to ensure that the least costly materials were chosen to obtain the necessary electrical parameters. Although it is designed to exhibit specific trace impedances, the blank printed circuit board is not a controlled-impedance design, which saves cost. The finished board size is optimized to make the best use of standard subpanel sizes used by the printed circuit board vendor. Although the board does use 0.005-inch traces and spaces, these minimum geometries are used only when necessary. Whenever possible, less aggressive routing is used to help with board yield and to keep down the cost of the board.

The design of the blank printed circuit board presented a number of technical challenges and some cost-saving opportunities.

Performance Challenges. The clock and cache layouts presented some very special challenges in designing the printed circuit board.

Fig. 5 shows a simplified block diagram of the clock circuit used in the Model 712. All ECL circuitry is powered from the V_{cc} supply, and all clock receivers in the VLSI are designed to operate at these shifted ECL voltage levels. This saves the cost of additional supply voltages and level translators. The master clock is first buffered, and multiple copies are routed to the receiving VLSI. This way, the delay to each device can be independently controlled to minimize clock skew and maximize system performance. Clocks are all routed on inner layers, where propagation delay is better controlled because of the trace's stripline nature. The clocks are driven as differential pairs and are routed to each other to minimize differential noise generation and susceptibility. The clock circuitry also features an interesting termination scheme. This pi-termination network is designed to approximate the same load as other more traditional termination schemes. However, it has the advantage of using zero supply current and fewer parts.

Fig. 6 shows a conceptual representation of how the cache is routed. The cache line is routed to minimize cache address drive delay. This arrangement also cuts down on the number of vias and maintains an unbroken ground plane. Address lines are routed from the CPU to the first via split on inner layers, where the impedance is close to half that of the outer layers. This is to better match the impedance of the traces on the two outer layers, which are essentially in parallel.

EMC and EMI Control. In addition to more traditional methods of EMC and EMI control, the Model 712 system board uses features built into the blank printed circuit board to mimic

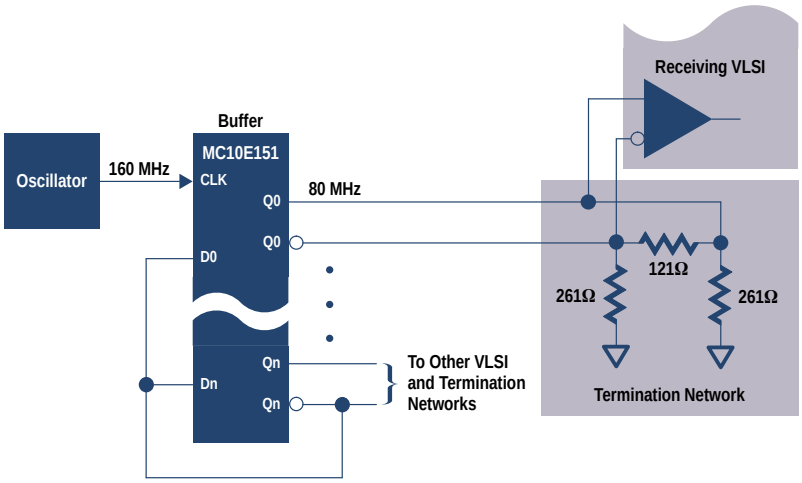


Fig. 5. The clock circuit used in the Model 712 system.

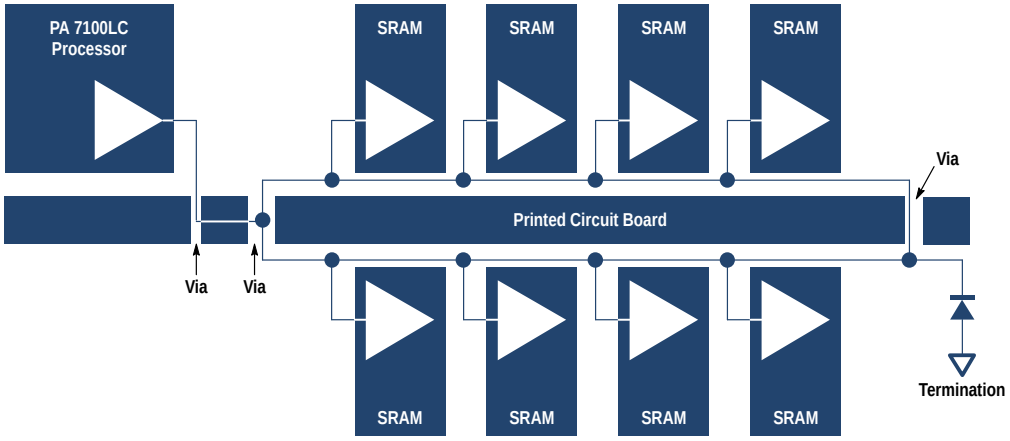


Fig. 6. A conceptual representation of the cache layout in the Model 712.

the functionality of equivalent discrete designs. However, since they are built into the printed circuit board their benefits are essentially free.

Small spark gaps are placed near many of the connectors to help control ESD. These spark gaps are simply very small trace segments separated at minimum geometries to provide a shunt path for ESD energy from signal to ground.

To control RFI, the printed circuit board makes use of a number of buried capacitors. Buried capacitors are essentially small capacitors whose plates are all or part of the printed circuit board's signal or ground layers. The dielectric material of the printed circuit board serves to separate the plates of the capacitors. Each power plane is effectively bypassed to ground by placing a ground plane in close proximity to it. Furthermore, some signals are also bypassed to ground with small buried capacitors to shunt unwanted RFI energy to ground.

Conclusion

By taking the approach of designing from the ground up, the Model 712 hardware designers have optimized each part of the design to work together to provide outstanding performance at very low cost. Designing the VLSI components as a set balanced performance and cost and also simplified the interfaces between the devices. By building in the features wanted by most customers and making less common features available only on low-cost option boards, the system cost is minimized for most customers.

The Model 712 system performance is summarized in Table II.

Table II
Model 712 Performance

Specification	712/60	712/80
SPECint92	58.1	84.3
SPECfp92	85.5	122.3
MFLOPS(DP)	12.8	30.6
AIM APR II	44.5	73.8

Acknowledgments

This paper would not have been possible without the help of Rob Horning, Mike Diehl, Jeff Hargis, Paul Tucker, Steve Scheid, and Howell Felsenthal.

The design of the Model 712 hardware was a team effort, and many people are to be thanked for its success. Special thanks to the firmware team including Jeff Kehoe and Doug Feller, whose innovations helped keep the hardware simple. Thanks to the R&D team in Fort Collins, Colorado including Jim McLucas and James Murphy. Thanks to the R&D team in Cupertino, California including Alan Wiemann, Wayne Ashby, Sharon Ebner, Maria Lines, Danny Lu, Rob Snyder, Steve La Mar, Robert Lin, Daniel Li, Rayka Mohebbi, Pat McGuire, Jean Lundeen, Jeff Swanson, and Paul Rogers. Thanks also to the teams who designed the VLSI chips that were crucial to the success of the project including the PA 7100LC design team, the LASI design team, and the graphics chip design team, led by Paul Martin. Thanks to Spence Ure's manufacturing team for their insights. Thanks to the marketing team of Barry Crume, Steve Johnson, and Evan James for providing focus. Finally, thanks to the guidance of Cliff Loeb and Joe Fucetola, and to the vision of Denny Georg.

References

1. Patrick Knebel, et al, "HP's PA7100LC: A Low-Cost Superscalar PA-RISC Processor," *Compcon Digest of Papers,* February 1993, pp. 441-447.
2. Steve Undy, et al, "A Low-Cost Graphics and Multimedia Workstation Chip Set," *IEEE Micro,* April 1994, pp. 10-22.
3. C. Dowdell and L. Thayer, "Scalable Graphics Enhancements for PA-RISC Workstations," *Compcon Digest of Papers,* February 1992, pp. 122-128.
4. Tom Spencer, et al, "A Workstation I/O System on a Chip," *Compcon Digest of Papers,* February 1994.

The PA 7100LC Microprocessor: A Case Study of IC Design Decisions in a Competitive Environment

Engineering design decisions made during the early stages of a product's development have a critical impact on the product's cost, time to market, reliability, performance, and success.

by Mick Bass, Patrick Knebel, David W. Quint, and William L. Walker

In today's competitive microprocessor market, successful design teams realize that flawless execution of product development and delivery is not enough to ensure that a product will succeed. They understand that defining the correct feature set for a product and creating design methodologies appropriate to implement and verify that feature set are just as important as meeting the product schedule.

The design decisions that engineers and managers make while defining a new product have a critical impact on the product's cost, time to market, reliability, performance, future market demand, and ultimate success or failure. Engineers and managers must make trade-offs based on these factors to decide which features they should implement in a new product and which they should not. Further, they must plan their product development effort so that the methodologies by which they develop their product are sufficient to ensure that they are able to implement the product definition within the required cost, schedule, and performance constraints.

Design choices arose frequently while we were defining and implementing the PA 7100LC microprocessor.[1] We were targeting the PA 7100LC to be the processing engine of a new line of low-cost, functionally rich workstation and server products. Our design goals for the CPU were to provide the system performance required for our target market at an aggressively low system cost and to deliver the CPU on a schedule that would not delay what was to become HP's steepest computer system production ramp to date. Fig. 1 shows a simplified block diagram of the PA 7100LC processor.

To meet these goals required that we sometimes had to shift our focus from the CPU to the impact of a particular feature upon performance and cost at the system level. Hewlett-Packard's position as a vendor of both microprocessors and computer systems allowed us to use this technique with much success.[2,3] Even with this focus, however, the correct

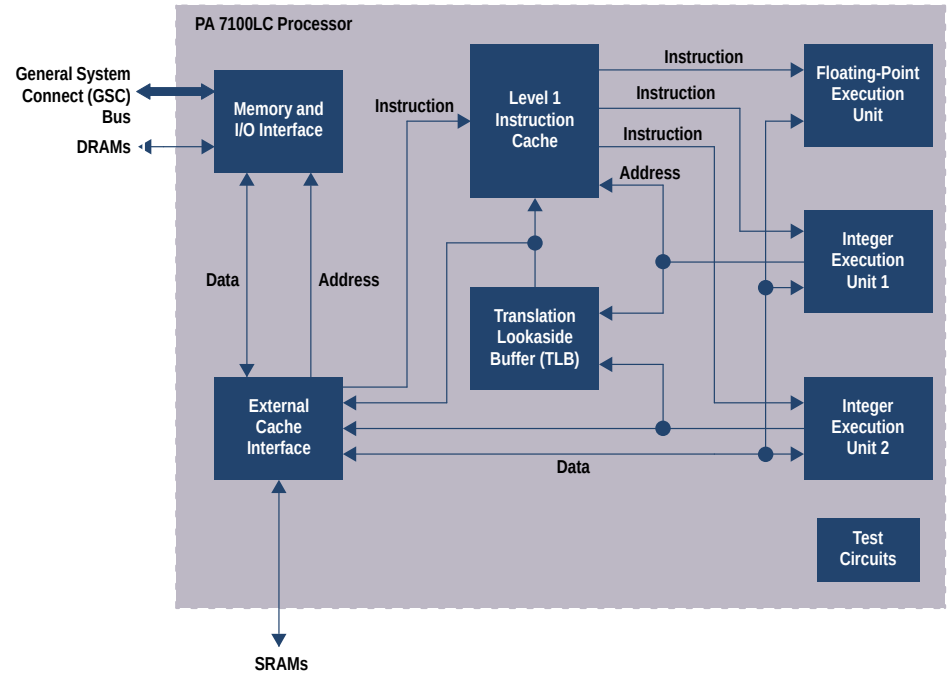


Fig. 1. A simplified block diagram of the PA 7100LC processor.

decision could be far from obvious. We often identified several alternative implementations of a particular feature, each with its own impact on cost, schedule, and performance. Trading these impacts against one another proved very challenging. Design decisions also impacted each other, with the outcome of one serving as a critical input to others. The effect of a decision, for this reason, was sometimes much larger than would have appeared at first glance. Sometimes decisions created additional requirements, either for new features or for new support methodologies. All of these factors played together to underscore the fact that it was critical to our product's success to have a decision process that worked well.

We knew that a good definition of the PA 7100LC would require that we make feature decisions in several areas, including:

- Cache organization
- Number of execution units and superscalar design
- Pipeline organization
- Floating-point functionality
- Package technology
- Degree of integration
- Multimedia enhancements.

We also knew that we needed to select development methodologies consistent with the feature decisions that we made. Product features and required design methodologies are often strongly connected. We couldn't consider the benefits of one without the costs of the other, and vice versa. Methodologies that were impacted by our feature-set decisions included:

- Synthesis
- Place and route
- Behavioral simulation
- Presilicon functional verification
- Postsilicon functional and electrical verification
- Production test.

These methodologies are discussed in the article on page 23.

The cumulative effects of our decisions led to the creation of a low-cost, single-chip processor core that includes a built-in memory controller, a combined, variable-size off-chip primary instruction and data cache, a 1K-byte on-chip instruction buffer, and a superscalar execution unit with two integer units and one floating-point unit. We reduced the size and performance of the floating-point unit, which we had leveraged from the PA 7100 processor.[4,5] We added I_{DDQ}, sample-on-the-fly, and debug modes to enhance testability, reduce test cost, and accelerate the postsilicon schedule. We tailored the methodologies by which we created the chip to match the features that we had decided upon.

This article provides examples of our decision-making process by exploring the decisions that we made for several of the features listed above. In each case, we present the alternatives that we considered, the costs and benefits of each, and the impact on other features and methodologies. We discuss our decision criteria. Since we strive to continually improve our ability to make good design decisions, we also present, wherever possible, a bit of hindsight about the process. In most cases, we still believe that we selected the correct alternative. However, if this is not the case, we discuss what we have learned and the modifications we made to our process to incorporate this new knowledge.

The Design Decision Process

Most design decisions ultimately come down to trade-offs between cost, schedule, and performance. Unfortunately, it is often difficult to determine the true cost, schedule, or performance for the wide variety of implementations that are possible. And since these three factors most often play against each other, it is necessary to make sacrifices in one or two of the areas to make gains in the others.

The cost of a processor core is determined by the cost of silicon die, packaging, wafer testing, and external SRAM and DRAM. Breaking this down, we find that cost of a die is determined by the initial wafer cost and the defect density of the IC process being used. Wafers are more expensive for more advanced processes because of higher equipment, development, and processing costs. The die packaging costs are determined primarily by package type and pin count. Large-pinout packages can be very expensive. An often ignored cost is the tester time required to determine that a manufactured part is functional. Reducing the time needed for wafer and package testing directly reduces costs. Finally, SRAM and DRAM costs are determined by the number, size, and speed of the parts needed to complete the design.

The schedule of a project is determined by the complexity of the design and the ability to leverage previous work. Each design feature requires certain time investments and has associated risks. Time is required for preliminary feasibility investigations, design of control algorithms, implementation of circuits, and presilicon and postsilicon verification. Schedule risks include underestimation of time requirements because of unexpected complexity and the extra chip turns required to fix postsilicon bugs associated with complex design features.

Performance is conceptually simple, but because of the intricacy of processor design it is often difficult to measure without actual prototypes. HP has invested heavily in performance simulation and analysis of its designs. Results from HP's system performance lab were invaluable in making many of the design decisions for the PA 7100LC. By supporting a detailed simulation model of each processor developed by HP, the system performance lab is able to provide quick feedback about proposed changes. HP also uses these models after silicon is received to help software developers (especially for compilers and operating systems) determine bottlenecks that limit their performance.

Engineers at the system performance lab design their processor simulators in an object-oriented language to allow easy leverage between implementations. All processor features that affect performance are modeled accurately by close teamwork between the performance modeling groups and the hardware design groups. As the hardware group considers a change to a design, the change is made in the simulator, and simulations are done to allow simple comparisons that differ by only a single factor. This is continued in an iterative fashion until all design decisions have been made,

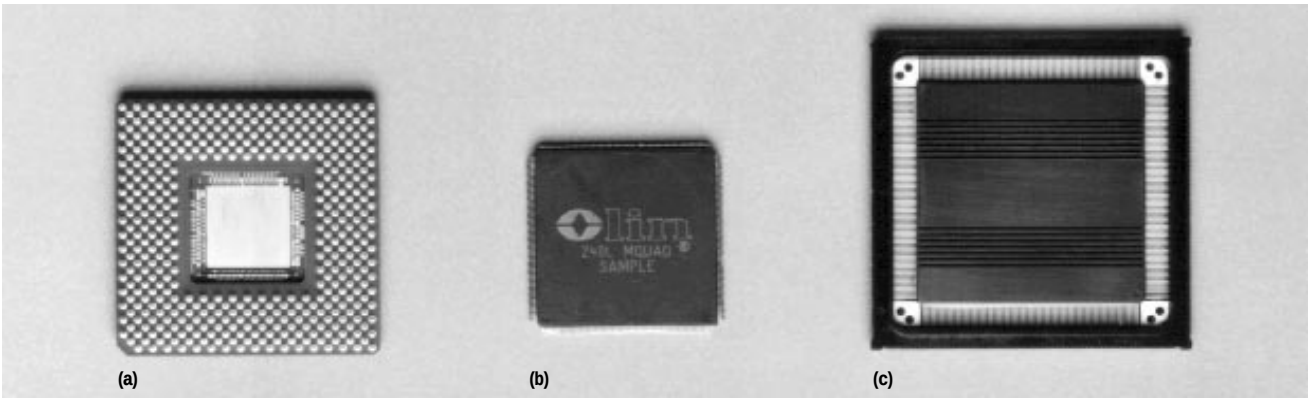


Fig. 2.* (a) 432-pin ceramic pin grid array (432-CPGA). (b) A 240-pin MQUAD and (c) a 304-pin MQUAD.

* The CPGA package is manufactured by Kyocera Inc. and the MQUAD packages are manufactured by Olin Interconnect Technologies.

after which we are left with a simulator that matches the hardware to be built.

Without performance simulations, it would be very difficult to estimate performance for a proposed implementation. Even something as simple as a change in operating frequency has effects that are difficult to estimate because of the interactions between fixed memory access times and processor features. As processor frequency increases, memory latencies increase, but this increased latency is sometimes (but not always) hidden by features such as stall-on-use. Stall-on-use allows the processor to continue execution in the presence of cache misses as long as the data is not needed for an operation. These interactions make accurate hand calculations impossible, creating a need to use simulations for comparing many different implementation options.

The performance simulations are based on SPEC and TPC benchmarks. While these benchmarks are useful for gathering performance numbers and making comparisons, they do not tell the whole story. Many applications are not represented by the benchmarks, including graphics, multimedia, critical hand-coded operating system routines, and so on. When evaluating features related to these applications, we work directly with people in those areas to analyze the impact of any decisions. Often this involves analyzing by hand critical sections of the code (e.g., tight loops) to evaluate the overall performance gain associated with a feature. For the PA 7100LC, this was especially true for the multimedia features.

The ability to quantify the impact of proposed features on cost, schedule, and performance was paramount to our ability to make sound design decisions.

Integration

The first design decisions that we made were related to the high-level question "How highly integrated should we make the chip?" This led to the questions: Should we include an on-chip cache or not? If so, how large should it be? If we have an off-chip cache, how should we structure it? How should the CPU connect to memory and I/O? Should the memory controller be integrated or not?

The primary question was whether the CPU, cache, and memory system should live on a single die in a single package, or whether we should partition this functionality onto two or more chips.

The trade-offs involved in this decision were numerous. Die cost would increase for a multichip solution. Package cost would vary with the partitioning that we chose, as would package type and maximum pin count. Required signal-to-ground ratios would vary with package type, which would either limit the signal count or require more pins (at a higher cost). Performance, design complexity, and schedule risk would be greatly impacted by the partitioning decision.

To sort out these trade-offs, we started with a packaging investigation that quantified cost, performance, and risk for different packaging alternatives. This investigation yielded a preferred package: a 432-pin ceramic pin grid array see (Fig. 2a). This package, with its large signal count, could accommodate the extra interfaces required to include a memory controller, an I/O controller, and an external cache controller.

The memory controller and cache investigations were tightly coupled. Performance simulations always included features from both subsystems because small changes in the behavior of one subsystem could drastically affect the performance of the other. In the end we realized that the performance gains brought by an integrated memory controller enabled smaller, cheaper caches without sacrificing overall performance. This realization drove the development of the cache subsystem.

Package Selection and CPU Partitioning. We targeted the IC package design with the objective of minimizing system cost with little compromise in performance. The customary package for CPU chips is either a quad flat pack (QFP) or a pin grid array. The QFP is a plastic, low-profile package with gull-wing connections on four sides. The QFP is inexpensive and easy to mount on a printed circuit board and has gained acceptance rapidly for surface mounting to printed circuit boards. It has the disadvantage that the number of pins is limited. Pin counts above 200 are fragile and difficult to keep

coplanar for surface mounting. The package also has very limited ability to dissipate heat because the chip is encased in plastic. A recent improvement to this package sandwiches the chip between two pieces of aluminum, which can dissipate up to four watts of heat (ten watts with a heat sink). It was this metal quad, or MQUAD, that became a candidate for a low-cost package for our high-performance CPU. HP's package of choice for previous CPUs has been the ceramic pin grid array, a complex brick of aluminum oxide and tungsten built in layers and fired at 2000°C. The PGA used for the PA 7100 processor (the basis for the PA 7100LC) was a 504-pin design that incorporated the following advanced features:

- A tungsten-copper heat-conducting slug for superior thermal conductivity to the heat sink
- Ceramic chip capacitors mounted on the package for power bypassing
- Thin dielectric layers that minimized power supply inductance
- Use of 0.004-in vias internally (most are 0.008-inch).

This package performed its thermal and electrical duties very well, but its cost had always been an issue.

Our strategy to develop a low-cost CPU coupled chip partitioning options with the packaging options of using either two low-cost MQUAD packages or placing a single large chip in a PGA. The two-chip CPU could be placed in one 240-pin and one 304-pin MQUAD (see Figs. 2b and 2c). The other alternative was to place a larger integrated chip in a single 432-pin PGA (see Fig. 2a). The first cost estimate assumed that the PGA would be priced similarly to the 504-pin package. The total cost of both MQUAD chips was initially thought to be about 75% less than the PGA estimate. This would seem to indicate that the MQUAD would be the definite candidate to meet our low-cost goals. However, that perception changed as our investigation continued.

We didn't expect the MQUAD's electrical performance to match that of the PGA because the MQUAD we were considering had only one layer of signals and no ground planes. Ground planes can be used to shield signal traces from each other and reduce inductances of signals and power supplies. The PGA could incorporate several ground planes if necessary. On the other hand, the MQUAD package can only approach the shielding effect of the ground planes by making every other lead a ground, which severely limits the number of usable signals. Gaining a lower package price by using the MQUAD would require redesigning the I/O drivers specifically to reduce rise times and thereby control crosstalk and power supply noise.

The PA 7100 PGA's electrical performance exceeded the needs of this chip, so the strategy shifted to trading away excess performance to gain lower cost. The number of power and ground planes was reduced to two. The design was also modified to optimize performance without using package-mounted bypasses or thin dielectric layers. The PGA design was reduced to four internal metal layers with no bypassing, no thin dielectric layers, and no 0.004-in vias, all of which reduced cost compared to the 504-pin PGA mentioned above.

The power dissipation of the chips would also have been an issue for the MQUADs. Heat sinking to further improve the thermal resistance of the packages might have been required. CPU designs are often upgraded to higher clock speeds after first release, so if package heat dissipation is marginal, upgrade capability is jeopardized. (Typically, power dissipation is proportional to operating frequency.) The 504-pin PGA had already been used to dissipate 25 watts, which left an opportunity for cost-saving modifications. With the thermal margin in mind, two design changes were investigated, one to use a lower-cost copper-Kovar-copper laminate heat spreader, and the other to eliminate the heat spreader entirely. The first option was dismissed because of failures found during a low-temperature storage test. (The laminate heat spreader detached from the ceramic body because of a disparity in thermal expansion rates.) The second option was also dismissed when the thermal resistance of the ceramic carrier was found to be too high.

The time schedule for the completion of reliability testing and manufacturing feasibility studies had to be considered when evaluating the two technologies. The PGA was a mature technology with considerable experience behind it, and the time schedule and results of the testing could be determined with some certainty. The MQUAD was a new technology by contrast. The design was solid, but had several new features that were untested in terms of long-term reliability. Despite the strong desire to exploit new technology, the schedule risk was a significant factor.

By the time the partitioning decision was to be made, the PGA cost had shrunk to almost half of its original cost, the 304-pin MQUAD was presenting schedule risks, and both MQUADs had marginal power dissipation. Possibly most important, the PGA provided a robust solution with thermal and electrical margins. The cost difference was still significant, but the PGA provided a flexibility to the chip designers that offset its disadvantages. Thus, the PGA package was chosen for the PA 7100LC.

Memory Controller Destiny. Whether or not to integrate the memory and I/O controllers onto the CPU die was one of the most direction-forming decisions that we made. To decide correctly, we had to consider the effects of integration on factors such as multiprocessor capability, system complexity, memory and I/O controller design complexity, die cost, memory system performance, and memory system flexibility.

Traditional multiprocessor systems have a single main memory controller and I/O controller (see Figs. 3a and 3b). These controllers maintain connections to the multiple processors. Systems organized in this way separate the memory and I/O controllers from the CPU. This organization allows users to upgrade entry-level systems to include multiple processors at the expense of reducing the memory and I/O performance of uniprocessor systems and adding significant complexity to both the memory and cache controllers.

Our design goals focused on maximizing uniprocessor performance. HP was already shipping desktop multiprocessor systems built around the PA 7100 microprocessor at the time we were making these decisions. The market segment that we were targeting for the PA 7100LC demanded peak uniprocessor performance at a low system cost. Since our target market didn't require multiprocessing as a system option, we directed our efforts toward the benefits that we

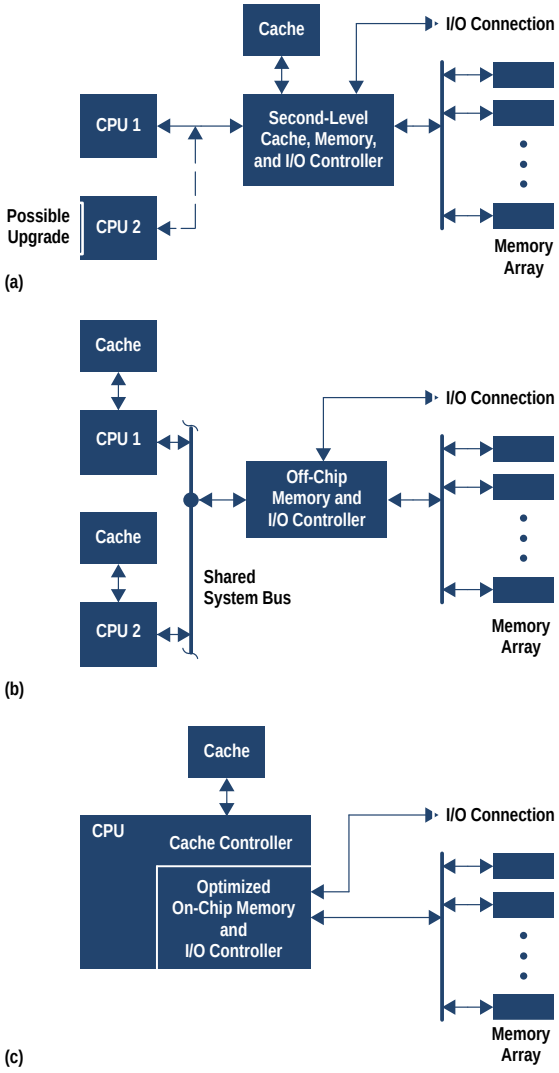


Fig. 3. (a),(b) Multiprocessor architectures in which the memory and I/O controller are separate from the CPUs. (c) A uniprocessor system in which the memory and I/O controller are integrated into the CPU chip.

could bring to a system through a focused uniprocessor design.

Integrating the memory and I/O controller with the CPU in a uniprocessor system (Fig. 3c) can have a dramatic effect on reducing cache miss penalties by decreasing the number of chip boundaries that the missing data must cross and by allowing the memory and I/O controller early access to important CPU internal signals. Miss processing on the memory interface can effectively begin in parallel with miss detection in the cache controller. An integrated memory controller can even use techniques such as speculative address issue to begin processing cache misses before the cache controller detects a cache miss.

The reductions in CPI (cycles per instruction) that we could achieve by integrating the memory controller allowed us the degrees of freedom that we needed to explore certain cache architectures in greater detail. Some of these architectures are described in the next section.

System complexity is reduced with an integrated memory and I/O controller. The 432-pin CPGA that we were considering for an integrated design had sufficient signal headroom to enable separate, dedicated memory and I/O connections. A two-chip approach, using the lower-cost MQUAD packages, would be forced to share pins between the memory and I/O connections to accommodate the low signal count of the MQUAD package, which would increase system complexity.

An integrated memory and I/O controller also simplifies the interface to the CPU. Since this interface connects two entities on the same die, signal count on the interface became much less important, which allowed us to simplify the interface design considerably.

On the down side, integrating a memory and I/O controller required enough flexibility in its design to satisfy the broad range of system customers that our chip would encounter. However, this requirement also exists for a nonintegrated solution. Historically, system partners have not redesigned memory controllers that the CPU team has provided as part of a CPU chipset. HP's advantage of providing both processors and systems has allowed us to work closely with system designers and enabled us to meet their needs in both integrated and nonintegrated chipsets.

In summary, integrating the memory and I/O controller onto the CPU core introduced a gain in performance, a reduction in complexity and schedule risk, and several possibilities for reduced cost in the cache subsystem. These were the compelling reasons to move the memory controller onto the CPU die and continue exploring cache alternatives and optimizing memory system performance.

Cache Organization. One of the distinguishing characteristics of HP PA-RISC designs over the past several implementations has been the absence of on-chip caches in favor of large, external caches. While competitors have dedicated large portions of their silicon die to on-chip RAMs, HP has continued to invest in aggressive circuit design techniques and higher pin count packages that allow their processors to use industry-standard SRAMs, while fetching instructions and data every cycle at processor frequencies of 100 MHz and above. This has allowed our system partners to take a single processor chip and design products meeting a wide range of price and performance points for markets ranging from the low-cost desktop machines to high-performance servers. For example, the PA 7100 chip has been used in systems with cache sizes ranging from 128K bytes to 2M bytes and processor frequencies ranging from 33 MHz to 100 MHz.

The main design goals for the PA 7100LC were low cost and high performance. Unfortunately, high-performance systems use large, fast, expensive caches. Obviously, trade-offs had to be made. As with previous implementations, the designers started with a clean slate and considered various cache options, including on-chip cache only, on-chip cache with an optional second-level cache, split instruction and data off-chip caches, and combined off-chip caches (see Fig. 4). Ultimately, the cache design was closely linked to the memory controller design because of the large effect of memory latency on cache miss penalties.

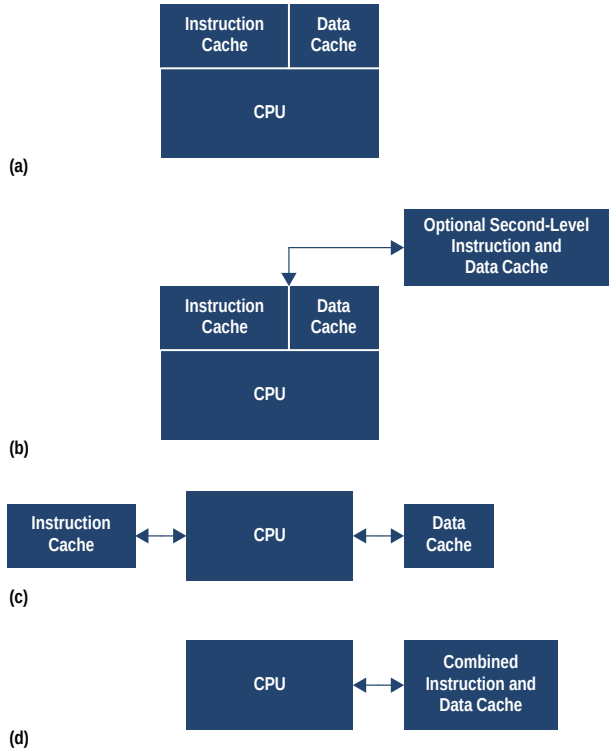


Fig. 4. Different cache organizations. (a) On-chip cache. (b) On-chip cache with an optional second-level cache. (c) Split instruction and data off-chip caches. (d) Combined off-chip caches.

On-chip caches have the obvious advantage that they can allow single-cycle loads and stores at higher chip frequencies than are possible with many off-chip cache designs. They also allow designers to build split and associative cache arrays which would be prohibitive for off-chip designs because of the large number of I/O pins required. Unfortunately, in current technologies on-chip caches tend to be fairly small (8K bytes to 32K bytes) and even with two-to-four-way associativity, they have higher miss rates than larger (64K bytes to 256K bytes) direct-mapped, off-chip caches. Also, on-chip caches require a substantial amount of chip area, which translates to higher costs, especially for chips using leading-edge technology with high defect densities. This extra chip area also represents lost opportunity cost for other features that could be included in that area. Examples include an on-chip memory and I/O controller, graphics controller, more integer execution units, multimedia special function units, higher-performance floating-point circuits, and so on.

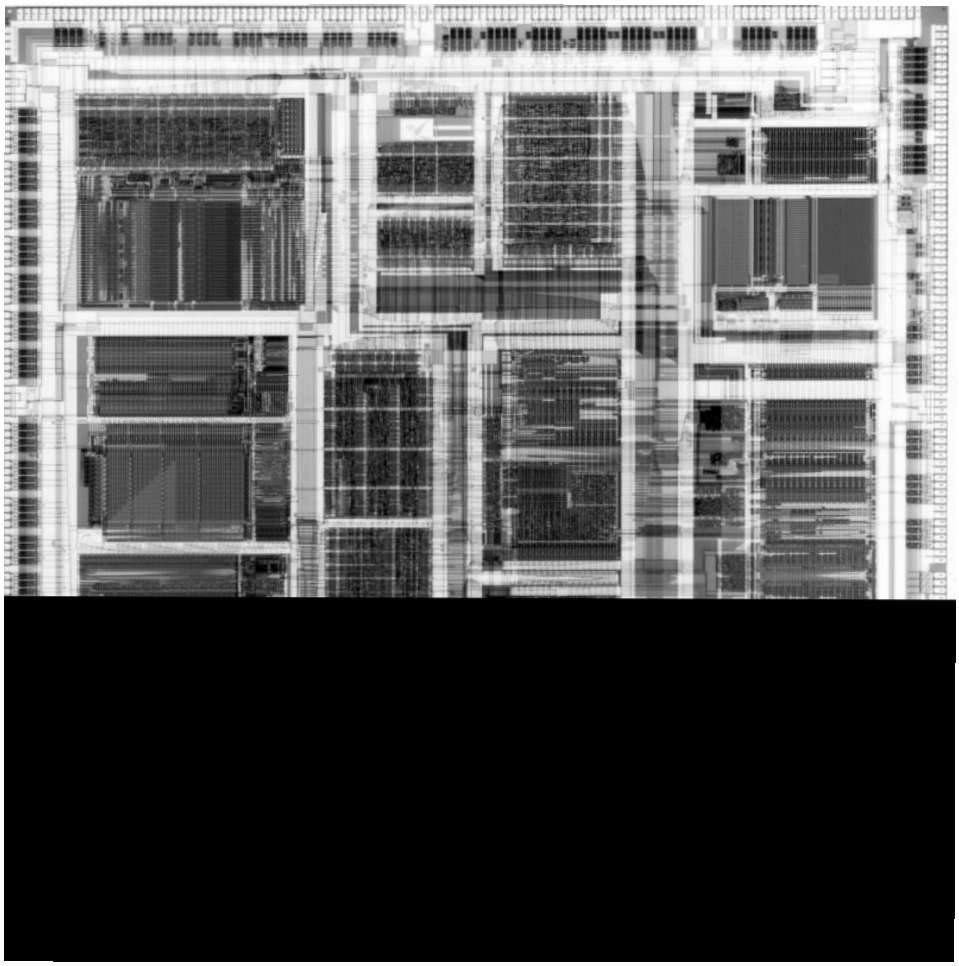
Another drawback of on-chip caches is their lack of scalability; providing multiple cache sizes requires fabricating multiple parts. To overcome this limitation designers can allow for optional off-chip caches. The off-chip caches can range in size and speed and can provide flexibility for system designers looking to meet different price/performance choices. Low-end systems need not include the off-chip cache and can be built for a lower cost. High-end systems can get a performance boost by paying the extra cost to add a secondary off-chip cache. For most systems, the cost for this flexibility is added pin count to allow for communication with the off-chip caches. Other systems might be able to multiplex the cache lines onto some already existing buses such as the memory bus.

For the PA 7100LC, we determined that a primary on-chip cache would cost too much in terms of more expensive technologies, increased die size, and the lost opportunity of putting more functionality on the chip. Without a primary on-chip cache, we were able to design a processor with two integer units, a full floating-point unit including a divide and square root unit, and a memory and I/O controller. We achieved this functionality using only 905,000 FETs in 0.8 micrometer (CMOS26) technology on a die measuring 1.4 cm by 1.4 cm (see Fig. 5). CMOS26 is a mature HP process that has been used for several processor generations. As a result, it has a low defect density and thus, a low cost. A processor with an on-chip cache would have required a more advanced technology having higher wafer costs and defect densities. Of course, without an on-chip cache, we were challenged to design a low-cost off-chip cache that allowed accesses at the processor frequency.

HP's previous implementations of PA-RISC were built with independent instruction and data caches made up of industry-standard SRAMs (see Fig. 4c). It would have been easy to leverage the independent direct-mapped instruction and data cache design from the PA 7100, but we were determined to find a less expensive solution. Independent cache banks require a high pin count on the processor chip because each bank requires 64 data pins and about 24 pins for tag, flags, and parity. Thus, combining instructions and data into a single set of cache RAMs (Fig. 3d) saves about 88 pins on the processor chip. These extra pins directly affect packaging costs. Also, providing split caches requires using more SRAM parts in a given technology. Systems based on the PA 7100LC with a combined cache require only 12 SRAM parts using ×8* technology. By leveraging the aggressive I/O design from previous implementations, the PA 7100LC can access 12-ns SRAM parts every cycle when operating at frequencies up to 66 MHz. Since 8K × 8, 12-ns SRAMs are commodities in today's market, the cost of a 64K-byte cache subsystem for a 60-MHz PA 7100LC is comparable to the price we would have paid for a much smaller on-chip cache.

Combined instruction and data caches have one large drawback. Since the PA 7100LC processor can consume instructions as fast as the cache can deliver them, there is little or no cache bandwidth left to satisfy load and store operations. To solve this problem, we needed to implement some type of instruction buffer on the processor chip. A large instruction buffer would have all the drawbacks of the on-chip cache design discussed above, so we were determined to find a way to achieve the desired performance with a small buffer. We knew we would need a mechanism to prefetch instructions from the off-chip combined cache into the dedicated on-chip buffer during idle cache cycles. Thus, we started with a standard direct-mapped 2K-byte buffer and simulated various prefetch and miss algorithms. As expected, we found that performance was extremely sensitive to the buffer miss penalty, which ranges from zero to two states

* RAM sizes are quoted in depth by width (i.e., 64K ×8 is 65,536 deep by 8 bits wide).



Design Methodologies for the PA 7100LC Microprocessor

Product features provided in the PA 7100LC are strongly connected to the methodologies developed to synthesize, place and route, simulate, verify, and test the processor chip.

by Mick Bass, Terry W. Blanchard, D. Douglas Josephson, Duncan Weir, and Daniel L. Halperin

Engineers who wish to create a leading-edge product with competitive performance, features, cost, and time to market are often challenged to create design methodologies that will enable them to succeed in their task. Decisions about the features of a product usually have an inseparable impact on the methodologies used to create, verify, debug, and test the product.

During the development of the PA 7100LC microprocessor,[1,2] engineers crafted several methodologies that supported the design decisions that were made throughout the project and provided the framework for implementing the design decisions.

This article explores several of these methodologies. For each methodology, we discuss the design decisions that impacted the methodology, the alternatives that we considered, and the course that we chose. We discuss the results produced by each methodology, as well as problems that we encountered and overcame during each methodology's development and use.

Some of the design decisions that motivated us to develop new design methodologies for the PA 7100LC are discussed in the article on page 12. The areas in which we developed these methodologies include control synthesis, place and route, production test, processor diagnosability, presilicon verification, and postsilicon verification.

The resultant methodologies were crucial to our ability to meet the design goals that we had set for the PA 7100LC. Taken together, they enabled good decisions leading to a successful product implementation.

Synthesis and Routing Methodology

The control circuits in any microprocessor typically represent a major portion of the complexity of the chip. The control circuits of the chip contain most of the chip's intelligence. It is these circuits that direct the rest of the components on the chip. The operation of the control circuits is similar to the way operators of complex machines on a factory floor control the way that those machines behave.

Blocks of control circuitry perform similar jobs, and the nature of these jobs determines the nature of the control blocks themselves. Control blocks typically implement logic equations, the outputs of which control some other function present on the chip. The logic equations implemented by control blocks tend to be irregular and loosely structured. A necessary characteristic of any control block is for its outputs to become valid in sufficient time to control its downstream circuits properly. Like other portions of the chip, control blocks can have timing paths that limit the overall chip operating frequency if the blocks are not carefully designed and implemented.

Another characteristic of blocks that implement control logic is that they change frequently throughout the design process. Experience has shown that a vast majority of bugs are found in the control blocks, probably because so much of the chip complexity resides there. We have found that it is very likely that the last bugs fixed before a chip design is sent to manufacturing will be in these blocks.

When we were defining the methodology for implementing the control circuitry for the PA 7100LC, we considered these general characteristics, as well as specific new requirements that stemmed from our design goals for the project. The PA 7100LC had new requirements, compared to earlier CPUs, in the areas of low power dissipation and support of I_{DDQ} testing. We knew that the PA 7100LC control would be even more complex than past CPUs because of its high level of integration and its superscalar design. To make it easy to accommodate this new functionality, we wanted to be able to make the control blocks as small and as flexibly shaped as possible. Finally, since we were leveraging the design of the PA 7100LC processor from the PA 7100 processor,[3,4] we wanted to leverage control equations or control circuitry from the past design for many of the blocks.

The control of the PA 7100, from which we were leveraging, is primarily implemented as a programmable logic array (PLA). Programmable logic arrays have very regular physical and timing characteristics. The PLA architecture used in the PA 7100 involves dynamically precharged and pseudo-NMOS circuits. The outputs of this PLA become true at least one CPU state after its inputs became valid. The PLA latches all inputs with respect to a specific fixed clock edge.

PLA Methodology. The methodology used to design PLAs for the PA 7100 was well developed as were the tools that were necessary to support it. PLAs were designed in a high-level language with a syntax reminiscent of the Pascal programming language. In-house tools were available to translate the high-level source language to optimized Boolean sum-of-products equations. Other in-house tools were available to

use these sum-of-products equations to generate the PLA artwork (including programming the array).

When the destination circuits could not tolerate the one-state delay required by the PLA core, we created schematics for handcrafted standard-cell blocks that could calculate their outputs in the required time. We then used an in-house channel router to create artwork for the standard-cell blocks.

The PA 7100 PLA methodology had several advantages. The PLA design and implementation tools were simple and well-understood. They provided a turnkey artwork generation solution from the high-level control equations, which made it easy to accommodate late changes. Most important, we already had a high investment in this methodology. We understood it very well, had all the required tools in place, and knew we wouldn't find any surprises.

However, when considered in light of the requirements of the PA 7100LC, the PLA methodology had several disadvantages. Although the physical structure of a PLA is fixed and very regular, its fixed shape would lead to difficulty in floor planning for a chip as highly integrated as the PA 7100LC. We also knew that PLA implementations of control logic do not yield optimal circuits with respect to absolute size. PLA circuits involve both precharged logic and pseudo-NMOS logic, leading to high power dissipation relative to fully static circuits. PLA circuits are also incompatible with our I_{DDQ} test methodology, which is described later in this article. Although PLAs can usually guarantee a one-state delay from input to output, their timing is inflexible. The addition of hand-designed standard-cell blocks to address this problem is not only labor-intensive, but also adds complexity to the overall solution and increases the probability of introducing bugs in these areas. Also, some types of control logic cannot be represented compactly in the sum-of-products form required by the PLA methodology. This logic must then either be moved into a standard-cell block or redesigned.

New Methodology. Since the disadvantages of the PLA methodology would compromise our ability to achieve our design goals, we began to investigate alternatives. We had some positive experience with using Synopsys, a commercial synthesis tool, on the floating-point control block of the PA 7100. We began to investigate the potential impact of combining automated synthesis using Synopsys with an over-the-cell router.† Our investigation of combining the synthesize and route methodology pointed out the following advantages and disadvantages:

- The absolute size of the blocks produced would be smaller than the blocks produced using either PLAs or channel-routed blocks. Additionally, the floor plan would be more flexible than that produced by a PLA, allowing us to partition the controller so that we could create control blocks that fit into available area close to the circuits they must control.
- We would have to pay more attention to timing because we would no longer have the regular timing structure of the PLA to guarantee that state budgets would be satisfied.
- The circuits produced would dissipate less power than corresponding PLA implementations because the synthesize and route methodology uses fully static circuitry. The circuits would also be I_{DDQ} compatible.
- We would have to design a new library of standard cells that would be compatible with the over-the-cell router. We would also need to design a new set of drivers that would drive output signals from the standard-cell core to the rest of the chip and that would be compatible with our production test design rules. These tasks were very well-defined and we understood the effort that would be required to complete them.
- Of greater concern was the realization that the synthesis path from the input equations to completed artwork would be more complex than the corresponding path in the PLA methodology and would be almost completely new.

With the PLA methodology, we knew that there would be no surprises. Incorporating this new technology would remove much of that certainty. However, the benefits clearly outweighed the costs. We felt that we couldn't afford to compromise our power, area, timing, and test goals by continuing with the PLA methodology.

We overcame several issues while making the new methodology work for us. We leveraged the source code of many of the control blocks from the PA 7100, all of which were specified in the PLA source language. We were able to leverage existing PLA sources directly by using the PLA tools to generate sum-of-product equations in a form that the Synopsys synthesis tool could understand. Synopsys was then free to massage the equations into a more optimal form. Source code development of these leveraged control blocks continued using the PLA source language, even though we were using the new methodology for synthesis and route. We developed control blocks that were new for the PA 7100LC using the Verilog behavioral description language, which has a more direct input path to Synopsys.

We chose the Cell3 router from Cadence Systems Inc. to perform the place and route portion of our new methodology. The main issue remaining was how to integrate this new tool with our other tools. To minimize the number of costly licenses we needed to purchase and to maximize the block designers' productivity, we decided to use our existing artwork editor as a front end to the router's floor planning capability. This approach allowed designers to preplace critical cells, power nets, and clock nets easily. We developed new tools that would translate this floor plan into a form that the Cell3 router could understand. While these techniques maximized designer productivity and minimized license cost, we found that it was sometimes difficult to isolate bugs in the methodology to either our front-end tools or to the Cell3 router itself.

We also discovered that the timing capabilities of the version of Synopsys that we used were less robust than we had believed at the beginning of the project. This discovery had only a minimal impact on blocks that were leveraged from PLAs because of the regularity in the timing of those blocks. However, to ensure robust timing on the remaining blocks, we needed to develop new tools. The need for these unanticipated workaround tools had a negative impact on our schedule.

As with PLAs, we also found that certain types of circuits do not map well to the synthesize, place, and route methodology. On a large block where we made much use of the timing flexibility offered by static standard cells, we found that our

† Over-the-cell routers place and route cells so that there is less need to provide routing channels between the cells.

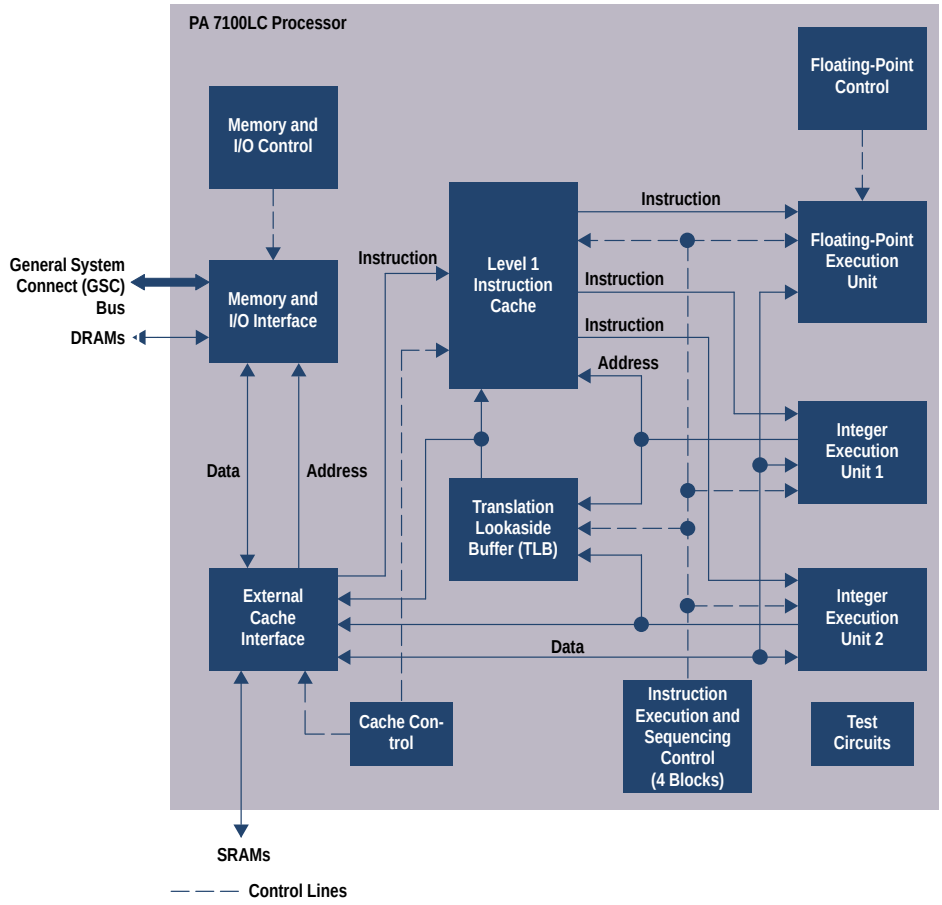


Fig. 1. A simplified block diagram of the PA 7100LC showing the relationship between the control blocks and the other major blocks in the processor. The instruction execution and pipeline sequencing control block consists of four separate blocks that are physically distinct but highly interconnected. Not all of the control connections on the PA 7100LC are shown in this figure.

synthesis tools were sometimes unable to produce circuits that met the timing and area constraints of the block. Whenever this occurred, we had to redesign the control source so that the synthesized circuits could meet their physical requirements, or help the tools by hand-designing portions of the circuit.

We found that on some of the standard-cell blocks leveraged from the PA 7100, the synthesis tools had difficulty creating circuits that performed as well as their PA 7100 counterparts. This difficulty was caused in part by differences in the standard-cell libraries for the two chips. The PA 7100LC library had no pseudo-NMOS circuits, which were used quite effectively to meet timing on the PA 7100 (at the expense of higher power dissipation). The rest of the difference lies in the fact that, for all its sophistication, automated synthesis is still no match for carefully hand-designed blocks. Fortunately, our design tools allowed us to hand-design portions of the block while synthesizing the rest of the block. Although time-consuming, we chose this approach in cases where the tool path was unable to provide a satisfactory solution.

The overall results of the methodology we chose were good. We were able to partition the PA 7100LC's control functionality into seven primary control blocks. Four of the blocks control the sequencing and execution of instructions by the pipeline. The remaining three control blocks control the memory and I/O subsystem, the cache subsystem, and the floating-point coprocessor (see Fig. 1). Together, these seven blocks represent only 13% of the total die area, and implement nearly all of the control algorithms and protocols used by the PA 7100LC.

Even though the PA 7100LC adds integer superscalar execution and a memory and I/O controller compared to the PA 7100, the area of the control core produced by the new methodology is about half the area of the PLA core of the PA 7100. The area occupied by the driver stacks in the control blocks on the two chips is about the same.

The new methodology implemented all of the control blocks correctly and introduced no functional bugs. The timing methodology that we had in place by the end of the project was very effective at identifying problem timing paths before they made it onto silicon. When we received chips from manufacturing, we found no problem timing paths in any of the control blocks that were created using the new methodology.

Verification Methodology

One of the most prominent design goals for the PA 7100LC was to meet the schedule required to enable a very steep production ramp. This goal, coupled with Hewlett-Packard's commitment to quality, meant that we needed to have in place a solid plan to verify the correctness of the chip at all stages of its design.

Our design goals and the knowledge that the PA 71000LC was to be the most highly integrated CPU that HP had ever created led us to focus early on the methodology that we

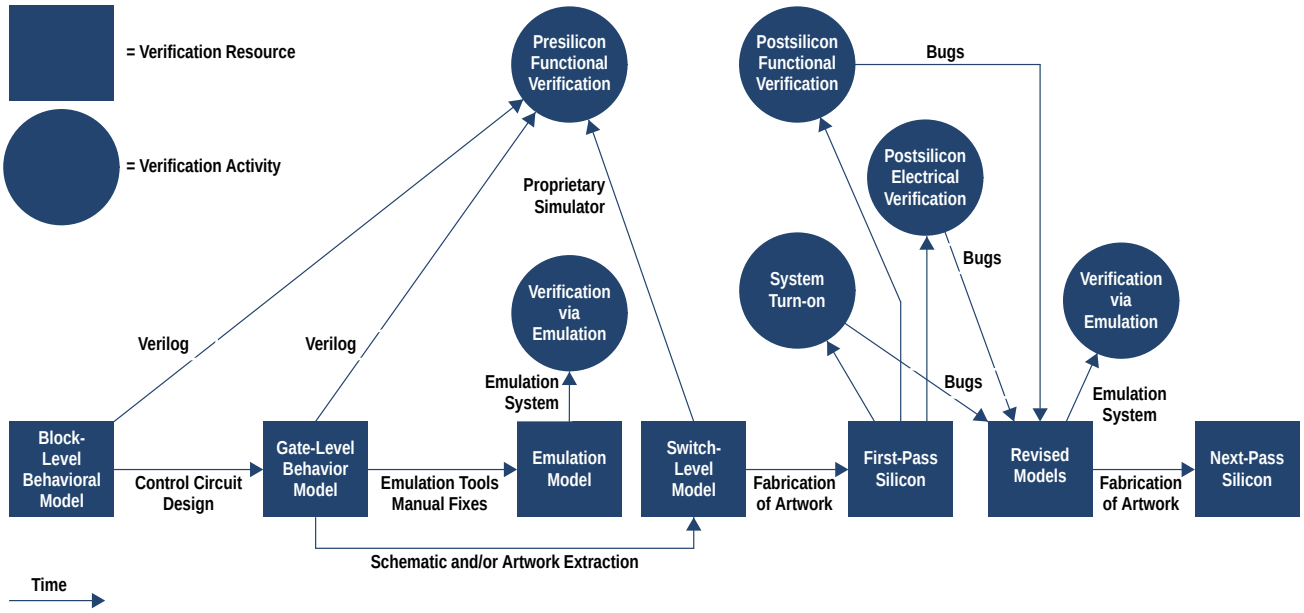


Fig. 2. Overview of the functional verification process.

would use to verify the chip. As shown in Fig. 2, our verification methodology included several distinct forms of verification, some of which occur before silicon is manufactured (presilicon verification) and some of which occur after first silicon appears (postsilicon verification).

Presilicon verification activities included:

- Creating software behavioral models through which we could verify the correctness of either the entire design or portions of it
- Creating switch-level models of the implementation to ensure that the implementation matched the design
- Writing test cases that provided thorough functional coverage for each of these models
- Using in-circuit emulation to increase vector throughput and to provide an orthogonal check of the chip's correctness.

Postsilicon verification activities included:

- Augmenting functional coverage by running hand-generated test cases, randomized test cases, and application software
- Testing actual silicon against its electrical specification using a rigorous electrical testing procedure.

We designed each portion of our verification methodology to ensure that we could meet our schedule and quality goals. The following sections describe in more detail the types of verification we used.

A New Strategy

At the time work was starting on the development of the PA 7100LC chip, HP was moving toward a new product development philosophy, which had as its basis the fact that HP could no longer afford to do everything for itself. The time had come to specialize in core competencies and look to outside vendors to cover the needs common in the industry. Unless HP provided a clear competitive advantage over industry-standard tools and methods, design teams were encouraged to adopt these standards, paying others to develop and maintain leading-edge tools and processes.

During the PA 7100LC investigation phase, engineers investigated industry-standard tools in the areas of behavioral simulation, static timing analysis, fault grading, timing verification, switch-level simulation, and other areas of chip verification. The first and foremost goal of these investigations was to determine which tools provided the fastest and most efficient contribution toward design and verification, ultimately leading to earlier products. The following section will provide an analysis of our behavioral simulator selection, which is just one example of the many tool decisions we made for the PA 7100LC.

Behavioral Simulation. Before the PA 7100LC development effort, we had been using a proprietary simulator which was written and maintained by an internal tools group. With the standardization of simulation languages in the industry, we questioned the value of high internal development and maintenance costs for this tool. We investigated the language and simulator options available in the industry and eventually reached a final list of choices:

- The proprietary HP solution
- Verilog
- VHDL (IEEE standard 1076).

Other HP design labs, responsible for graphics and IC hardware design, had migrated to Verilog from the HP simulator and had found significant improvements in simulation throughput on their ASIC designs. The throughput disadvantage of the HP simulator was somewhat balanced by the fact that it carried no licensing fees, was fully robust, and had been proven capable of simulating a large, custom IC design such as a CPU.

Verilog had become a de facto standard in the U.S. for high-level and gate-level simulation in 1992 and had been used extensively in HP's graphics hardware and IC design labs. Their experience indicated that Verilog was very robust and that it allowed personalized extensions through linking with C code. The IC design lab demonstrated simulation speeds

with Verilog that were about seven times faster than the internal HP simulator. Since Verilog was becoming more common within HP, it would ease our task of sharing and combining simulation models with design partners. For example, the floating-point circuits that we would be leveraging from the PA 7100 for the PA 7100LC were modeled in Verilog. The graphics chip and the LASI chip used in the Model 712 workstation were being developed using Verilog, and many of the commercial ICs used in the system had Verilog models available for system simulation. By choosing Verilog, we would create a homogeneous environment. We also felt that Verilog's C-like syntax would allow engineers to learn the language quickly. Finally, the Verilog language would provide a bridge to other useful industry-standard tools for static timing, fault grading, and synthesis.

At the time we were investigating simulators we found only one supplier who could provide a mature Verilog simulator in our required time frame. This particular simulator had some disadvantages compared to our internal simulator, which included higher main memory requirements and the need to recompile the simulation model at each invocation of the simulator. For large models, this compile phase could last a full minute. The internal simulator, by contrast, compiled the model once into an executable program which contained the simulation engine, and incurred no run-time startup penalty. Also, because Verilog was licensed we would have to purchase sufficient licenses to cover our simulation needs, which would present a large initial expense.

A third major simulation language we investigated was VHDL (IEEE Standard 1076). While Verilog was becoming a de facto standard in the United States, VHDL was sweeping Europe. VHDL shared many advantages and disadvantages with Verilog. Simulation models of commercial system chips were often available in both languages. VHDL provided hooks to support industry-standard tools for timing, fault grading, synthesis, and hardware acceleration. VHDL was also licensed and would be expensive. The primary differentiator between VHDL and Verilog was in ease of use and ease of learning. Other HP design labs indicated that VHDL was more difficult to learn and use than Verilog. Also, there was no local expertise in VHDL, while proficiency in Verilog had been growing, and significant inroads had already been made at integrating Verilog into the remainder of our tool set.

With this information in mind, the PA 7100LC technical team decided to use Verilog as the modeling language for the PA 7100LC processor. The compelling motivations for this choice were:

- The demonstrated success of other HP labs in using the Verilog simulator in ASIC designs
- The availability of local expertise and support for the simulator and modeling language
- The ability to standardize on a single simulator and modeling language for the development of all custom VLSI used in the HP 9000 Model 712
- The ability to interface easily to other industry-standard tools.

Given this decision, we joined an effort with other design labs to enhance the Verilog simulator to include an improved user interface and more tool interfaces to be used throughout our verification effort.

Turn-on Process. We migrated to the Verilog modeling language and simulator in two steps. First, we validated that Verilog could simulate an existing PA-RISC design of comparable complexity to the PA 7100LC by converting the PA 7100 simulation model (from which the PA 7100LC design is leveraged) into Verilog. Second, we used the knowledge that we gained during this conversion process to complete the development of the PA 7100LC.

Converting the PA 7100 simulation model into Verilog was a good decision for several reasons. We wanted to start with a known functional model from which we could leverage. We also needed to confirm that Verilog was robust and accurate enough to model a design as large and complex as a CPU. The PA 7100 offered a hierarchical, semicustom design model that consisted of high-level behavioral blocks (e.g., the translation lookaside buffer) and FET descriptions (e.g., in custom leaf cells). This varied design would provide a good test of the simulator's ability and would help us to learn about Verilog's unique requirements.

To aid the conversion process, we created a tool that converted the HP proprietary modeling language to Verilog syntax. We fixed code by hand wherever the two languages did not have similar constructs or where they evaluated similar constructs differently. The converted model passed its first test case within two months.

Once the PA 7100 model was up and running in Verilog, we measured its simulation throughput. Instead of the expected 7× speedup, we discovered a full 4× slowdown compared to the HP simulator. We also found that the model consumed more memory than we had anticipated. Through careful analysis and support from our supplier, we learned that much of our model syntax was very inefficient. In addition to inefficiencies created by the translation tools, many syntax structures that were optimum in HP's simulator were nonoptimal in Verilog. Profiling and correcting these inefficiencies greatly improved performance and resource requirements.

Results. The result of the decision to use Verilog to model the PA7100LC was positive, with a few disappointments. The main disappointment was that the Verilog model of the PA 7100LC achieved only parity in throughput and required five times more memory than the HP simulator.

However, Verilog brought strengths in other areas. Verilog allowed us to make incremental changes to the model quickly and easily. Verilog enabled us to capitalize on industry-standard tools in the areas of synthesis, timing, fault grading, and in-circuit emulation. We were able to use a single modeling language across all of the custom components in the HP 9000 Model 712 workstation and to obtain compatible models for many of the external components.

We soon learned to use the new strengths provided by Verilog and became efficient in using the language and the new simulator. Verilog successfully modeled all constructs required in the PA 7100LC design, and a high level of quality was the end result of using this tool.

Presilicon Functional Verification

Because the cost and lead time of manufacturing CPU die are so great and because our system partners depend on fully functional first silicon to meet their schedule goals, it is important that our presilicon verification methodology give us high confidence in the functional quality of the first silicon. This task proved to be a challenge for the PA 7100LC chip because it was designed by many engineers, and its feature set is extensive and complex. These factors introduced the opportunity for design and implementation bugs.

The PA 7100LC is the first HP processor chip to integrate the memory and I/O controller on the same die as the CPU. In the past, these designs lived on separate die and were owned by separate project teams. The verification efforts for the two designs were mostly independent. A careful specification of the interface between the two designs allowed this approach to succeed.

We realized that even though the PA 7100LC would integrate the memory and I/O controller onto the CPU die, it would be more effective to verify the memory and I/O controller separately from the CPU core for the majority of the tests. This would allow test cases for both the CPU and the memory and I/O controller to be more focused, smaller, and faster to simulate than they would be in a combined model. We created a well-defined interface between the CPU and memory and I/O controller to enable this approach.

Each of these presilicon verification efforts was structured as shown in Fig. 2. First we created a behavioral model for the portion of the design whose function was to be verified. A behavioral model represents the design at some level of abstraction, and typically moves from very high-level to much more specific as the project progresses. As mentioned above, we chose Verilog as the modeling language for our design.

The behavioral model was the heart of the simulation environment that would enable us to verify the CPU and the memory and I/O controller. Our job was to find deficiencies in this model. However, to do this we needed a way to stimulate the model, observe its results, and ensure that its behavior was correct. To meet these needs, we created additional software objects to complete the simulation environment.

At each of the external interfaces of the behavioral model, we created custom code that was capable of modeling the behavior of the device on the other side of the interface and of stimulating and responding to the interface as appropriate for that device. For example, these stimulus-generating software objects were used in our simulation environment in the same way that dynamic RAM, external cache, and I/O devices are used in a physical system. We authored the code that models these objects in a high-level language (typically C).

Another type of custom software that augments the simulation environment consists of checkers. A checker monitors the behavioral model and checks aspects of model behavior for correctness. We used a number of different checkers during the PA 7100LC verification effort. Some checkers were very focused (e.g., a protocol checker on the I/O bus), and others were more global (e.g., the PA-RISC architectural simulator).

Creating "watchdog" pieces of code to detect and signal errors automatically in the simulation environment helped us to maintain our schedule. Previous CPUs had an independent model of the design that matched the behavioral model state-by-state for all external pads and architected internal state.* Creating the independent model was time-consuming and not easily broken into small pieces that could be worked on in parallel. We couldn't run test cases on the behavioral model without a fully functional independent model. Replacing this independent model with a collection of checkers allowed us to create multiple checkers at the same time. We were able to turn on the checkers independently as the functionality that they checked became available in the behavioral model. Also, the checkers didn't need to be fully functional for us to run useful test cases.

The final aspect of the simulation environment is the test case. A test case provides initialization to the model and the stimulus generating software objects and then orchestrates the stimulus generators to provide external stimulus while the model is simulating. The checkers constantly watch model behavior and identify rules that the model violates. The test cases are not self-checking. They simply stimulate the model and rely on the checkers to ensure that the model responds correctly.

We wanted the test cases to create the complex interactions in the CPU core and in the memory and I/O controller that are necessary to find subtle bugs. The model, stimulus generators, and checkers provide an environment that makes it easy to generate short, powerful test cases. To improve test case coverage, we gave the responsibility for test case creation to both the CPU and the memory and I/O controller designers, who had a detailed knowledge of the internal operation of the chip, as well as to independent verification engineers, who knew only the external functional specification of the chip. We used design reviews to ensure that our suite of test cases adequately covered all functionality present in the design.

Testing on the behavioral model is the first line of defense against flaws in a design. To ensure that our implementation matched the design, we ran our full suite of test cases on a gate-level behavioral model. We created this model from the complete chip schematics. We also tested a switch-level model that we created by extracting the FET netlist from the completed chip artwork. Since this was the same artwork that manufacturing would use to fabricate the chip, this regression served as a final test of the functional correctness of both design and implementation.

*In this usage architected state refers to a particular pattern of ones and zeros on internal chip nodes.

To ensure that there were no coverage holes in the interface between the CPU and the memory and I/O controller, we created a model that merged these two designs into a single behavioral model of the entire chip. We tested this model to gain certainty that both parts would work properly together.

Finally, we combined behavioral models of the PA 7100LC with behavioral models of other chips in the system and performed system-level verification to ensure that each of the chips interpreted the interchip interfaces consistently and to ensure that all the chips in the system functioned as expected.

Using this extensive verification methodology, the first silicon we delivered allowed us to boot the HP-UX* operating system and enabled our system partners to progress towards meeting their system schedules.

Postsilicon Functional Verification

Presilicon verification, while providing an excellent first pass at ferreting out design or implementation flaws, is not capable of identifying all bugs in a complex custom CPU such as the PA 7100LC. Two factors make this true. First, the simulation speeds of even high-level behavioral models (typically less than 10 Hz) are not sufficient to exercise all the interesting state transitions within the CPU in the time available. Second, experience has shown that in a chip of this type there are sometimes subtle differences between the presilicon model and actual chip behavior.

To ensure a quality CPU design, we performed extensive postsilicon testing on the PA 7100LC in systems running at actual processor speeds (50 to 100 MHz). The difference of about seven orders of magnitude in vector throughput between running test cases on presilicon models and code running on actual silicon underscores the potential for thorough testing offered by postsilicon verification.

One of the goals of presilicon testing is to ensure that the simulation model matches the behavior specified by the design. We carried this goal into postsilicon testing and ran a suite of tests on actual chips in a computer system. The tests behaved the same when they were run in the computer system as they did on the PA 7100LC presilicon models.

We knew that postsilicon testing would be the last opportunity to find functional problems with our processor before we shipped systems to customers. Since the cost of finding a serious functional problem once systems are shipped is extremely high, we wanted to exercise the processor thoroughly with as many different tests as possible. The variety of features that we had added to the PA 7100LC made this process more difficult. Each of these features had to be tested, usually in combination with other features.

The tests that we used during the PA 7100LC postsilicon verification effort included:

- A collection of handwritten tests, run in an environment that made them more stressful for the processor
- Random code generators that produced software that deliberately stressed complex areas of the processor
- A collection of application software including operating systems, benchmarks, and other applications.

Handwritten Tests. Hewlett-Packard has created a library of programs whose purpose is to ensure that a processor conforms to the PA-RISC architecture. In addition to this library, we created other programs to test specific processor features. We also created a small operating system that allowed many of these programs to run simultaneously and repetitively in a manner that was stressful to the processor. This operating system would interrupt the programs at different intervals and also change portions of the processor state (e.g., cache and TLB) before restarting a program. Finally, the operating system kept an extensive log of program activity to help us track down bugs that it found.

In addition to the programs that we ran under the special operating conditions, we created another set of handwritten tests specifically to test the memory and I/O controller portion of the processor. These tests used an I/O exerciser card to ensure that the memory and I/O controller would behave properly in the presence of any conceivable I/O transaction. We also used these tests to exercise the DRAM interface of the memory and I/O controller.

Focused Random Testing. To supplement the handwritten tests we developed two random code generators. Experience gained during past processor designs had taught us that a certain class of bugs appear only when a number of complex interactions occur within the CPU. It wasn't feasible to create handwritten tests to cover all of these interactions because the time requirements to do so would be prohibitive. Additionally some of the tests would need to cross so many interactions that it would be difficult to guarantee adequate coverage with handwritten cases. Using a random code approach, we used code generators to create the test cases that found bugs in this class.

Another strength of the random code approach was that we were able to take full advantage of the speed of postsilicon testing. We could run all handwritten tests in a short time on an actual processor. Random code generators made it possible to generate millions of different tests to keep the processor fully exercised, at speed, for long periods of time.

One could create many conceivable random code generators, which could differ in many ways including the type of code produced, fault latency, ease of debugging, repeatability, and initialization. Design differences in random code generators cause coverage differences (one generator may be able to find a bug that another missed). Random code generators mainly differ in the sequence of instructions and in what constitutes initial and final processor state. In general it is best to run code from as many different sources as possible to ensure the best coverage.

Of the two random code generators that we developed, one stressed the floating-point unit and another stressed the integer unit. Each of these generators produced tests consisting of:

- An initial processor state
- A sequence of PA-RISC instructions
- An expected final processor state.

The focused random approach worked extremely well during the PA 7100LC verification effort. Using it, we were able to

complete thousands of machine-hours of testing and identify a majority of postsilicon bugs.

Our decision to emphasize random code testing paid off. Because of the proven effectiveness of the random approach, we will probably continue in this direction and make evolutionary changes to make the approach even more effective.

Application Software. In addition to handwritten and random tests, we ran a variety of "real-world" software applications to further ensure that we had found and fixed all bugs. These applications were intended to help diagnose failures suspected to be caused by the hardware. We booted operating systems (like HP-UX) shortly after chips were available. We also conducted long-term operating system reliability tests when more stable hardware and software became available. We filled out our array of application software tests with benchmark suites and other applications.

Acceptance Criteria. A challenging question that engineers and managers face during any postsilicon verification effort is "When are we done?" Having clear criteria for the quality required to ship the chip to customers is paramount. For the PA 7100LC, we used the following acceptance criteria:

- All failures are diagnosed to root cause.
- No chip failures exist.
- All handwritten code works.
- Random code generators have run for a long time without finding any failures.
- Application software has run without any indication of hardware bugs.

In-Circuit Emulation

In addition to constantly tuning existing design and verification methodologies in areas where high-impact productivity gains are essential to stay on the leading edge of the industry, we also look for new breakthrough technologies and areas for paradigm shifts. We considered in-circuit emulation as such an area for the PA 7100LC.

In-circuit emulation means that a chip is modeled at the gate level in field programmable gate arrays (FPGAs) and connected directly to a chip socket in a real system running at a reduced frequency. This allows the modeled chip to run real system-level software.

Continual increases in chip complexity must be countered with more effective verification to ensure high-quality first-silicon chips. The goal is to have a perfect chip, but the requirement is to prevent masking bugs. A masking bug is a serious bug that causes a class of chip functionality to fail. The verification team is unable to "see behind" the bug to test for other failures in that area of functionality. The chip must be redesigned to fix the masking bug and must pass through fabrication before this functionality can be tested. Emulation was viewed as a way to prevent these serious masking bugs.

Besides ensuring high-quality first silicon, it is also desirable to have enough presilicon simulation throughput to verify any proposed postsilicon bug fix. Since turning a chip is costly and time-consuming, incorrect bug fixes that cause additional bugs must be eliminated.

During the early phases of the PA 7100LC chip design effort, in-circuit emulation technology came of age and was available through external vendors. We investigated this new technology in depth. For us, in-circuit emulation was viewed as a paradigm shift in verification and very attractive because it would:

- Provide near "real hardware" throughput with a presilicon model
- Allow thorough regression of any mask or full chip turns necessitated by bugs or timing paths found during postsilicon verification
- Allow the firmware and software teams to test their code before real hardware was available
- Add another important debugging capability to our suite of debug tools that allow us to isolate postsilicon bugs
- Allow us to recreate real hardware failures on a presilicon model and allow visibility to all internal nodes of the chip.

We also saw some areas of concern in pursuing in-circuit emulation. We perceived in-circuit emulation as challenging and risky because it was a new technology within a very young industry. We lacked expertise in using emulation tools, and it would be expensive to gain the necessary expertise to make in-circuit emulation part of our chip design methodology. In addition to this, the emulation tools and hardware were very expensive.

Our concern with technology risk was eased by several factors. We were promised very strong (on-site) support from the emulation company that we chose. They assured us that tools capable of handling large designs would be available early in our design cycle. We had independent corroboration from other HP entities, who had seen great success with emulation in ASIC design efforts.

After weighing the potential advantages, risks, and our long-term needs we determined to pursue in-circuit emulation. We didn't believe that emulation was absolutely critical to our success on the PA 7100LC, but we felt that dramatic improvement in simulation throughput would be required to verify the increasing complexity of our next-generation processor design. This effort was simply the first step in a long-term strategic direction.

Emulation Methodology

The real goal of our emulation effort was to plug the emulation model into the physical system and run at frequencies near 1 MHz. The team modified an HP 9000 Series 700 workstation to provide the required boot ROM, disk, and I/O subsystem. A special processor board was designed that allowed the emulation system to plug into the CPU socket. This board also provided external cache (SRAM) and main memory (DRAM). One challenge was to keep the DRAM refreshed since the processor wasn't running fast enough to keep memory refreshed and make forward progress on the code stream at the same time. We implemented a solution that coalesced the processor memory transactions between refresh cycles provided at a constant frequency by a module external to the CPU. This made refresh transparent to the PA 7100LC emulation model. Fig. 3 shows our emulation setup.

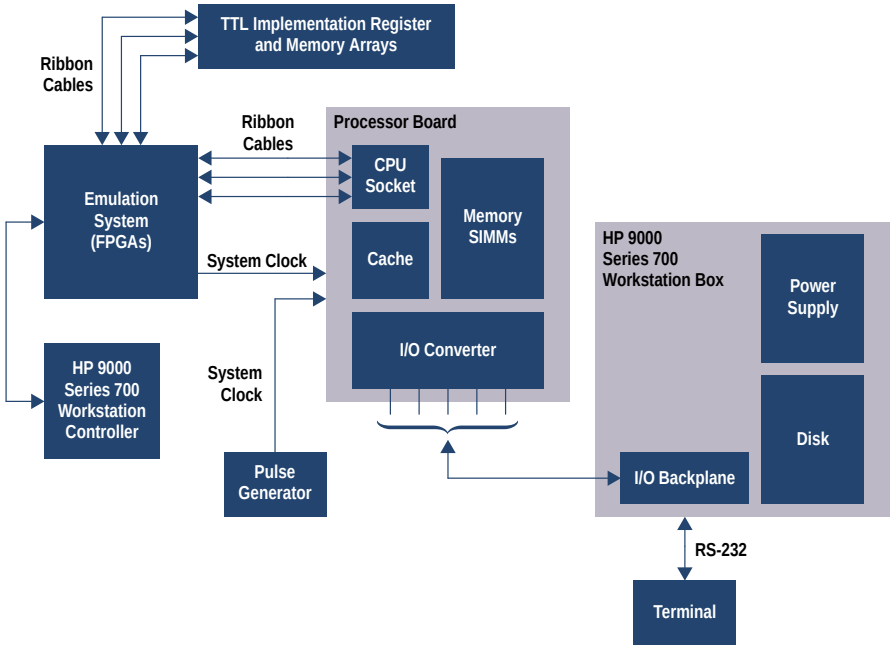


Fig. 3. Emulation setup for the PA 7100LC.

Along with these physical challenges, we also addressed modeling issues. The emulation company provided an on-site, experienced engineer to join our emulation team. The preliminary goal was to take a substantial top-level block netlist and prove that our style of custom design would emulate successfully. We chose a block that contained many unique and difficult-to-model elements. It contained custom data path blocks and some control blocks, and included some large regular arrays such as register stacks, TLB, and internal cache. Because of their size and regular structure, we chose to model the cache, register stacks, and TLB on external component boards using TTL parts and PALs. We turned to industry tools to translate our library of custom cells into emulation gates, but quickly found that the tools were incapable of generating accurate gate-level models. We were forced to create handwritten translations for the entire library to make progress.

Once we had completed this initial block, we ran the model in cosimulation mode with a Verilog simulator. The emulation hardware modeled our target block, while the Verilog simulator modeled the rest of the PA 7100LC. The models exchanged stable input and output values after every CPU clock transition. This approach allowed turn-on and testing of the external component boards as well as flushing out of modeling issues.

Next, we attacked the full chip. Our emulation team created a full chip model, which was partitioned and programmed into the FPGAs in the emulation boxes. This became a painful process as we learned that the hardware and software had never been used on a design of this size, and fatal tool failures stopped progress many times.

We achieved our first working model that ran through all the firmware code shortly after the PA 7100LC chip achieved tape release. We debugged all firmware code before first silicon arrived from fabrication. This made silicon turn-on much faster than would have been possible otherwise. We resolved some nagging emulation failure modes in the difficult-to-model floating-point circuits within one month of receiving the first silicon chips. This emulation model allowed extensive testing on the final chip specification before the masks were released to fabrication. Only one hardware bug was found using emulation.

From our emulation efforts we learned the following:

- Our method of custom VLSI design was difficult to model in emulation gates. Many unanticipated race conditions were found which had to be resolved. For example, we allow races (e.g., between a latch's data signal and its enable signal) that we can guarantee will be won on the chip. However, with uncertain delays on these signals within the FPGAs, these races are easily lost. We also found that wire-OR logic is very difficult to model.
- We found that electrical characterization was the limiting issue for shipping products in volume. Emulation does not help this problem directly. Although it does help to prevent masking bugs, it may not actually shorten the ship-release date.
- Even though custom VLSI chips are much more difficult to emulate than ASICs, in-circuit emulation is a viable technology. As emulation technology matures, the effort required to model complex CPUs will become more reasonable. Because of the immaturity of in-circuit emulation technology at the time we were using it, we were only able to make a minor contribution to the development of the PA 7100LC with this technology.

The learning curve for emulation technology was steep, but this effort can be seen as successful when used as a stepping stone to a new technology paradigm. We identified many issues and shortcomings with using current emulation technologies to accelerate vector throughput. We can now continue to move towards either applying more mature emulation technology or developing new approaches that better address the issues that we identified.

Postsilicon Electrical Verification

The goal of postsilicon functional verification is to identify failures caused by inappropriate logic within the chip. These

functional failures generally manifest themselves on every chip that we manufacture and will be unrelated to the operating point (e.g., temperature, voltage,or frequency) of the CPU.

Electrical failures are another class of failures that we sought out during the postsilicon verification effort for the PA 7100LC. Electrical failures cause the chip to malfunction and typically have a root cause in some electrical phenomenon such as:

- Ground or power supply noise on the board or chip
- Coupling between signals
- Charge sharing
- Variation in FET speed or drive capability caused by variation in the manufacturing process
- Leakage related phenomena
- Race conditions
- Unforeseen interchip circuit interactions.

Because the integrated circuit manufacturing process varies slightly with time, electrical failures may or may not be present on all chips that are produced. Further, certain operating conditions will typically exacerbate the failure. Sometimes a failure will occur at any operating point and can be difficult to distinguish from a functional failure. However, most will be dependent upon some parameter of the chip's operating point.

To deal appropriately with failures of this class, we staffed an electrical verification effort for the PA 7100LC that was mostly independent from its functional verification (described earlier). The goals of this effort were to:

- Identify, isolate to root cause, and repair all failures within the operating range possible in customer systems
- Identify and isolate to root cause any failures within a significant, well-defined region of margin outside of this operating range.

The first goal is clearly necessary to provide quality systems to customers. We created the second goal with the knowledge that in some cases, understanding the root cause for failures outside of our expected operating range would be beneficial. Sometimes this knowledge would enable us to make proactive design changes which would increase chip yields, resulting in lower chip and system costs. Such knowledge is also useful when moving the chip into a higher-frequency range or a new process technology.

To meet these goals, we instrumented several systems so that we could independently control each of the CPU supply voltages and the operating frequency of the system. We interfaced each set of controlling instruments to a host computer which could systematically vary the operating point parameters, direct the system under test to run a variety of possible tests, and observe and log the results of those tests. We placed each system under test in an environmental chamber that was capable of varying the temperature from –40°C to 100°C. In each system under test, we also varied system parameters such as memory loading and I/O bus loading.

In the presence of an electrical failure and the appropriate operating conditions, certain code streams will not evaluate as expected. To ease the task of isolating electrical failures, we created test code specifically for electrical verification that stressed the various interfaces and functional units of the chip in turn. Each segment of this test code would indicate its progress as it ran. This allowed us to isolate a failure quickly to a particular, very short segment of the test code.

In addition to this electrical verification code, we leveraged the random code generators used by the functional verification team, and ran the code sequences that they produced at the corners of the PA 7100LC's operating region.

Using this data generating and collection system, we were able to create graphs that indicated passing and failing code sequences as a function of voltage, frequency, temperature, system conditions, and IC process. By inspecting the operating point dependencies (or lack of dependencies) of a failing code stream, we could gain insight into the root cause for a failure. To confirm our root cause analyses and potential fixes, we created new handwritten test codes, altered existing silicon using focused-ion-beam milling, and performed electron beam probing of chips in systems.

The PA 7100LC's postsilicon electrical verification effort ensured that the chip would perform well in a wide range of electrical environments. It identified easily repaired yield limiters that allowed us to maximize yield and minimize the cost of the CPU. Each of these successes allowed our system partners and customers to be more successful in meeting their goals.

Debug and Test

Since the PA 7100LC processor was designed to be the core component of a low-cost workstation line, the factory cost goals and expected volumes clearly indicated that careful attention to ease of test and manufacturability was necessary. The following test features were defined based upon design and manufacturing needs:

- Parallel test vector capability in excess of 100 MHz
- IEEE Standard 1149.1-compatible boundary scan interface
- On-chip clock gating circuitry
- Retention of internal state when the chip clocks are halted
- Internal scan with single and double clock step capability
- Fully static operation to support off-chip I_{DDQ} testing
- Signature analysis capability for testing the on-chip instruction buffer
- At-speed capture of internal states by scan registers.

To meet manufacturing cost goals, the PA 7100LC had aggressive quality and test time goals compared with our previous processor designs. Both of these items significantly affect final chip cost. A test methodology was developed early in the design phase to facilitate the achievement of these goals. The methodology encompassed chip test and characterization needs and manufacturing test needs.

Testing is accomplished through a mixture of parallel and scan methods using an HP 82000 semiconductor test system. The majority of testing is done with at-speed parallel pin tests. Tests written in PA-RISC assembly code cover logical functionality and speed paths and are converted through a simulation extraction process into tester vectors. Scan-based block tests are used for circuits such as standard-cell control blocks and the on-chip instruction buffer which are inherently difficult to test fully using parallel pin tests. I_{DDQ} measurements are also performed after some parallel tests

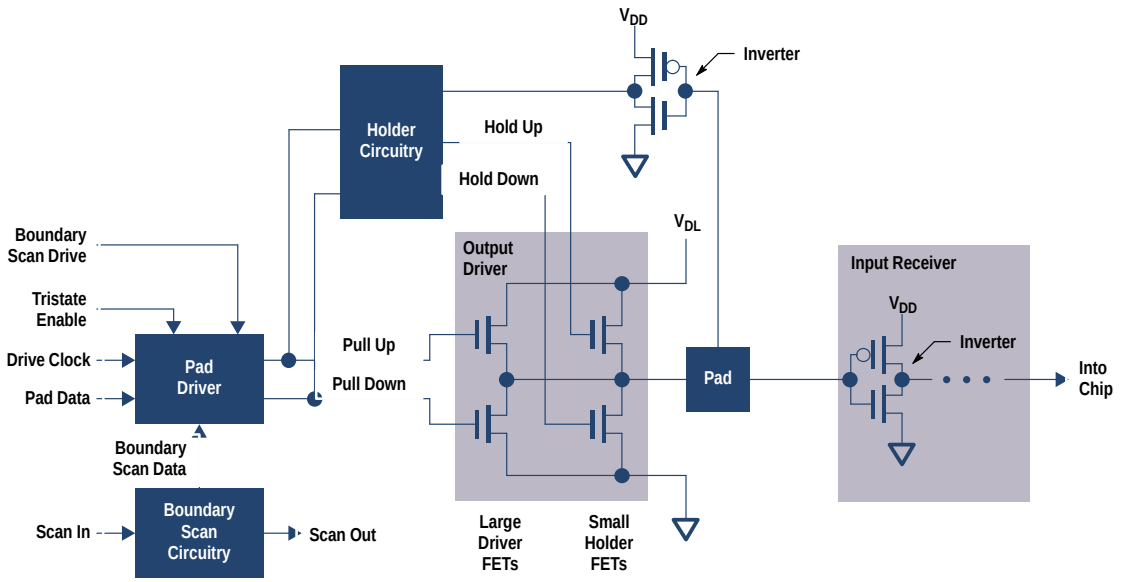


Fig. 4. Simplified diagram of a PA 7100LC I/O driver. Static current can flow from V_{DD} to ground in the inverters if the pad is not driven to V_{DD} or ground. For example, if the pad driver drives a one, the pad would be driven to 3.3V (V_{DL}), which would cause static current to flow, invalidating the I_{DDQ} test. For I_{DDQ} measurements, the pad is driven to 0V (ground) through the boundary scan circuitry and pad driver.

to provide additional defect coverage. The parallel test sequence is 600,000 states long, and 42 Mbits of scan vectors are used during scan testing.

To meet our test quality and cost goals, we implemented two new chip-test techniques that had not been used on previous PA-RISC implementations: I_{DDQ} testing and sample-on-the-fly testing.

I_{DDQ} Implementation

I_{DDQ} testing is a test methodology in which the presence of defects is detected by measuring dc current when the chip is halted. Nondefective full CMOS gates draw static current made up of leakage currents that are in the nA range. However, defective gates can draw currents many orders of magnitude higher. If a current measurement is made on the power supply during a static state, a good chip will draw very little current and a defective chip will draw much more. I_{DDQ} has high observability and detects many different types of defects. It was decided early in the design of the CPU that I_{DDQ} test capability would be a desirable test feature. I_{DDQ} test capability was also desirable because it substantially reduces static power consumption.

Design Rules. To support I_{DDQ} testing, most of the circuits leveraged from past PA-RISC implementations that drew dc current were eliminated. For each case in which using a circuit that drew static current was the only reasonable design solution, the circuitry was redesigned to be disabled with a test signal during I_{DDQ} measurements. Most blocks containing pseudo-NMOS circuitry were redesigned using static CMOS circuitry. Dynamic circuits were modified to eliminate static current and to retain state while the chip is halted. No FET gate is allowed to be in a situation where it could float if the clocks are halted because this could possibly cause the FET to turn on. Internal pullups on input pins are disabled during I_{DDQ} measurements, including the IEEE 1149.1 test pins. No drive fights are allowed in a static state. All nodes make a full transition to a supply rail, which is accomplished through the use of restorative static feedback when full CMOS transfer gates are not used in latches and multiplexers. Any bus that could be completely tristated in any state uses a bus holder circuit to maintain proper levels.

Special Considerations. The floating-point ALU, which was leveraged from the PA 7100 processor, drew static current and redesigning it was not feasible given our schedule constraints. However, it is possible to eliminate the static current during I_{DDQ} measurements if the ALU is not evaluating during the measurement. Since I_{DDQ} testing was not going to be used to test the ALU, this was acceptable. I_{DDQ} testing during parallel vectors is still possible, but if a floating-point operation occurs that uses the ALU, the ALU loses its internal state if I_{DDQ} test mode is enabled during the test.

Another area of consideration for I_{DDQ} involved the I/O bit slices. The CPU uses two power supplies, V_{DD} and V_{DL}, which are nominally at 5V and 3.3V respectively. V_{DD} supplies all of the internal chip logic, while V_{DL} is the supply for the output driver pullup FETs. The input receivers on the CPU normally draw static current when an output driver is on that drives to V_{DL}. In addition, a circuit to hold the current value on the pad can draw static current if the pad is not driven to V_{DD} or ground. Therefore, when I_{DDQ} measurements are taken, the output drivers are driven to ground through the use of the boundary scan circuitry to eliminate static current flow in the receiver and pad holder circuits (see Fig. 4). The parallel tester drives input-only pins to V_{DD} or ground as appropriate, including the IEEE 1149.1 interface pins. The analog inputs of the clock buffers are also driven to appropriate values to prevent static current.

These rules were easy to adhere to and followed our rationale to increase test capability with little design impact. I_{DDQ} compliance was verified by running functional simulation cases through an HP proprietory FET-level switch simulator which also has the ability to check for static current

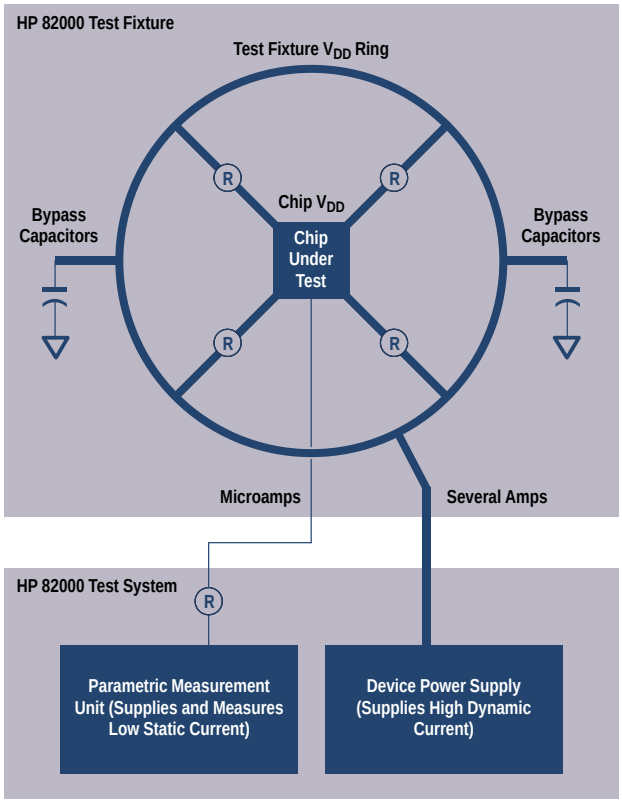


Fig. 5. I_{DDQ} measurement setup.

violations. Because of careful attention to the design guidelines, only six I_{DDQ} violations were discovered when the simulations were run, all of which were easily resolved.

I_{DDQ} Measurement

I_{DDQ} measurements are taken using a parametric measurement unit on the HP 82000 tester (see Fig. 5). When a measurement is to be taken, a vector sequence is run to place the device under test (DUT) into a static state. After the dynamic current transients have settled, the measurement unit is connected to the chip power plane with a relay, and the regular V_{DD} supply is then switched out with relays. The parametric measurement unit then supplies and measures the current flowing into the DUT. The power plane for the DUT is separated from the test fixture power plane by relays connected between the chip and the test fixture. Bypass capacitors to control supply noise are placed on V_{DD} on the power supply side of the relays. This is important because leakage currents in large electrolytic capacitors can be tens of microamps, which would compromise the accuracy of the measurement.

Typical measurements are in the range of 1 μA. The I_{DDQ} current is dominated by reverse bias leakage current and subthreshold leakage. Measurements are taken during wafer and package test, and four measurements are made. Four parallel vectors are used, which initialize the registers, cache, TLB, and other state logic to zeros or ones and two patterns of alternating ones and zeros (to check for bridging faults). This provides a great deal of defect coverage while incurring minimal test overhead.

I_{DDQ} testing was very effective at catching defects on the PA 7100LC. Results indicate that 50% of scan test failures and 70% of parallel failures are caught by I_{DDQ} testing. In addition, other types of defects are caught that might not be caught by conventional voltage-level testing, like gate oxide shorts and some types of bridging faults. These can lead to reliability problems over the life of the product, so it is important to catch them at the chip test stage.

We plan to do more directed I_{DDQ} testing on future chips, using scan testing and parallel testing to set up and measure current for specific chip states indicated by automatic test generation tools. This should improve the level of coverage we get for I_{DDQ} tests. However, one problem that may occur is that off-FET leakage will increase in the effort to improve FET performance in future IC processes. This has a direct effect on the ability of I_{DDQ} techniques to resolve low current defects. Additional techniques like power supply partitioning may be necessary to make I_{DDQ} usable with more advanced IC processes.

Sample-on-the-Fly Testing

An interesting new feature that is implemented on the CPU enables scan registers to capture the internal state of the chip while the chip is operating at speed in a normal system. We refer to this capability as sample-on-the-fly testing. The sample is nondestructive, and the data can be accessed while the chip continues to execute code by scanning the results out using the on-chip IEEE 1149.1-compatible test access port (TAP). This feature was very useful for debugging and characterizing system-level performance because it is essentially a logic analyzer built directly into the chip which allows access to over 4000 internal state values. Samples can be taken with any IEEE 1149.1-compatible test controller and appropriate software.

Internal Sampling. The internal sampling capability allows a sample to occur when the architected PA-RISC interval timer reaches a count that matches a preset value in a register and the TAP circuitry is in a specific state. In the PA 7100LC the interval timer on the chip is a 32-bit register that increments by one for every clock cycle that occurs on the chip. An additional 32-bit register provides a value to compare with the value in the interval timer register. This value can be set by doing a PA-RISC mtctl (move to control register)† instruction. When the interval timer value matches the value set by the mtctl instruction, a comparator circuit generates a signal which is normally sent to the control logic to cause an interval timer interrupt to occur. This signal is also sent to the TAP in this implementation. If the current TAP instruction is ISAMPLE, the state of the chip is sampled into each scan register on the following chip state by allowing each scan register to update during the phase when the functional latch is not being updated. An indication that a sample has occurred is sent from one of the test pins when the sample is taken. The pin can be monitored by an external IEEE 1149.1-compatible controller system to determine when data can be shifted out of the chip. The shifting of the sampled data does not corrupt the state of the internal logic.

† This instruction moves data to a control register. In this instance it is moving data to the timer comparison register.

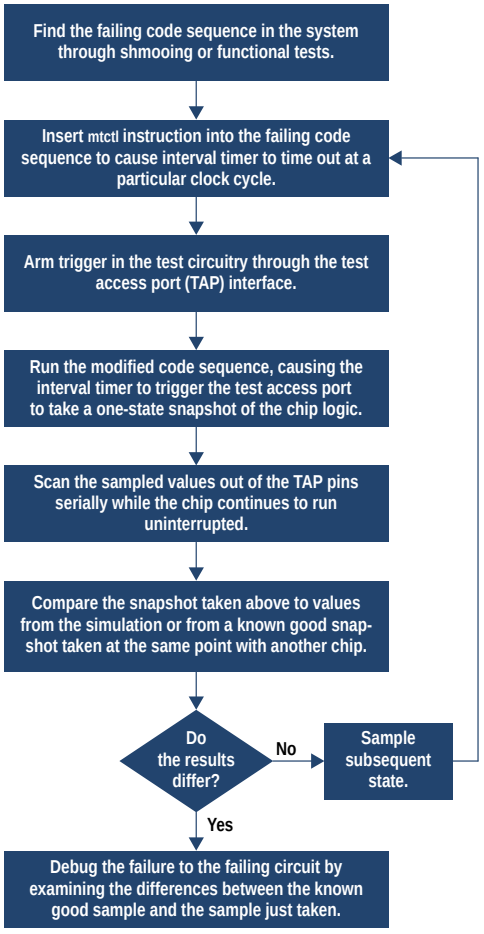


Fig. 6. Sample-on-the-fly testing process.

If another sample is desired, the above procedure is simply repeated. Fig. 6 summarizes the sample-on-the-fly process.

Results. Although sample-on-the-fly testing capability required careful electrical and timing design, it has proven to be very effective for debugging. It was vital at system frequencies approaching 100 MHz, since our traditional external debugging hardware was unable to function at this frequency because of electrical constraints. Sample-on-the-fly testing became our only debugging tool in systems with high-frequency critical paths. It was used several dozen times in high-speed characterization and led to the resolution of several slow timing paths. It is clear that as CPU frequencies increase, more debugging circuitry will need to be included directly on the chip to assist in diagnosing functionality, speed, and electrical failures.

Debug Mode

The sample-on-the-fly technique allowed us to observe the values present at many nodes, at one very specific point in time, and at any operating frequency. Since this test technique uses the test access port to observe these values, it provides information about the chip state at a relatively low bandwidth. This information is an extremely valuable diagnosis tool for designers because it enables them to know exactly when a problem is occurring.

Sometimes, especially when a problem is not yet fully understood, a higher-bandwidth path to diagnostic information is useful to designers. To allow designers access to larger amounts of information across broad slices of time, we added a debug mode to the PA 7100LC. This mode makes available externally the values of several key internal buses and control interfaces, on a state-by-state basis.

Software can place the chip in the debug mode by executing a series of CPU diagnostic instructions. Software can also be used to choose a set of signals to be made externally visible. These signal sets were carefully chosen by the chip's designers as being indicative of the internal state of the CPU. Examples of signal sets that can be made visible using the debug mode include:

- Internal instruction and data buses
- CPU to memory and I/O controller interface
- Key cache controller state information.

When the chip is operating in the debug mode, it identifies unused cycles on the I/O bus and uses them to drive the selected debug information onto the I/O bus. The debug circuitry can be programmed by software either to throw away debug data during states when the I/O bus is unavailable, or to cause the CPU pipeline to stall during these states so that no debug information is lost.

Externally driving debug information allows engineers to see a sufficient amount of state information on a large enough number of CPU states to be able to quickly direct further efforts at locating postsilicon problems.

Both debug mode and sample-on-the-fly turned out to be invaluable debugging aids in the highly integrated environment of the PA 7100LC.

Conclusion

Supporting design methodologies allow implementation of the features that a product requires to meet its design goals. The methodologies used to synthesize, place and route, simulate, verify, and test the PA 7100LC processor were crucial to the processor's success.

References

1. P. Knebel, et al, "HP's PA 7100LC," A Low-Cost Superscalar PA-RISC Processor, *Compcon Digest of Papers*, February 1993, pp. 441-447.

2. S. Undy, et al, "A Low-Cost Graphics and Multimedia Workstation Chip Set," *IEEE Micro*, Vol 14, no. 2, April 1994, pp. 10-22.

3. T. Asprey, et al, "Performance Features of the PA 7100 Microprocessor," *IEEE Micro*, Vol. 13, no. 3, June 1993, pp. 22-35.

4. E. DeLano, et al, "A High Speed Superscalar PA-RISC Processor," *Compcon Digest of Papers*, February 1992, pp. 116-121.

An I/O System on a Chip

The heart of the I/O subsystem for the HP 9000 Model 712 workstation is a custom VLSI chip that is optimized to minimize the manufacturing cost of the system while maintaining functional compatibility and comparable performance with existing members of the Series 700 family.

by Thomas V. Spencer, Frank J. Lettang, Curtis R. McAllister, Anthony L. Riccio, Joseph F. Orth, and Brian K. Arnold

The HP 9000 Model 712 design is based on three custom pieces of VLSI that provide much of the system's functionality: CPU, graphics, and I/O. These chips communicate via a high-performance local bus referred to as GSC (general system connect). This paper will focus primarily on the I/O chip.

A major goal of the Model 712 I/O subsystem was to provide a superset of the I/O performance and functionality available from other family members at a significantly reduced manufacturing cost. This goal was bounded by the reality of a finite amount of engineering resources, and it was obvious from the start that integrating several of the I/O functions onto a single piece of silicon could greatly reduce the total I/O subsystem manufacturing cost. Each function of the I/O subsystem was examined individually as a candidate for integration. The value of maintaining exact driver-level software compatibility was also evaluated with respect to the advantages of minimizing the hardware cost for each of the I/O functions.

The investigation indicated that the optimal solution for the Model 712 was an I/O subsystem that centered around a single piece of custom VLSI. The chip that resulted from this investigation directly implements many of the required I/O functions and provides a glueless interface between the GSC bus and other common industry I/O devices. This chip was named LASI, which is an acronym that refers to the two major pieces of functionality in the chip, LAN and SCSI. The LASI chip also provides several miscellaneous system functions that further reduce the amount of discrete logic required in the system.

Chip Overview

The LASI chip was designed in a 0.8-μm CMOS process and is 13.2 mm by 12.0 mm in size (including I/O pads). It contains 520,000 FETs and is packaged in a 240-pin MQUAD package. LASI dissipates approximately three watts when operating at the maximum GSC frequency (40 MHz). LASI was designed primarily using standard-cell design methodologies although several areas required full custom design.

A functional block diagram of LASI is shown in Fig. 1. The majority of circuitry in LASI is consumed by only two functions, LAN and SCSI. Both of theses designs were purchased from outside companies and ported to HP's design process. The SCSI functionality is exactly identical to the NCR 53C710 SCSI controller, and the LAN functionality is exactly identical to an Intel 82C596 LAN controller.

Other I/O functionality that is completely implemented on LASI with HP internal designs includes: RS-232, Centronics parallel interface, a battery-backed real-time clock, and two PS/2-style keyboard and mouse ports. In addition, LASI provides a very simple way of connecting the WD37C65C flexible disk controller chip to the GSC bus. The system boot ROMs are also directly controlled by the LASI chip. The Model 712 provides 16-bit CD-quality audio and optionally supports two telephone lines. LASI provides the GSC interface and clock generation (using digital phase-locked loops) for both of these audio functions. Fig. 2 shows an approximate floor plan of the LASI chip. The floor plan shows the general layout and relative size of each block.

LASI contains several system functions that help to minimize the miscellaneous logic required in the system. This includes GSC arbitration and reset control. LASI also serves as the GSC interrupt controller.

It is possible to use up to four LASI chips on the same GSC bus. LASI can be programmed at reset to reside in one of four different address locations. The arbitration circuit supports chaining, and LASI can be programmed to either drive or receive reset.

System Support Blocks

The following sections give a brief overview of each of LASI's major functional blocks that provide system support functionality in the Model 712, but do not directly support or implement any I/O function.

GSC Interface. The GSC (general system connect) bus connects the major VLSI components in the Model 712. It is a 32-bit bus with multiplexed address and data. The bus consists of 47 signals for devices capable of being a bus master. The GSC bus is defined to run at up to 40 MHz giving a peak transfer rate of 160 Mbytes/s.

The GSC interface block in LASI provides the connectivity between the GSC bus and the wide variety of internal bus blocks, many of which have different logical and timing requirements. This block converts the GSC bus to a less complex internal LASI bus. The LASI internal bus is very similar to the GSC bus, but it is not as heavily multiplexed and is more flexible than the GSC bus in that it easily accommodates the simpler interface for the general-purpose I/O blocks in LASI. The GSC interface block handles bus errors and keeps track of parity information for other internal

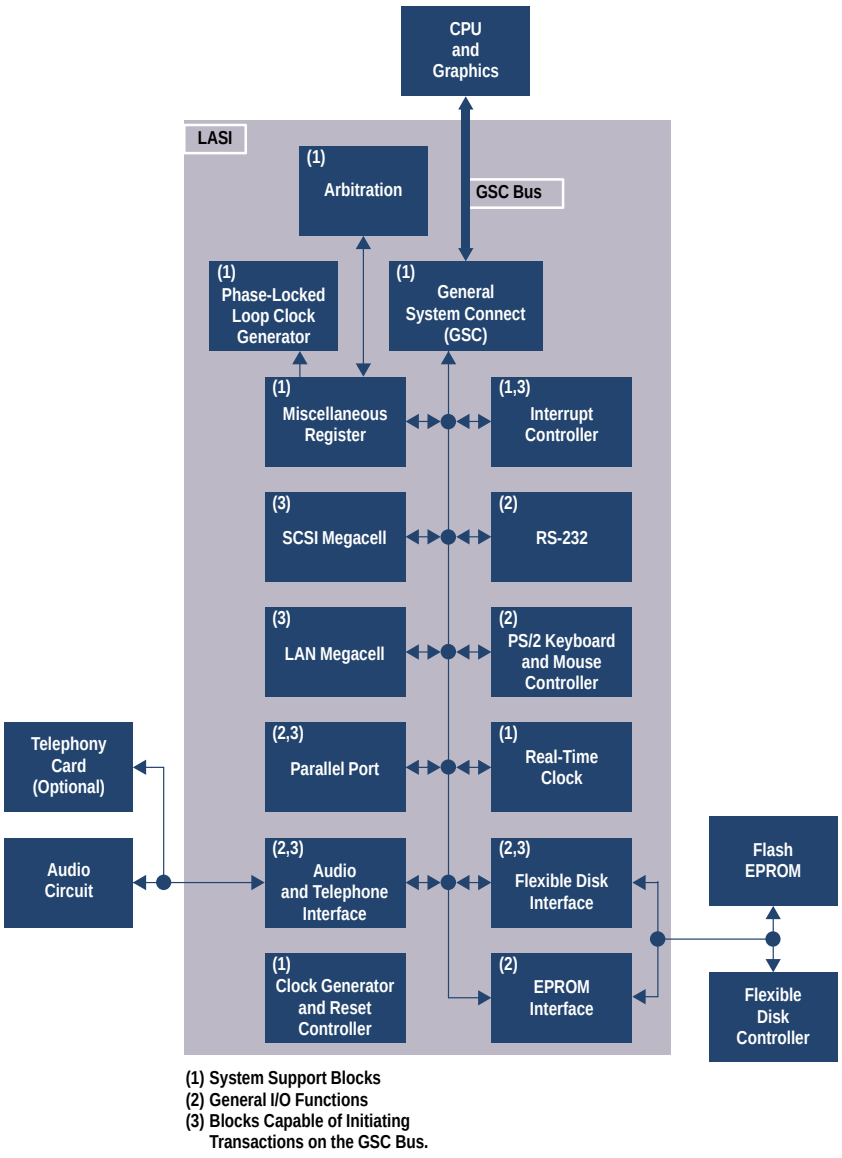


Fig. 1. LASI chip block diagram.

blocks, removing the associated complexity from these controllers. Both master and slave devices reside on the LASI internal bus.

LASI is a slave whenever the CPU initiates data transfer. As a slave, LASI supports only subword and word write, and subword, word, and double-word reads.* Internal slave devices only need to support a subset of these transactions. There are five different protocol behaviors for slave devices in LASI: unpaced byte wide, paced byte wide, packed byte wide, unpaced word wide, and paced word wide.

Unpaced devices, such as the real-time clock, don't use a handshake with the GSC interface, making their protocol very simple. When a device requires a variable length of time to transfer data it is called paced. The SCSI interface is an example of a paced device. A packed device is one that sends a sequence of bytes to make up a word or double word. The boot ROM interface is an example of a packed device.

* In PA-RISC a subword is typically one byte, a word is 32 bits, a double word is 64 bits, and a quad word is 128 bits.

A simple strobe signal is asserted while internal data and address buses are valid. Internal devices have no direct interaction with bus errors.

As a bus master, LASI is capable of initiating subword, word, double-word, and quad-word transactions on the GSC bus. Once one of LASI's internal bus masters owns the bus, it can signify the start of a transaction by asserting the master_valid signal (see Fig. 3). The device must then simultaneously drive its DMA address (master_address), transaction type, and byte enables onto the bus. On a read, the first available data word will appear on the internal bus when the master_acknowledge signal is asserted by the GSC interface. The GSC interface will not accept another master_valid until all the read data has been transferred.

If a timeout error, address parity error, or data parity error is encountered on the GSC bus, the GSC interface will always do a normal handshake for the transaction by asserting the master_acknowledge signal. The transaction will complete as usual except that an error is logged, disabling arbitration for the device so it cannot be a bus master again. This means

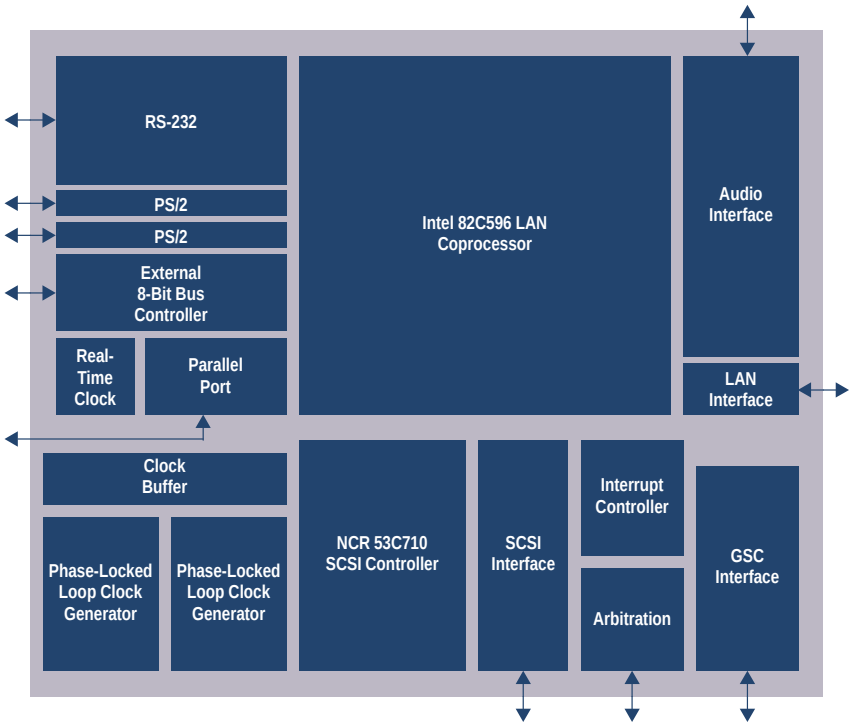


Fig. 2. LASI floorplan.

that internal masters, at the hardware level, never need to respond directly to bus errors. When the GSC interface block sees a timeout error it will, from the perspective of its internal bus blocks, complete a transaction normally. In this way the GSC's error signaling mechanism can correctly terminate an errant transaction without adding complexity to LASI's internal blocks.

Parity is generated in the GSC interface whenever LASI sources data or an address on the bus. Parity is checked whenever LASI is a data sink. LASI does not respond to address parity errors on the GSC bus, which result in a timeout error.

Arbitration. LASI contains six different blocks capable of initiating a transaction on the GSC bus (see Fig. 1). To initiate a transaction, a block must first own (or gain control of) the GSC bus. Deciding which potential master owns the bus is the job of LASI's arbitration block. The arbitration circuit in LASI provides internal bus arbitration for all six internal devices and provides external GSC arbitration signals for the CPU and an expansion slot. This capability allows LASI to function as the central arbiter for the GSC bus in low-end systems. The arbitration circuit can also be pin-programmed at reset to behave as a secondary arbitration device that is controlled by another arbiter. This feature allows LASI to be used in larger systems that provide their own arbitration circuit. A second LASI can also be used for I/O expansion in low-end systems in which the first LASI is providing the central arbitration. Support for multiple LASI's on the same GSC bus makes the speedy development of multifunction I/O expansion boards a relatively simple task.

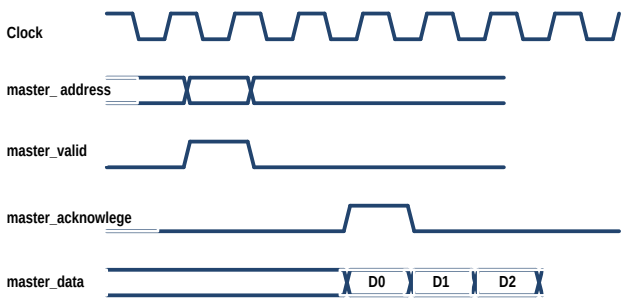


Fig. 3. Master read timing diagram.

The LASI design was simplified by requiring that the LASI arbitration circuit gain control of the GSC bus before granting the internal bus to potential bus masters. This saved a significant amount of complexity in the GSC interface block as well as greatly reducing the number of cases that needed to be tested during the verification effort. This simplification does create a couple of wasted GSC cycles for each transaction initiated by LASI. However, this inefficiency has a negligible impact on system performance.

The LASI arbitration circuit provides a simple round-robin scheme that provides roughly equal access to all devices. The arbitration circuitry keeps track of the identity of the last device granted the bus and all currently outstanding requests. (A simple truth table makes sure the GSC resource is handed out fairly.) If no devices are requesting the bus, LASI will default to granting the bus to the CPU. This has a small positive impact on performance, given that the CPU is the most likely device to initiate the next transaction. This arbitration scheme helps simplify the arbitration circuit by not requiring it to monitor bus activity. Each bus master is responsible for being "well-behaved" with respect to bus use.

The arbitration circuit plays a key role in the error handling strategy for LASI. If an error occurs on the GSC bus while LASI is the bus master, the arbitration circuit will not grant the bus to additional internal devices until the CPU clears the error by clearing a bit in the arbitration circuit. This simplifies the design of other devices within LASI by not requiring

them to use the error signal as an input to their state machines. When an error is detected, the transaction will terminate normally, but no additional transactions will be allowed until the situation is rectified by software.

Interrupt Controller. A total of 13 different interrupt sources exist on the LASI chip. Each interrupt source drives a single signal to the interrupt controller block. When the interrupt signal is asserted, the interrupt controller block will master the bus and issue a word write to the I/O external interrupt register (IO_EIR), which is physically located in the CPU. The data transferred to the IO_EIR contains a value that indicates the source of the interrupt. The address of the IO_EIR and the interrupt source value can be programmed by writing to the interrupt address register located in LASI's interrupt controller block. Individual interrupt sources can be masked by setting bits in the interrupt mask register.

LASI's interrupt controller is designed to provide a variety of interrupt approaches. The Model 712 uses only one of these alternatives. Asserting an interrupt causes a write to the IO_EIR to be mastered on the GSC bus. Upon receiving an interrupt from LASI (via IO_EIR), the CPU will read the interrupt request register located in LASI's interrupt controller block. One bit in the interrupt request register is designated for each potential interrupt source in LASI. The interrupt request register is cleared automatically after it is read by the CPU.

Real-Time Clock. The Model 712 needs to keep track of time when the system power is off. To this end, LASI provides a battery-backed real-time clock. The real-time clock is logically very simple and consists of a custom oscillator circuit and a 32-bit counter that can be read and written to by software. The 32-bit counter is used to keep track of the number of seconds that have elapsed from some reference time.

The oscillator unit operates at 32.768 kHz and typically uses less than 10 μA of current when operating on battery backup. It uses a minimum of external circuitry (consisting of two capacitors, a crystal, and a resistor) to accomplish its task.

Inside the LASI real-time clock, the 32-kHz signal is reduced to a 1-Hz signal by a 15-bit precounter. The 1-Hz signal is then used to increment the main 32-bit counter. Both the counter and the precounter are implemented using simple ripple counters. The 15-bit precounter is always cleared when software writes to the 32-bit counter.

Phase-Locked Loop Clock Generators. The goal for the LASI clock subsystem was to generate all the I/O subsystem clocks from one crystal oscillator over a wide range of system frequencies. The LASI clock block generates five different clock frequencies required for the wide variety of I/O interfaces. Three of these clocks are subharmonics of the processor clock, and are generated using simple digital state machines. However, the 40-MHz clock and the audio sample clock are fixed-frequency clocks. The 40-MHz clock is used for the SCSI back end and RS-232 baud rate generator, and the audio sample clock is used for the external CODEC chip. The frequency of this clock (16.9344 MHz to 24.576 MHz) is selectable on the fly by the audio and telephone interface.

Two digital phase-locked loop circuits are provided in LASI to generate the two fixed-frequency clocks from the CPU

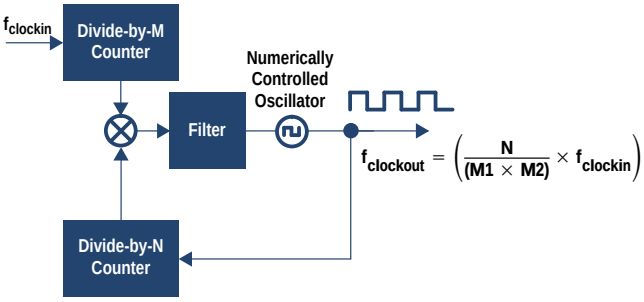


Fig. 4. Phase-locked loop clock controllers.

clock. These digital phase-locked loops implement the equation: $f_{clockout} = (f_{clockin} \times N)/(M1 \times M2)$, where N, M1, and M2 are digital coefficients stored in LASI control registers. $f_{clockin}$ comes from the main system reference clock. The $f_{clockout}$ from one of the phase-locked loops is used for the audio clock, and the $f_{clockout}$ from the other phase-locked loop is used for SCSI, RS-232, and other I/O functions. At power-on, the processor initialization code (stored in the flash EPROMs) loads the coefficients corresponding to the processor clock for the particular product. The audio sample clock has two sets of coefficient control registers, which are selected by a multiplexer based on a signal from the audio interface. Fig. 4 shows one of the phase-locked loop circuits.

The phase-locked loop circuits are completely digitally controlled, including a digitally controlled oscillator, digital phase detector, counters, and scan test hardware. This design eliminates analog control voltages which are susceptible to noise and integration errors. The digitally controlled oscillator is a ring oscillator with a digitally programmable delay element. This design is capable of generating frequencies of up to 135 MHz. A combination of custom and standard-cell design techniques are used in this design. Each phase-locked loop cell measures 1500 μm by 890 μm.

General I/O Functions

The blocks shown in Fig. 1 that make up the general I/O functions include the parallel port, audio and telephone interface, RS-232 port, and flexible disk and boot ROM interface. These I/O functions originate from HP internal standard-cell designs that were originally designed using Verilog RTL models and then synthesized into a standard-cell design using Synopsys. Some blocks were designed specifically for LASI while others were leveraged from previous HP ASIC designs.

Parallel Port. The parallel port is designed to be software compatible with previous generations of HP 9000 Series 700 I/O subsystems while minimizing overall complexity and chip area. This port allows interfacing to printers and other peripherals supporting the industry-standard Centronics parallel interface. The parallel port signals are driven directly from the LASI chip without additional buffering.

DMA was supported on previous workstation controllers and therefore needed to be provided on LASI's controller. However, since no central DMA controller exists, all DMA hardware is contained within the parallel I/O block. Since parallel port bandwidth requirements are fairly modest

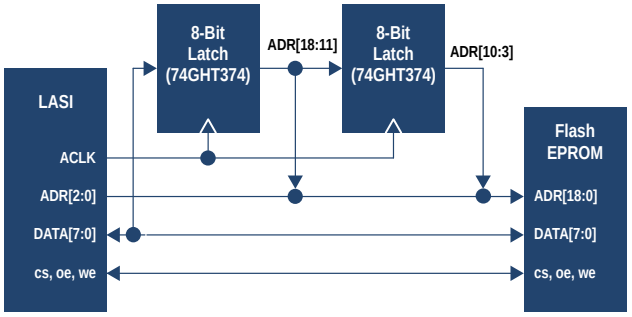


Fig. 5. Address latching logic, and the data and control lines associated with the external 8-bit bus.

(about 400 kbytes/s), DMA is done by reading one 32-bit word of data, releasing the bus, transferring one to four bytes of data over the interface, and then requesting the bus again. This approach keeps the DMA controller quite simple while easily accommodating byte unpacking.

Keyboard and Mouse Controller. LASI provides support for two IBM PS/2-style keyboard and mouse devices, making the keyboard and mouse ports just like those used on a standard IBM personal computer. These interfaces are new to the Series 700 family so there were no software compatibility issues, allowing us to optimize the design for low manufacturing cost. The interface provides only a minimal amount of hardware and relies on the driver to do most of the work. The interface also performs the serial-to-parallel and parallel-to-serial conversion and does a small amount of buffering. An interrupt is generated for every byte of data received from the PS/2 device. The software overhead is not a performance issue because of the extremely low data rate of the interface.

Flexible Disk and Boot ROM Interface. LASI supports an external 8-bit bus that provides the capability to connect discrete flash EPROM devices and a flexible disk controller with very little additional logic. Fig. 5 shows a simple schematic of a flash EPROM and the required address latching logic on the 8-bit bus. It was not cost-effective to integrate these devices into the LASI chip. The 8-bit bus is also capable of supporting other types of 8-bit devices, giving some degree of flexibility to the I/O system.

The 8-bit bus supports 1M bytes of address space (the first half of the LASI address space). All transactions to this address space on the 8-bit bus begin with two address cycles. These cycles transfer bits 18:3 of the address to two 74GHT374-type 8-bit latches wired in series and controlled by LASI. Multiplexing the address on the data lines saves 15 pins on LASI.

LASI is capable of supporting byte, word, and double-word reads and byte writes to devices on the 8-bit bus. Word and double-word reads are accomplished by doing multiple accesses to devices on the 8-bit bus and packing the bytes into words before returning them on the GSC bus. Word and double-word accesses require the address to be latched only once since LASI drives the lower three address bits directly. This greatly reduces the word and double-word access time. Double-word reads take approximately 75 GSC cycles to complete because eight accesses are required on the 8-bit bus. During each of the eight accesses a new address is presented to the flash EPROM which results in valid data being driven to the 8-bit bus by this flash device. Byte accesses are also relatively slow (12 GSC cycles) to support very slow devices on the 8-bit bus. It is important to note that the 8-bit bus is not electrically connected to the GSC.

LASI is designed specifically to support the WD37C65C flexible disk controller on the 8-bit bus. The Model 712 uses a personal computer style flexible disk controller instead of a SCSI-based flexible disk controller because of the significantly lower cost of the drive mechanism. The flexible disk controller was not integrated into the LASI chip because of the low cost of the WD37C65C chip and the potential for SCSI drives to come down in cost in the future. The WD37C65C shares the data bus and two control lines with other devices on the 8-bit bus, but does not consume any of the 1M bytes of allocated address space. Supporting the WD37C65C requires six dedicated signals and no external glue logic. LASI supports the WD37C65C running in DMA mode and provides the capability to move data directly between main memory and the WD37C65C without processor intervention.

RS-232. The RS-232 block in LASI is an HP internal standard-cell design that emulates the behavior of the National Semiconductor NS16550A. The Verilog HDL description for this design was leveraged from previous HP ASIC designs used in other members of the HP 9000 Series 700 workstation family.

One difference between this block and the NS16550A is that its baud clock is derived from a 40-MHz signal. This allows the block to share the phase-locked-loop-generated 40-MHz clock with the back end of the SCSI block and eliminates the need to support an external crystal or dedicated phase-locked loop for baud clock generation.

Audio Interface. The Model 712 supports built-in CD-quality audio and an optional telephony card.[1] The telephony card is DSP-based and provides simultaneous access to two telephone lines both capable of supporting voice, fax, or data modems. LASI provides the interface between the GSC bus and the audio and telephony circuitry.

An objective for the Model 712 audio subsystem was to maintain complete software compatibility with previous discrete designs. As a result, a good deal of the audio interface circuitry on LASI is dedicated to supporting this compatibility and is not optimized for minimal manufacturing cost.

The audio interface in LASI has two DMA channels that support the input and output audio streams. Each channel has two 4K-byte pages of main memory continually reserved for transferring data to and from the CS4215 CODEC. The buffering in the interface is sufficient to guarantee isochronous audio operation, given worst-case GSC bus latencies in the Model 712. A wide range of audio formats is supported including 8-bit or 16-bit words sampled in either linear, μ-law, or A-law format at a variety of sample rates from 8 kHz to 48 kHz.[1] The clock that determines the sample rate in the CODEC is generated in one of LASI's programmable phase-locked loop circuits. Communication between LASI and the CODEC is accomplished via a full-duplex, serial bit stream.

The high-speed serial bus over which LASI communicates with the daughter card is similar to a concentrated highway bus developed by AT&T but has several modifications. The core pinout is the same using the signals data transmit (DX), data receive (DR), and frame synchronization (FS), but the definition of the bus has been extended to incorporate control of external buffers and bus reset.

Communication with the telephony card is accomplished via two TTY channels internal to LASI. The serial concentrated highway bus data is multiplexed onto the high-speed serial stream and sent to the CODEC and the telephony card. Since TTY devices are used, the driver for the telephone system is a highly leveraged version of the existing TTY drivers. The audio interface and HP Teleshare[2] have a common digital interface which resides in LASI. HP Teleshare is described in more detail in the article on page 69.

Megacell I/O Functions

LASI contains two megacells whose designs were purchased by HP from external vendors. The decision to do this was based on maintaining software compatibility with past HP 9000 Series 700 workstations and the availability of engineering resources in HP. In both cases, an important goal was to maintain the integrity of the megacell as much as possible. A definite boundary was drawn between functionality leveraged from external vendors and new design work. This boundary proved vital to functional verification and production testing.

LAN Megacell. IEEE 802.3 LAN support is provided by a megacell derived from the Intel 82C596 LAN coprocessor. To understand the integration, two key areas should be considered. First, importing the megacell at the artwork level solved some problems and imposed others. Second, in the area of interfacing, the integrated megacell eliminated a substantial number of chip pins but raised some protocol issues that had to be overcome.

The LAN megacell was imported into our IC design flow at the artwork level. Because the original Intel design was done in a custom fashion, a netlist translation would have required a significantly longer design time and a much larger manpower deployment than the artwork translation. Even at the artwork level, several modifications were made because of differences between the original CMOS process design rules and those of our target process.

One challenge in importing the megacell at the artwork level was developing a verification strategy that allowed concurrent simulation of the megacell and the rest of the chip. Because the megacell vendor used proprietary simulators running in a mainframe environment, the vendor simulation models couldn't be used in our Verilog-based environment. Hardware modeling was explored, but characteristics of the part made this solution impractical. Converting either functional representations or transistor-based representations to Verilog HDL raised too many concerns about modeling accuracy. In view of these roadblocks, an unconventional approach to simulation modeling was employed. First, FET-level model was extracted from the artwork. This model was turned on and verified using Intel's production test vectors and a proprietary in-house simulator. Second, the in-house simulator was compiled and linked into the Verilog simulator using a procedural-level interface. Third, a Verilog HDL interface module was written that defined synchronization events for data transfer between the two simulators, and the model was reverified using production vectors. Finally, tests were run that were specifically designed to test the interface between the megacell and the internal bus.

Integrating the LAN megacell did provide a clear win by improving the ratio of I/O to core area. When sold as a separate device, the Intel 82C596 has 89 signal pins devoted to the host interface. Once the megacell was integrated, all of these signals remained on-chip. In addition, 77 of the removed signal pins had output drivers, so the associated power and ground pins were eliminated.

The megacell did require a small amount of circuitry to interface the 82C596 bus to the LASI internal bus. The primary difficulty in this area was burst transactions. The system bus wanted to know at the start of the transaction how many words were to be bursted. In contrast, the 82C596 burst protocol would only indicate whether or not it had one more word to burst. To minimize complexity and avoid the area associated with a FIFO buffer, the decision was made to support only two-word bursts. This logical intersection of the two bursting protocols provided a bandwidth utilization improvement over nonbursted transactions while minimizing chip area and development time.

SCSI Megacell. To provide SCSI-2 support, LASI uses an NCR 53C710 megacell. This megacell was imported into our design methodology as a netlist port. The design was translated from NCR's standard-cell library to HP's cell library. A few unique components were added to HP's library specifically to support the SCSI megacell. While this created some challenges, doing a schematic port allowed more flexibility to optimize the aspect ratio of the megacell for a more efficient floorplan. This technique also masked differences between NCR's process and HP's process. Verilog models for this schematic port were simulated in the conventional way.

The programming and SCSI bus model for the 53C710 megacell is completely compatible with the industry-standard component marketed by NCR. However, the host side interface of the megacell is modified to eliminate the pads and replace them with standard-cell components. These components connect directly to internal megacell signals, providing an interface to the chip's internal bus.

The 53C710 can be a master and a slave device on the GSC bus. LASI's internal bus protocol for slave transactions requires only combinational logic between the megacell and the internal bus. As a slave, byte and word transactions are supported in the megacell. If SCSI is a bus master, the interface supports all the transaction types needed by the megacell, with the help of a small state machine located in the SCSI interface block shown in Fig. 2. SCSI data is typically transferred using four-word read and write transactions on the GSC bus.

Test Support

The primary objective for LASI testing was to provide an extremely high level of coverage with a limited amount of test development resources. Test support was complicated by the diverse nature of the circuits on LASI.

The non-megacell functionality is tested by a combination of parallel pin vectors in conjunction with automatically generated scan vectors. LASI has an enhanced JTAG (IEEE 1149.1) test block, 25 distributed internal I/O device scan chains, and embedded test functionality in the I/O pads. The JTAG test block contains a test access port and boundary-scan architecture defined in IEEE standard 1149.1-1990 and private instructions used for clock control, full-chip step control, and specific scan-chain functions.

To maximize test coverage for the two megacells and to minimize the required test development resources, the vectors used for production testing by Intel and NCR are used on LASI. Doing this requires multiplexing all megacell signals to pads to create what looks to the chip tester like an Intel 82C596 or an NCR 53C710, depending on the test mode.

This technique provides important verification and test coverage, but complicated the design. Each output pad includes a three-input multiplexer, and each input pad drives signals to three destinations on-chip, significantly increasing the loading. The additional routing complexity required devoting more space for routing channels, and the larger pads reduced placement flexibility.

Conclusions

Integrating multiple I/O functionality onto a single VLSI chip can significantly reduce the cost of the I/O subsystem. However, many system dependent factors and each candidate functionality need to be examined carefully in the system context before deciding to integrate. Some important system considerations are software compatibility, the cost of discrete alternatives, the cost of printed circuit board area, customer connect rate, available IC fabrication capacity, available engineering development resource, and so on. The LASI chip definition is the result of a detailed investigation into optimizing an I/O system for HP's low-end workstations.

Acknowledgments

Many people who made significant contributions to the development of LASI did not have the opportunity to contribute directly to this paper. Some of those individuals include Tom Baker, Greg Burroughs, Kin Chan, Larry Chesler, Marc Clarke, Keith Erskine, Mercedes Gil, Jeff Hargis, Rob Horning, Peter Meier, Eileen Murray, Cheryl Ranson, Charlie Spillman, and Chuck Summers.

References

1. Monish S. Shah, "Overview of A-law and μ-law Data Formats," *Hewlett-Packard Journal,* Vol. 45, no. 2, April 1994, p. 65.
2. Gary P. Rose, et al, "Development of a Multimedia Product for HP Workstations," *Hewlett-Packard Journal,* Vol. 45, no. 2, April 1994, p. 9.

An Integrated Graphics Accelerator for a Low-Cost Multimedia Workstation

Designing with a system focus and extracting as much performance and functionality as possible from available technology results in a highly integrated graphics chip that consumes very little board area and power and is 50% faster and five times less expensive than its predecessor.

by Paul Martin

The graphics subsystem of the Model 712 workstation is a high-performance, low-cost solution that sits directly on the system bus of the Model 712 and consists of the graphics chip, a video RAM-based frame buffer, and a few support chips (see Fig. 1). The project goals closely reflect those of the overall HP 9000 Model 712 program. In priority order these goals were:

- Very low manufacturing cost
- Leadership graphics performance at entry cost levels
- Architectural compatibility
- Compelling new functionality.

Achieving these goals required a major step in the evolution of HP entry-level graphics workstation hardware. Two philosophies helped the team responsible for the graphics chip achieve these goals. The first guiding philosophy was to design with a system-level focus. We examined all required functionality to decide whether it was best to implement it in the graphics subsystem, the host processor, or some combination of the two.

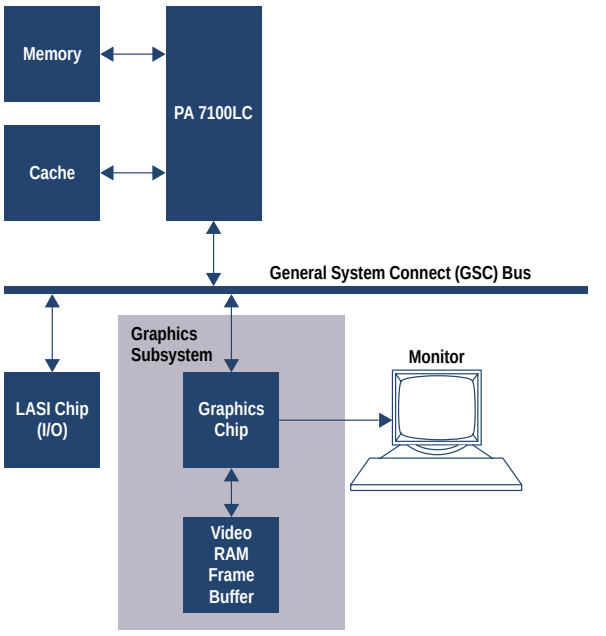


Fig. 1. A block diagram of the essential components that make up the HP 9000 Model 712 workstation.

The second philosophy was to extract as much performance and functionality as possible from readily available technology. We avoided leading-edge technology because of the cost implications. We did make an attempt to use all the features and performance available in mature technologies such as video RAMs (VRAMs) and HP's CMOS26B IC process.

This article describes the features and functionality of the HP 9000 Model 712 graphics subsystem. The considerations that went into accomplishing the goals mentioned above are also described.

Architectural Compatibility

The CRX window accelerator card† introduced by HP in 1991 marked the beginning of a standardized graphics hardware architecture for window system acceleration.[1] This architecture was chosen for its simplicity of implementation and for the clean model it presents to the software driver developers. One of our fundamental design decisions was to accelerate key primitives only—a RISC approach. Many earlier controllers chose to accelerate a large gamut of graphical operations such as ellipses, arithmetic pixel operations, and so on. Graphics subsystems designed with these controllers were typically expensive and exhibited only moderate window system performance. In the CRX and subsequent accelerators, including the Model 712's graphics chip, we decided to accelerate a carefully chosen smaller set of primitives, which are described in the following sections.

Block Transfer. Writing pixels from system memory to the frame buffer or reading from the frame buffer to system memory is a block transfer (see Fig. 2). Writes are used to transfer image data to the frame buffer. Reads are used primarily to save portions of the screen temporarily obscured by pop-up menus (see Fig. 2b).

† A window accelerator is the hardware that provides the images seen on the workstation monitor. In particular, an accelerator is geared toward speeding up environments such as the X Window System. The window accelerator enables the fast movement of windows on the screen, scrolling of text, painting of window borders and backgrounds, and so on.

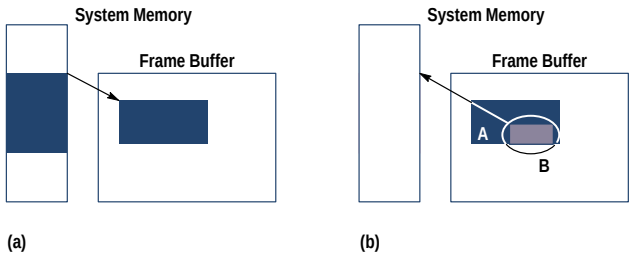


Fig. 2. (a) Block transfer write. (b) Block transfer read. Window B obscures window A. The obscured area is stored in system memory for restoration when the area of window A is exposed.

Block Move. A block move involves transferring pixels from one rectangular area in the frame buffer to another (possibly overlapping) area in the frame buffer (Fig. 3). This is very useful for moving windows on the screen and scrolling lines of text within a window. The block move in the graphics chip supports Boolean operations on the data being moved, such as highlighting text by complementing colors.

Vectors. The ability to draw vectors (line segments) very quickly is a requirement of design applications such as schematic capture and mechanical design (Fig. 4). Thus, the graphics chip has a high-performance vector generator that creates X Window System-compliant line segments.

Fast Text. Characters are accelerated by the graphics chip because of their pervasive use in window systems and the large potential for performance improvement over software-only solutions. A character is defined as a rectangular array of pixels that contains only two colors called foreground and background colors. Because there are only two choices, a single bit is sufficient to specify the color of each pixel in a character. This improves performance by reducing the amount of data that is transmitted from the processor to the graphics chip. For example, the h_p character in Fig. 5 requires only 8 bytes of data versus 48 bytes if this optimization had not been made.

Rectangular Area Fill. This primitive is widely used by window systems to generate window borders, menu buttons, and so on (Fig. 6). It is also important for applications such as printed circuit board layout and IC physical design. Rectangular areas can be patterned using two colors or contain only a single color. Hardware acceleration again gives a large speedup over software-only solutions.

Cursor. Until the late 1980s when hardware cursors started appearing in video ICs, screen cursors were typically generated using software routines. Hardware support is a good trade-off because the circuitry is relatively simple, and a system without hardware acceleration can spend a significant portion of its time updating the cursor. A 64-by-64-pixel, two-color cursor is supported directly in the graphics chip.

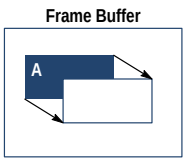


Fig. 3. Block move. Rectangular area A is moved to a new, possibly overlapping location.

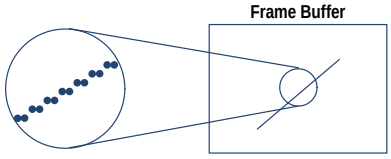


Fig. 4. Vector primitive. A vector is drawn by turning on successive pixels using the Bresenham algorithm.

More complex functionality such as wide lines, circles and ellipses, and 3D primitives are not accelerated directly by the graphics chip because the application performance improvement was determined to be too low for the cost of implementation. These functions can be efficiently implemented in software. This is an example of the system-level design trade-offs mentioned above.

An important aspect of this standardized architecture is software leverage. It is estimated that several software engineering years were saved on the graphics chip because the architecture is virtually identical to that of the CRX graphics subsystem. The savings in software engineering time was applied to tuning and adding new functionality instead of rewriting drivers.

Graphics Chip Operation

To get a better understanding of the operation of the graphics chip let's follow a graphics primitive through the block diagram shown in Fig. 7. A vector is a good example because it involves all of the blocks in the chip. Assume we have a vector that starts at x,y coordinates 0,0, is 8 pixels long, and has a slope of 1/2.

First, several parameters are calculated to set up the vector in the graphics chip. This is done by graphics software (e.g., the X Window System) running on the PA 7100LC CPU. The high-level specification of a vector is:

- Starting x,y coordinate
- Ending x,y coordinate.

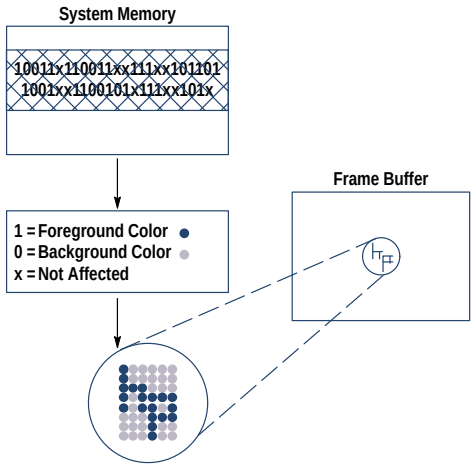


Fig. 5. Fast text primitive. A character is a rectangular array containing two colors, foreground and background colors. Only a single bit is needed to specify each color.

This data is transferred across the GSC bus, through the GSC interface, and into a set of registers in the macro function unit. If these registers are already in use by the macro function unit the data is placed in a 32-word-deep FIFO buffer that the unit can access when it becomes free. This increases efficiency by allowing overlap between the software and hardware processes. The macro function unit's basic job is to break down the high-level descriptions of graphics primitives such as vectors, text, and rectangles into a series of individual requests to draw pixels.

Drawing the vector is automatically triggered when the last of the parameters described in the specification is written into the macro function unit. The macro function then steps its way along the vector using the Bresenham algorithm[2] and issues requests to draw pixels. Since the slope of our vector is 1/2, the y-coordinate is incremented after every two steps along the x-axis as indicated in Fig. 8.

One might expect that a separate x- and y-address would be specified for each pixel to be written. However, with vectors there is excellent coherence between successive x- and y-addresses as pixels are drawn sequentially along the vector. Thus, there are special bus cycles between the macro function unit and the data formatter that specify that the previous x- or y coordinate should be incremented or decremented to generate the new coordinate. This saves sending a full x,y coordinate pair for each pixel drawn and significantly improves bandwidth use on the bus. This optimization is also useful for other primitives such as text and rectangles.

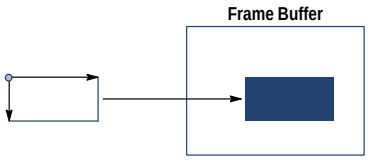


Fig. 6. Rectangular area fill primitive. A rectangle is defined by corner, width, and height. Color or pattern may be applied.

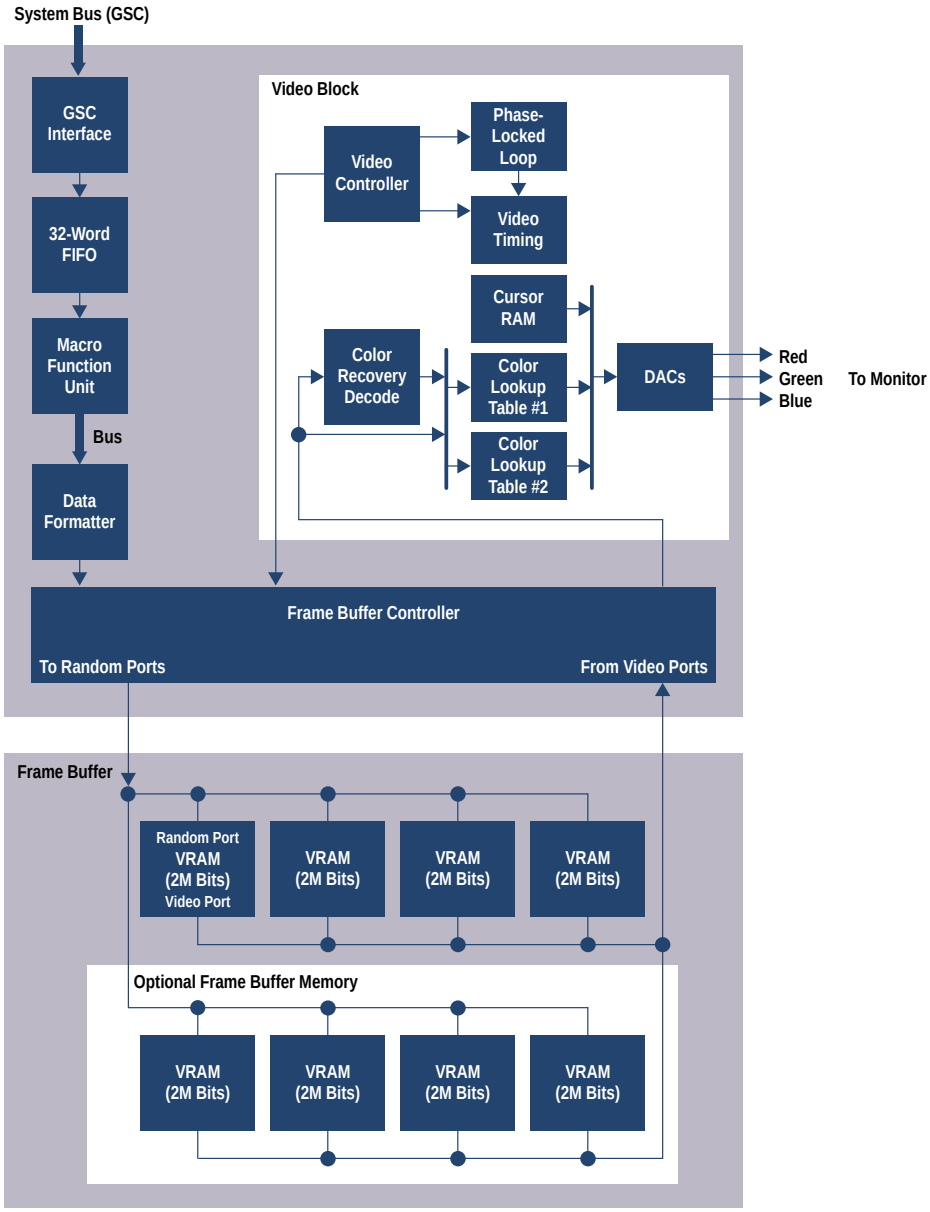


Fig. 7. A block diagram of the components inside the graphics chip.

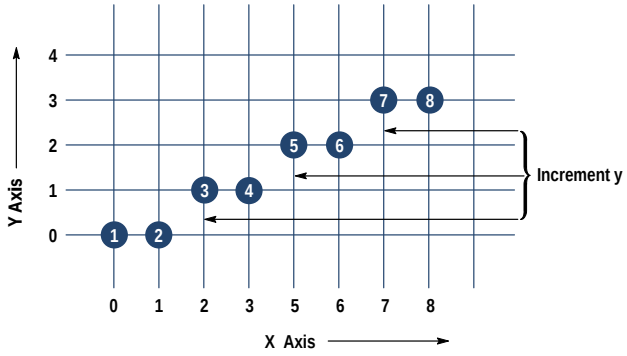


Fig. 8. Pixel representation of a vector that starts at coordinate 0,0, is 8 pixels long, and has a slope of 1/2.

The data formatter's job is to take requests and data from the macro function unit and format them in a way that is best for the frame buffer. In the case of our vector, the pixel addresses received by the data formatter are coalesced into rectangular tiles that are optimized for the frame buffer. The data formatter also recognizes when special VRAM modes may be enabled to improve performance, based on the sequence of data it receives from the macro function unit. For example, page mode (which is described in more detail later in this article) would be enabled during a vector draw. The data formatter also stores the current pixel address, vector color, and a host of other parameters for other primitives.

The frame buffer controller generates signals for the VRAMs based on the requests from the data formatter. The controller looks at the sequence of writes and reads requested and adjusts the timing on the VRAM signals to maximize performance. For our vector, we only need to do simple writes into the frame buffer, and cycles can be as fast as 37.5 ns per pixel. More complex primitives might require data to be read, modified, and written back, possibly to a different frame buffer location.

The graphics chip supports an 8-bit-per-pixel frame buffer. This means that, using normal techniques, only 256 colors can be displayed simultaneously. This is not always adequate for today's graphics-oriented systems. Two methods can be employed to increase the perceived number of colors. The first is dithering, in which an interleaved pattern of two available colors is used to visually approximate a requested color that is not directly available. The second approach is color recovery. Color recovery is visually superior to dithering and is described later.

The Model 712's entry-level configuration frame buffer uses four 2M-bit VRAM parts which allows screen resolutions of up to 1024 by 768 pixels. Adding four more VRAM chips on a daughter card enables screen resolutions up to 1280 by 1024 pixels.

In addition to the screen image data, data for the cursor, color lookup table, and attributes are stored in offscreen frame buffer memory. This is an area in the video RAM frame buffer that is never directly displayed on the CRT. Data in this region is accessed in exactly the same fashion as the screen image data, presenting a consistent interface to software driver writers.

At this point our vector exists in the frame buffer, but cannot be seen by the user. The video block is responsible for getting the screen image data from the frame buffer and converting it for display on the monitor. This display process is asynchronous to the rendering process which placed our vector in the frame buffer.

To get the data in the frame buffer to the monitor, the video controller first sends a request to the frame buffer controller to access the frame buffer data. This data is requested in sequential or scan-line order to match the path of the beam on the monitor. Next, the data from the frame buffer is run through a color lookup table to translate the 8-bit values into 8 bits each of red, green, and blue. The graphics chip supports two independent color lookup tables which are selected on a per-displayed-pixel basis by the attribute data. This feature helps eliminate color contention between applications sharing the frame buffer. Finally, cursor data is merged in by the video block and the digital video stream is converted to analog signals for the monitor.

This completes an overview of the life of a vector primitive, from a high-level description in the software driver to display on the monitor. This basic data flow is the same for other primitives such as rectangles and text.

Low Manufacturing Cost

Low cost was the primary objective for the graphics chip design. As a measure of our success, the manufacturing cost for the Model 712 graphics subsystem is 1/3 the cost of the original CRX graphics subsystem. In addition, the entry-level 1024-by-768-pixel version of the graphics chip costs five times less than the CRX subsystem.

These cost reductions were achieved primarily through an aggressive amount of integration, which is summarized in Fig. 9. The graphics chip represents the culmination of a series of optimizations of the CRX family, combining almost the entire GUI (graphical user interface) accelerator onto a single chip. The only major function not currently integrated is the frame buffer. Frame buffer integration is not feasible today because RAM and logic densities are not quite high enough and there is currently a cost advantage to using commodity VRAM parts.

Since the introduction of the CRX subsystem, industry trends such as denser and cheaper memory and inexpensive IC gates have contributed to cost reductions in graphics hardware. However, the graphics chip's high level of integration also contributes cost reductions in the following areas:

- Elimination of value-priced parts. The color lookup table and the digital-to-analog converter (DAC) have traditionally been an expensive component of the graphics subsystem. This is especially true for systems capable of high resolution (1280 by 1024 pixels, 135 MHz) and having multiple color lookup tables, such as the one built into the graphics chip. The digital phase-locked loop in the graphics chip replaces another expensive external part.
- The density of FETs achieved with the graphics chip, over 4500/mm^2, is significantly higher than with previous generations. This is important because silicon area is a major contributor to overall design cost.
- IC packaging and testing contribute significantly to the cost of each chip in a system. Reducing the number of chips eliminates this overhead. The graphics chip has a full internal scan path and many internal signature registers to reduce test time and chip cost significantly.

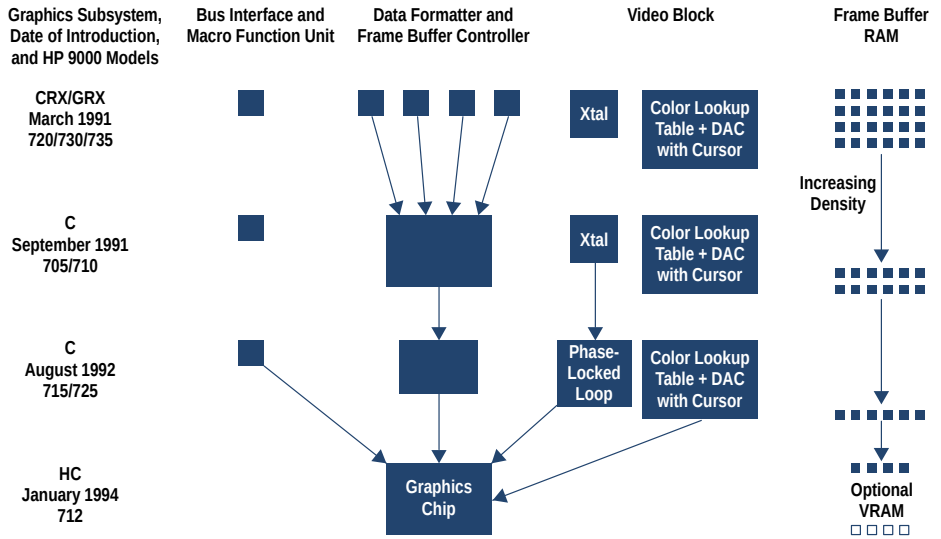


Fig. 9. The evolution of HP's graphical user interface (GUI) accelerator.

- Printed circuit board area is a significant system cost. The elimination of a large number of chips not only reduced the printed circuit board area from about 60 in^2 for the CRX to 14 in^2 for the graphics subsystem in the Model 712, but allowed the graphics to be integrated directly onto the motherboard, eliminating connectors, a bulkhead, and other mechanical components.
- Power consumption for the graphics subsystem in the Model 712 is only six watts. This low power consumption reduces power supply capacity and cooling requirements and therefore cost.
- Manufacturing costs associated with parts placement, test, and rework are proportional to the number of discrete components in a system. The graphics chip and and other chips in the Model 712 include JTAG (IEEE 1149.1) capability and signature generators to reduce the cost of printed circuit board test.

Several factors made this high level of integration practical. First, improved VLSI capabilities such as increased FET density, decreasing wafer costs and the availability within HP of video DAC technology. Secondly, the desktop availability of design and simulation tools capable of handling a model of over 300,000 gates and 500,000 transistors. VLSI design and verification were accomplished on HP 9000 Series 700 workstations using Verilog, Synopsys, and many in-house IC development tools. The performance of the workstations allowed the gate-level simulation of entire video frames (1/60 s of operation) of over 1.2 million pixels, which was the first time this was accomplished within HP.

Performance

The integration described above has also resulted in significant performance benefits. The two major reasons for the performance benefits are wider buses and increased clock rates.

Wider buses are possible between blocks when they are on the same piece of silicon. Wider buses allow better communication bandwidth at a given clock rate, with very little cost impact. A good example on the graphics chip is the much improved communication between the macro function unit and the data formatter which once existed as separate chips.

Increased clock rates are possible because of the elimination of chip-to-chip synchronization delays, pad delays, and printed circuit board trace delays. This compounds the bandwidth benefit of wider buses. HP's CMOS26B technology allows the bus interface, macro function unit, and frame buffer controller blocks of the graphics chip to operate at 80 MHz while the three DACs and two color lookup tables of the video block operate at 135 Mhz.

Intelligent system-level design also made major contributions to performance. A simple example is the block transfer commands which are responsible for transferring data from system memory to the graphics chip and its frame buffer. A special mode was introduced to the memory and I/O controller in the PA 7100LC which allows fast sequential double-word transfers without incurring the overhead of two single-word transfers. This simple change boosted block transfer performance by 50%.

Besides designing with a system-level focus, the other driving philosophy was to extract as much performance and utility as possible from available technology. A good example of this is the use of the advanced features available in the latest 2M-bit and 4M-bit VRAMs. HP has been instrumental in proposing and driving many of these enhancements within the JEDEC committee over the last few years. The more important features include:

- Page mode. This feature eliminates the need to send redundant portions of the pixel address when writing into the frame buffer. The result is that many operations can write a pixel in as little as 37.5 ns versus the more typical 70 ns (see Fig. 10). The key here is that these operations must occur within a page of VRAM or a significant penalty is incurred. By default this page is long and narrow, which is good for block move and block transfer operations but bad for randomly oriented vectors and rectangles. To achieve a better performance balance, we made use of the next feature.
- Stop register/split transfer. This feature allows the frame buffer to be organized in pages that are more square than long and narrow. Moving to this organization improves random vector and small rectangle performance significantly while only slightly reducing large horizontal primitive performance (see Fig. 11).

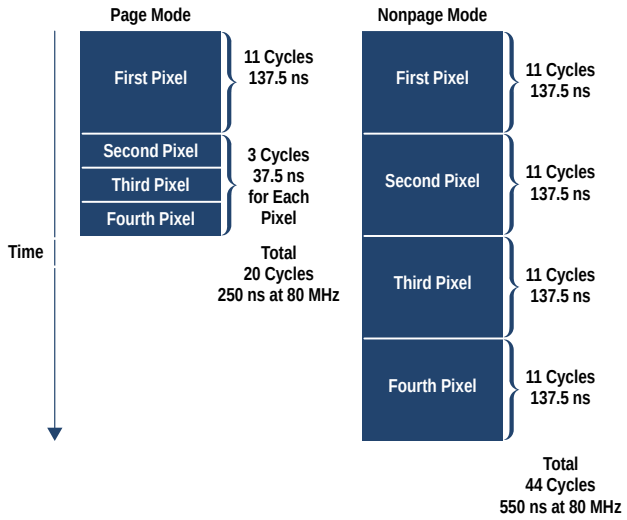


Fig. 10. An illustration of the performance improvement possible using the page mode to write pixels into the frame buffer. This example compares the performance of each mode when just four pixels are transferred to the frame buffer.

- Block write. As mentioned earlier, operations such as text and rectangular fill frequently require only one or two colors to be selected on a per-pixel basis. For this reason VRAMs provide a mode (via a single bit) in which a pixel's color can be selected from an 8-bit foreground or background color stored in the VRAMs. This translates into an 8× performance improvement for these types of operations.

The graphics chip's performance is summarized in Table I. The table compares the performance of the graphics chip at its theoretical hardware limit to its performance in 80-MHz and 60-MHz Model 712 workstations and the Model 720 CRX. The final row in Table I, Xmark, is an industry-standard metric that is an average of several hundred X Window System tests.

Note that the graphics chip's hardware limit is significantly higher than the Model 712 system performance limits. This headroom means that future systems with higher levels of CPU performance or even more highly tuned software drivers will be capable of even better window system performance.

Table I
Summary of the Graphics Chip's Performance

Benchmark	Hardware Limit	Model 712/80	Model 712/60	CRX 720
Block transfer 8-bit pixels/s (frame buffer to system memory)	96 M	60 M	52 M	42 M
Block transfer 8-bit pixels/s (system memory to frame buffer)	20 M	9M	8 M	2 M
Block move pixels/s (frame buffer to frame buffer, 500 by 500 pixels)	47 M	40 M	31 M	40 M
Vectors/s (10-pixel, X compliant)	2.1 M	1.4 M	1.1 M	1.1 M
Text characters/s (6 by 13 pixels/character)	1.0 M	681 k	385 k	295 k
Rectangles/s (10 by 10 pixels/rectangle)	1.7 M	790 k	588 k	270 k
Xmark	—	7.9	6.0	5.6

Compelling Functionality

Beyond improving performance and dropping cost substantially it was an important goal to include useful new functionality in the graphics chip. Below are some of the more important additions.

Software Video Support. One of the design goals for the Model 712 was to be able to play MPEG and H.261 video sequences without expensive hardware acceleration. Through careful analysis of the decoding process it became clear that this was possible at full frame rates and high visual quality using a combination of the following algorithmic, PA 7100LC, and graphics enhancements:

- Rewriting the standard decode algorithms to make them as efficient as possible
- Adding key instructions to the PA 7100LC
- Implementing YUV-to-RGB color space conversion in the graphics chip.

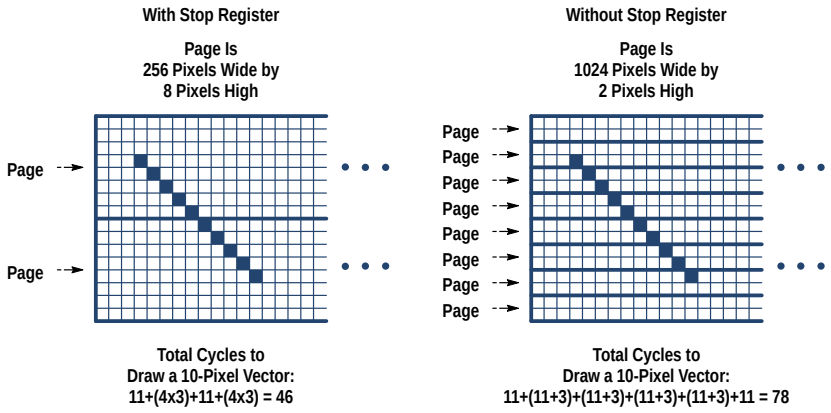


Fig. 11. Improving performance with frame buffer pages that are more square than long and narrow.

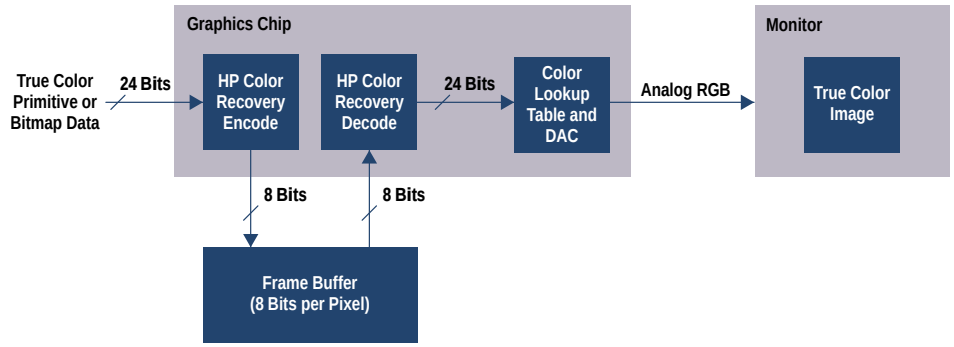


Fig. 12. The HP Color Recovery pipeline.

YUV encoding is used in many video formats. It allocates proportionately more bits to encode the brightness or luminance (Y) of the image, and fewer bits to represent the color (UV) in the image. Since the human eye is more sensitive to brightness than color, this is an efficient scheme. However, since the graphics chip's frame buffer is stored in RGB format, a conversion from YUV to RGB is necessary.

This conversion is a good example of an operation that was relatively expensive in software (a 3-by-3 16-bit matrix multiply) but simple to do in the the graphics chip hardware. This simple addition alone improves video playback performance by as much as 30% and helps enable full 30-frame/s 320-by-288-pixel resolution MPEG playback on a Model 712/80.

HP Color Recovery. The graphics chip incorporates a new display technology called HP Color Recovery. Using a low-cost 8-bit frame buffer and HP Color Recovery, the graphics chip can display images that are in many cases visually indistinguishable from those of a 24-bit frame buffer costing three times more. This feature is useful for the following application areas:

- Visual multimedia (JPEG, MPEG, etc.)
- Shaded mechanical CAD models
- Geographical imaging system
- Document image management
- Visualization
- High-quality business graphics.

A block diagram of the HP Color Recovery pipeline is shown in Fig. 12.

The HP Color Recovery encoding scheme causes no loss of performance for rendering operations and is related to traditional ordered dithering. Dithering is widely used to approximate a large number of colors with an 8-bit frame buffer and is also available in the graphics chip.

The HP Color Recovery decode is much more sophisticated and based on advanced signal processing techniques. This circuitry cycles at 135 MHz and achieves over 9 billion operations per second. HP Color Recovery is described in more detail in the article on page 51.

Multiple Color Lookup Tables. Typically, entry-level workstation and personal computer graphics subsystems have had only a single color lookup table with a limited number of entries, usually 256. In the X Window System this results in the annoying flashing of backgrounds or window contents when a new application is started that takes colors from existing applications. The graphics chip solves this problem in a majority of cases by providing two 256-entry color maps. For most interactions in which the user is focused on a single application and the window manager, this completely eliminates the resource contention and results in a visually stable screen (see Fig. 13).

Software Programmable Resolutions. One of the problems of past workstation graphics subsystems is that they operate at a fixed video resolution and refresh rate. This has posed problems in configuring systems at the factory and during customer upgrades. The graphics chip incorporates an advanced digital frequency synthesizer that generates the clocks necessary for the video subsystem. This synthesizer, based on HP proprietary digital phase-locked loop technology, allows software configurability of the resolution and frequency of the video signal. Thus, alternate monitors can be connected without changing any video hardware. Currently supported configurations include:

- 640 by 480 pixels 60 Hz, standard VESA timing
- 800 by 600 pixels 60 Hz
- 1024 by 1024 pixels 75 Hz and flat panel
- 1280 by 1024 pixels 72 Hz.

As new monitor timings appear, the graphics chip can simply be reprogrammed with the parameters associated with the new monitor.

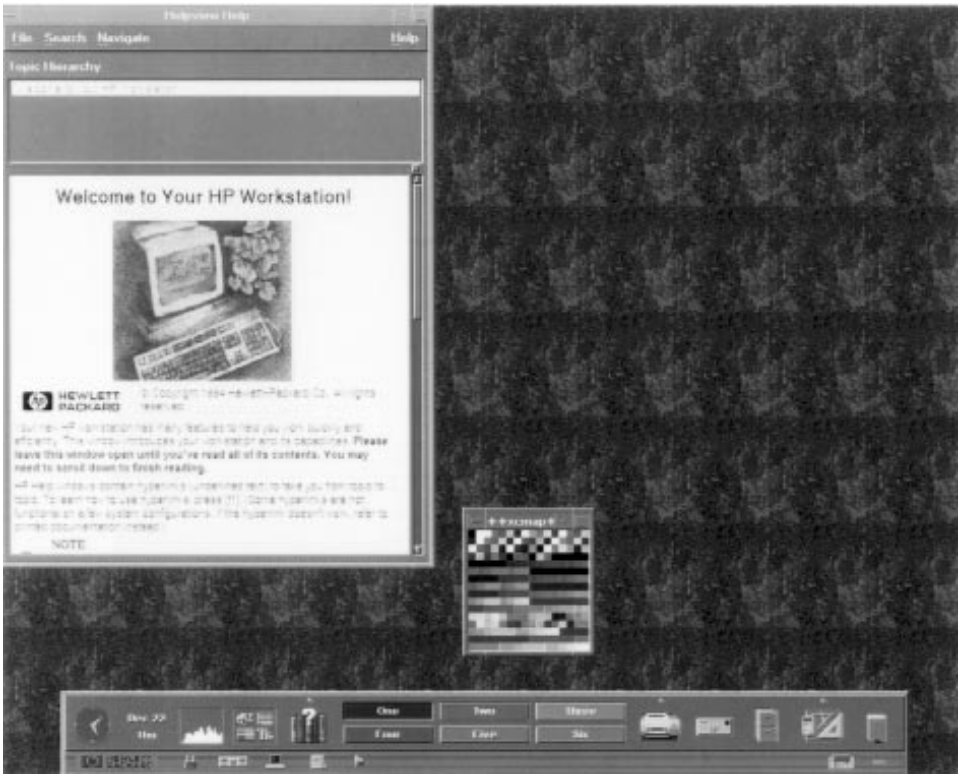
Summary

We created the graphics chip with the philosophies of system-level-optimized design and optimal use of technology. This enabled us to meet our goals of very low manufacturing cost, leadership performance at our cost point, architectural compatibility, and introduction of some important new functionality.

Acknowledgments

This paper briefly summarizes the work of many dedicated and creative members of the graphics chip development team in the graphics hardware and software laboratories in Fort Collins and the Integrated Circuits Business Division. Many thanks to Harry Baeverstad, Tony Barkans, Raj Basudev, Dale Beucler, Rand Briggs, Joel Buck-Gengler, Mike Diehl, Ales Fiala, Randy Fiscus, Dave Maitland, Bob Manley, Dave McAllister, Peter Meier, John Metzner, Brian Miller, Gordon Motley, Donovan Nickel, Cathy Pfister, Larry Thayer, Brad Reak, Cal Selig, James Stewart, and Gayvin Stong for their exceptional efforts.

(a)



(b)

Fig. 13. Comparison between single and multiple color lookup tables. (a) One color lookup table. (b) Two color lookup tables.

References

1. D. Rhoden and C. Wilcox, "Hardware Acceleration for Window Systems," *Proceedings of SIGGRAPH '89, in Computer Graphics,* Vol. 23, no. 9, July 1989, p. 67.

2. J. Bresenham, "Algorithm for Computer Control of a Digital Plotter," *IBM System Journal,* Vol. 4, no. 1, 1965, pp. 25-30.

HP Color Recovery Technology

HP Color Recovery is a technique that brings true color capability to interactive, entry-level graphics devices having only eight color planes.

by Anthony C. Barkans

For many years the only practical way to display high-quality true color images was on a computer with a graphics subsystem providing at least 24 color planes (see the definition of true color on page 52). However, because of the high cost of color graphics devices with 24 planes, many users chose 8-plane systems. Unfortunately, using these 8-plane systems required giving up some color capabilities to save cost.

HP has developed a technique called HP Color Recovery which provides a method for displaying millions of colors within the cost constraints of an 8-plane system. For an example of the image quality provided by HP Color Recovery consider Fig. 1. Fig. 1a shows a close up of a jet plane stored as a full 24-bit-per-pixel true color image. Fig. 1b shows the same jet plane displayed using a traditional 8-bit-per-pixel

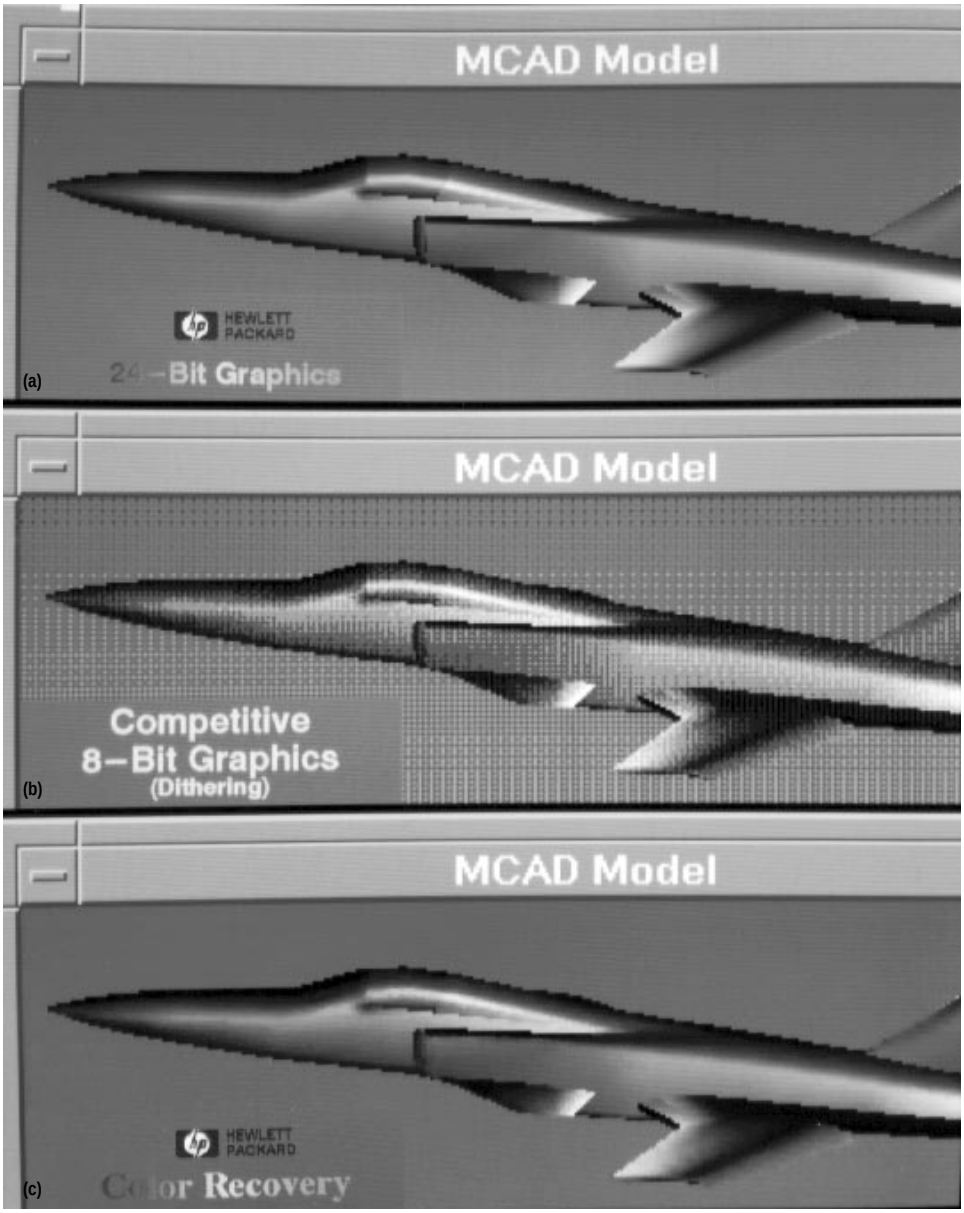


Fig. 1. A true color image and its dithered representations. (a) True color 24-bit image. (b) Typical eight-bit graphics dithered image. (c) An HP Color Recovery dithered image.

True Color

In this paper the term true color is used to define color reproduction such that the underlying digital quantization of the color within an image is not discernable by the human eye. In other words a continuous spectrum of color, such as in a rainbow, can be be displayed so that the color appears to vary continuously across the image. In most computer graphics systems this is accomplished using 24 bits of color information per pixel. With 24 bits, any single pixel can be displayed at one of 2^{24} (16.7 million) colors.

Some graphics systems may define true color to be represented by less than 24 bits per pixel.

system. Finally, Fig. 1c shows how the jet plane will be displayed when using HP Color Recovery in an 8-bit-per-pixel mode on the HCRX-8 graphics device.

Of course, pretty pictures aren't enough. Therefore, one of the primary design goals for HP Color Recovery was to supply the additional color capabilities without giving up interactive performance. Another goal was to be able to work with all types of applications running in a windowed environment such as the X Window System and HP VUE. The implementation of HP Color Recovery used in current HP workstations meets these goals.

Traditional Eight-Plane Systems

Traditional eight-plane systems can display only 256 colors. Two approaches have been employed to get the best results with limited colors. The first is called either *pseudo color* or *indexed color*. This method selects a set of 256 colors and then limits the application to using only that fixed set of colors. For many applications, such as word processing and business graphics, this approach works reasonably well. This is because the resultant images are made up of very few colors. However, when an application needs more than 256 colors, such as realistically shaded MCAD (mechanical computer-aided design) images or human faces in video sequences, then another approach is needed. Since more than 256 colors are required for these applications, a technique to simulate more colors is used. For these applications a technique called *dithering* is employed. The idea of dither is to approximate a single color by displaying two other colors at intermixed pixel locations. For example, a grid of black and white pixels can be displayed to simulate gray. Such a grid of black and white pixels will indeed look gray when viewed from a distance. The primary problem with dithering is that since most people tend to work close to the display, dithered images are viewed as having a grainy or textured appearance (see Fig. 1b).

Color Theory and Dither

Before discussing the details of how HP Color Recovery works, an overview of color theory as it relates to computer generated images and dither should be helpful. This overview describes how the human eye is tricked into seeing color, color precision in graphics, and a dithering method.

Tricking the Human Eye

It is often noted that computer monitors use red, green, and blue (RGB) to produce true color images. A reasonable question to ask is: "Why use these particular colors?" If one examines the spectrum of visible light, it can be seen that red is at the end of the spectrum with the longest wavelengths that the human eye can see while blue is at the other end. Note that green is in about the middle. Also note that white is a mix of all colors. Therefore by mixing varying amounts of red, green, and blue any color can be created. For example, forcing both the red and the green CRT beams to be on at any single location will result in a dot that appears yellow to the human eye.

Thus, one can create the visual appearance of any color by mixing the red, green, and blue components at any pixel location. However, it is interesting to note that the human eye can also perceive a new color when the component colors are mixed spatially. For example, a checkerboard of red and green pixels will be perceived as yellow when viewed from a distance. It is this spatial mixing of color to form a new color that is exploited by dither.

Color Precision

In most systems that deal with true color, color is specified to eight bits for each of the three color components: red, green, and blue. The choice of eight bits is based on two factors. First, the human eye cannot distinguish an infinite number of shades because the dynamic range of the eye is limited. For the most part shaded surfaces rendered with eight bits per color appear smooth with the underlying quantization not readily apparent to the viewer. The second factor that works in favor of using eight bits per color component as a standard is that eight-bit bytes are very convenient to work with in a computer system.

Simple Dithering

When using a 24-bit color system, any displayable color component can be specified using eight bits. For example, consider the red component. When there is no red in a pixel the red component is specified with a binary value of 00000000, which is a decimal 0. A full bright red is specified as a binary number 11111111, which is decimal 255. Of course, high-end display systems, such as the HCRX-48Z, use 24 bits to store and display true color information. The visual quality of these high-end displays is shown in Fig. 1a. However, since low-cost systems typically have a total of only eight bits per pixel to store the color information, an approximation to the true color image is made. The most common method is dither using three bits each for the red and green components. This leaves two bits for blue. Using fewer bits for blue is based on the fact that the human eye has less sensitivity to blue. With fewer bits available per color component, the quantization of the colors becomes apparent to the viewer. The effects of using a limited number of bits for each color can be seen in Fig. 1b.

Dithering approximates any color by using a combination of colors at adjacent pixels. When viewed from a distance the

image appears to be the correct color. However, since dithered systems can store only a limited number of bits in the frame buffer, the primary task of the dithering logic is to select the best set of values to use.

For dithering purposes it is convenient to think of each eight-bit binary component of color as a number in a *three point five* (3.5) representation. This representation means there are three bits on the left side of the binary point and five bits on the right side of the binary point. For example, assume the true color value for red is given as the binary number 01011000. In a 3.5 representation the number becomes 010.11000 binary, which is 2.75 decimal. Since the final dithered values can only be three-bit integers, it can be seen that using only numbers two and three would be desirable. Ideally, the dither would set 3/4 of the pixels to three and 1/4 to two.

If we consider the original color component as being an eight-bit value in a 3.5 format, then the dither values stored in the dither table should be evenly spaced between 000.00000 and 000.11111 (decimal 0.0 to almost decimal 1.0). The output of the table is added to the original eight-bit color component. Once the addition is complete the value is truncated to the desired number of bits for storage in the frame buffer. As a simple example assume that we are continuing to work with a red component that is originally specified as the binary number 01011000. In addition assume that we are using a 2 × 2 dither to reduce the original 8-bit color component to three bits. (The notation 2 × 2 dither means that the dither pattern will repeat in a 2 × 2 grid across the image.) To use a 2 × 2 dither, the least-significant bits of the X and Y window addresses of the pixel are used to index the dither table. The following example shows how a 2 × 2 dither is applied to one pixel of the true color value for red. Table I represents the values in a 2 × 2 dither table.

Table I
A 2 × 2 Dither Table

Indexes		Dither Value	
LSB of Y	LSB of X	Binary	Decimal
0	0	.10100	.625
0	1	.01100	.375
1	0	.00100	.125
1	1	.11100	.875

At the upper left of the window the X and Y addresses are both 0. To dither the data for this pixel location using our color value for red we do the following:

	Binary	Decimal
Input color	010.11000	2.750
Dither value (from Table I)	+ .10100	+ .625
Result	011.01100	3.375
Truncated three-bit value	011	3

Therefore at address 0,0 we would store a 011 binary in the frame buffer for red. Applying the above dither would result in three of the four pixels within every four-pixel block being stored in the frame buffer with a value of 011 (see Fig. 2). The fourth pixel in each block, the one with the LSB of Y set to a 1 and the LSB of X set to a 0, will have a 010 stored in the frame buffer. When a region of this color of red is viewed from a distance the color would appear to be the correct value of 010.11000. If the dithered jet plane shown in Fig. 1b is examined, it can be seen that it is dithered using a method similar to the one described above.

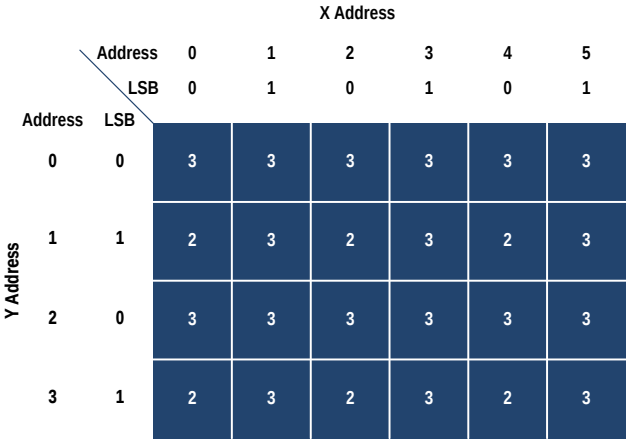


Fig. 2. Results after applying dither. Each box represents a pixel location on the display screen. For example, address (0,0) is defined as the upper left corner of the display. Also note that the numbers stored at each pixel location represent the results of applying the dither values given in Table I to a red component of color originally specified as 01011000 binary (2.75 in our 3.5 notation).

From a distance the colors in the dithered image are integrated by the eye so that they appear correct. However, the fundamental problem with dither is that most dithered images are viewed up close and so the dithering pattern is noticeable in the image.

Dithering Is Key

It is important to realize that to approximate any true color value, a spatial region of the screen is required. This often leads people to say that dithering is a method that trades off spatial resolution for color resolution. However, this is misleading. Some people believe that a single-pixel object cannot be dithered. Actually a single-pixel object can be dithered. The result is that the object will be one of the two dither colors. Going back to the example above, a single-pixel red object specified as binary 01011000 (decimal 2.75) will be stored at any single pixel location as either binary 010 or 011 (decimal two or three). Taken by itself, any single pixel is not a perfect approximation of the true color. However, it is still a reasonable approximation.

The idea of being able to encode each pixel in the image independently by using dither is key to enabling color recovery to work in an interactive environment. As a historical note it should be mentioned that over the last few years several people have developed methods to bring true color capabilities to eight-bit graphics devices. However, these attempts have been based on complex multipixel encoding

schemes. For the most part they have applied data compression techniques to the data stored in the frame buffer. These methods have produced high-quality images, but the encoding is so complex that the user must give up interactive performance to use them. Because of the performance problems these methods have not been widely adopted by the computer graphics community.

HP Color Recovery

The simplest explanation of HP Color Recovery is that it performs the task your eye is asked to do with an ordinary dithered system. In essence, an HP Color Recovery system takes 24-bit true color data generated by an application and dithers it down to eight bits for storage in the frame buffer. Then as the frame buffer data is scanned from the frame buffer to the display, it passes through specialized digital signal processing (DSP) hardware where the work of producing millions of colors is performed. The output of the DSP hardware is sent to the display where millions of colors can be viewed. It is important to recognize that since the data stored in the HP Color Recovery frame buffer is dithered, thousands of applications can work with it. It is also important to recognize that these applications will run at full performance in an interactive windowed environment. In other words, applications do not need to be changed to take advantage of HP Color Recovery.

The Process

HP Color Recovery is a two-part process. First, true color information generated by the application is dithered and then stored in the frame buffer. The type of application generating the true color information is immaterial. For example, true color data can be generated by a CAD application program or as part of a video sequence. The dithering may be done in a software device driver or in the hardware of a graphics controller. It is very important to note that each pixel is treated independently. This pixel independence is key to the ability to work within an interactive windowed environment. The second part of the HP Color Recovery process is to filter the dithered data. The filter is placed between the output of the frame buffer and the DACs that drive the monitor. Fig. 3 shows the HP Color Recovery process starting from when an application generates true color data to when the image appears on the screen. Note that "application" refers to any program that generates true color data for display.

After the application generates the data, it is sent to the device driver. The function of the driver is to isolate the application from hardware dependencies. The driver is supplied by HP. It causes hardware dithering to be used when possible. However, there are times when the driver must perform the dither in software. It is important to note that compared to other dithered systems, there is no performance penalty suffered by an application using HP Color Recovery dither.

The frame buffer stores the image data. Note that in most current systems the output of the dithered frame buffer is sent to the display, resulting in the common patterned appearance in the image. However, with HP Color Recovery, as the frame buffer data is scanned, it is sent through a specialized digital signal processing (DSP) circuit. The DSP is a sophisticated circuit that removes the patterning from the dithered image stored in the frame buffer. This circuit performs over nine billion operations per second. Despite this enormous amount of processing the circuit is surprisingly small. It is this small size that makes HP Color Recovery inexpensive enough to be considered for inclusion in low-end graphics systems.

The Dither Process

In HP Color Recovery the quality of the displayed image depends on the dither used to encode the image. During the development of HP Color Recovery it was found that the size of the dither region determines how well a color can be recovered. It was found that from a region of 2^N pixels the technique can recover about N bits of color per component. Therefore, an eight-bit frame buffer that stores data in 3-3-2 format (3 bits each for red and green and 2 bits for blue) would need a dither region of 32 pixels for each color component to recover 5 additional bits. Thus, using a 32-pixel dither region, an area in the image of uniform color can have the same visual quality as an 8-8-7 image. For example, the sky behind the jet plane in Fig. 1c was recovered to within 1 bit of the original 24-bit true color data shown in the top image.

Most dithers use a 4×4 dither region. Since a 4×4 region covers only 16 pixels, a larger dither region is needed for HP Color Recovery. Therefore, a dither table with 32 entries organized as 2×16 was selected. (The reason for this odd shape is discussed later in this paper.) In addition, most dithers are as simple as the one described earlier in this paper. However, there are cases in which a simple dither does not

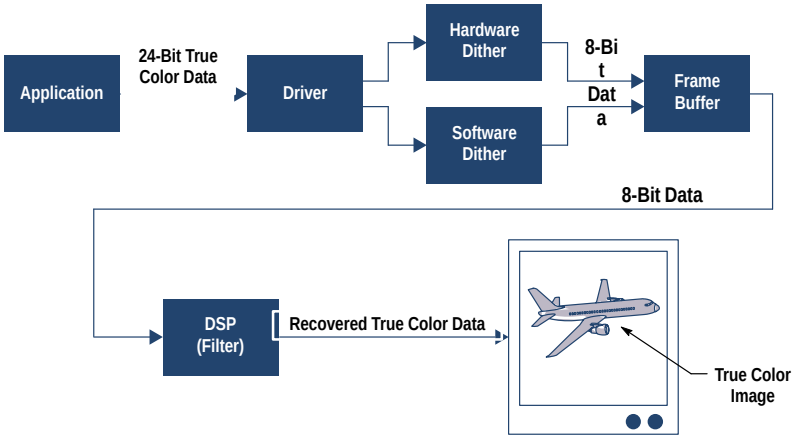


Fig. 3. HP Color Recovery process flow.

work well. Note that in using the simple dither method described above, all true color values from binary 11100000 to 11111111 would dither to 111. For HP Color Recovery the dither table includes both positive and negative numbers. This improves the color range over which the dither is useful.

The HP Color Recovery dither is a little different from most dithers. However, it is on the same order of complexity. It should also be noted that the HP Color Recovery dither is included in the hardware of all HP graphics workstations that support this technique. This means that using the HP Color Recovery dither does not cause a decrease in performance.

The Filter Process

In the example given earlier a red color component represented by the binary value 01011000 (2.75 in decimal) was used to illustrate simple dithering. For this example we used a 2 × 2 dither region in which the end result of the dither was that 3/4 of the pixels stored in the frame buffer were set to 3 (011) and 1/4 of the pixels were set to 2 (010). It is easy to see that if we average the four pixels in the 2 × 2 region we will recover the original color. This can be done as follows:

([value_1 × number_set_to_value_1] + [value_2 × number_set_to_value_2])/total_number_pixels

Using the example data we obtain: ([3 × 3] + [2 × 1])/4 = 2.75.

This averaging works very well in regions of constant color, such as the sky behind the jet plane in Fig. 1. However, there is one fundamental issue that must be addressed for HP Color Recovery to be viable and that is how to handle edges in the image. If edges are not accounted for then the resultant image will blur. The two-dimensional representations of an area of a display screen shown in Fig. 4 are used to illustrate the problem of edge detection and the way the problem is addressed in HP Color Recovery.

As in Fig. 2, each box represents a pixel location on the display screen. In Fig. 4a the numbers represent the original true color data for one of the color components (e.g., red) in a 24-bit per pixel system. Fig. 4b shows the same region after simple dithering has been applied. Fig. 4c shows the pixel values after the application of HP Color Recovery. Fig. 4c pixel values represent the color data that would be displayed on the computer screen.

Region A in each of these figures is an area of constant color, whereas region B encompasses an edge. For illustration purposes, the dither region is again assumed to be 2 × 2 pixels.

The dithered color data shown in Fig. 4b is derived from the original color data shown in Fig. 4a and from using the simple dithering technique described in connection with Table I. The data shown in Fig. 4b is what would be stored in the frame buffer and displayed in a typical dithered system (e.g., Fig. 1b).

When it is time to display Pix_1 the data for the four pixels shown as Region A in Fig. 4b would be sent to the filter. The data stored in the region would be summed and then divided by the number of pixels in the region. The sum of the pixels in Region A is 11 and 11/4 = 2.75. Thus, the output of the filter when evaluating Pix_1 would be 2.75. This output value would be displayed on the computer display at Pix_1's location. Note that the output of the filter is the exact value of the original data at that point in Fig. 4a.

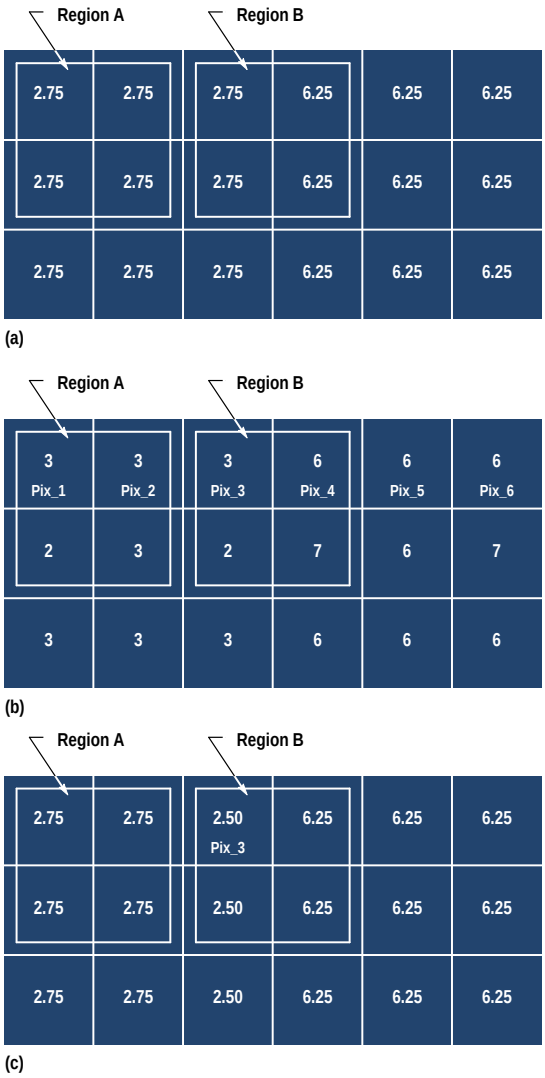


Fig. 4. (a) Pixel values for the original 24–bit per pixel color data. (b) The color data from Fig. 4a after it has been dithered and placed in the frame buffer. This would be the data displayed in a typical dithered system with the result appearing as in Fig. 1b. (c) The pixels from Fig. 4b after applying HP Color Recovery.

The next pixel along the scan line to be evaluated is Pix_2. The filter region for evaluating Pix_2 would include the two rightmost pixels of region A and the two leftmost pixels of region B (see Fig. 4b). Applying the filter operation for Pix_2 again results in the output value matching the value at that location in Fig. 4a (2.75).

If the evaluation is done on Pix_3, the pixels in region B would be summed and then divided by the number of pixels in the region, and the result would be 4.50. This value is very different from the original data value of 2.75 in Fig. 4a. Using the value of 4.50 at Pix_3 would result in edge smearing. To solve this problem a special edge detector that looks for edges in noisy data is used. The idea is to compare each pixel in the filter region with a value that is within ±1 of the pixel being evaluated. Since the data stored at Pix_3 is a 3,

only pixels within region B that have a value of 2, 3, or 4 would pass the edge compare. The values that pass the edge detector are then summed and the total is divided by the number of pixels that pass the edge compare. For Pix_3, only Pix_3 and the pixel below it would pass the edge detector. Summing the two passing values together and dividing by 2 gives a result of 2.50. This value is slightly different from the original value of 2.75, but it is a better estimation to the original than the 4.50 obtained without the edge detection. The displayed values for the entire example region are shown in Fig. 4c.

Software Considerations

Since dithered frame buffers are in common use today, many existing software applications can work with a dithered frame buffer. All of these applications could work with HP Color Recovery.

On products that use the Model 712's graphics chip, which is described in the article on page 43, and HP's Hyperdrive (HCRX), HP Color Recovery is supported. In these products we have chosen to have HP Color Recovery enabled as the default for 3D applications run in an eight-bit visual environment. Thus when using the 3D graphics libraries Starbase, PHIGS, or PEXlib, and opening an application in an eight-bit visual environment with true color mode, HP Color Recovery will normally be enabled. Of course, setting an application to use a pseudo color map will disable HP Color Recovery and give the application the desired pseudo color capability. Because Xlib is tied into the pseudo color model rather than the 3D libraries, Xlib applications leave HP Color Recovery off by default. However, a mechanism is supported that allows HP Color Recovery to be enabled when using Xlib.[1] The biggest change is that Xlib applications must do their own dithering.

Implementation

The implementation of HP Color Recovery was based on the assumption that color recovery would be most useful in entry-level graphics products. Entry-level graphics products are defined as products in which there is storage for only 8 bits per pixel in the frame buffer. These same products that benefit the most from HP Color Recovery are also the ones where product cost must be carefully controlled. Therefore, the implementation effort was driven with a strong sense of cost versus end user benefit.

Dither Table Shape. As mentioned earlier, the dither region shape used with HP Color Recovery is 2×16. The optimum shape would be closer to square, such as 4×8. However, the filter circuit needs storage for the pixels within the region. A 2×16 circuit requires that the current scan line's pixel and the data for the scan line above be available. This means that as data for any scan line enters the circuit, it is used to evaluate pixels on the current scan line. In addition, the data is saved in a scan line buffer so it can be used when evaluating the pixels on the next scan line (see Fig. 5). It should be noted that the storage for a scan line of data uses approximately one half of the circuit area in the current implementation. Therefore, if a 4×8 region had been used, three scan

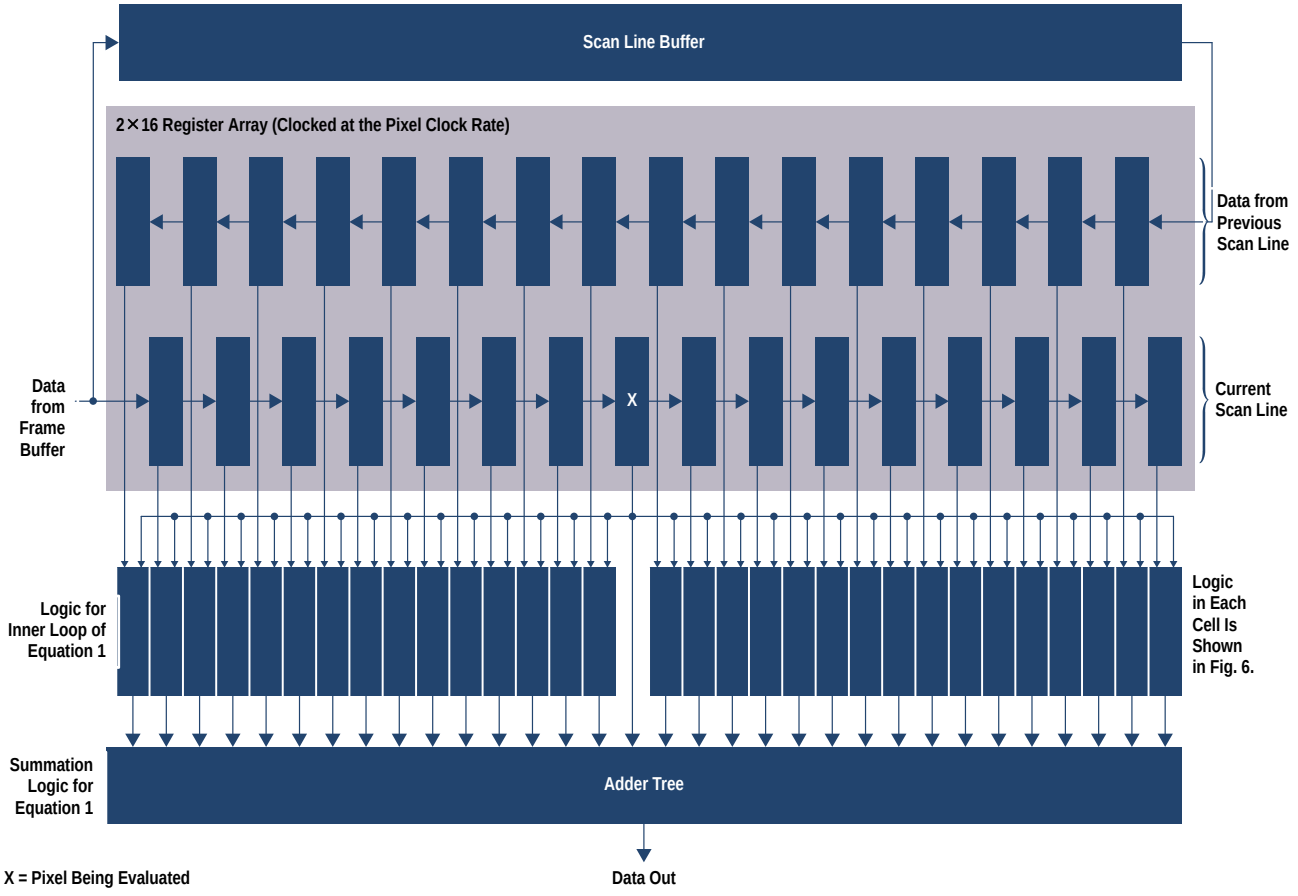


Fig. 5. A block diagram of the HP Color Recovery filter circuit.

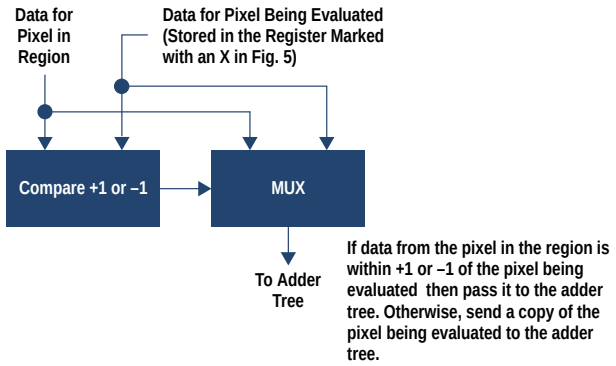


Fig. 6. A simplified representation of the logic circuitry that exists in each of the logic blocks in Fig. 5.

line buffers would have been required, almost doubling the cost of the HP Color Recovery logic.

Filter Function Logic. As explained earlier, the HP Color Recovery filter function averages the data within a region by summing the data for the pixels that pass an edge compare operation. The sum is then divided by the number of pixels that pass the edge compare. Typically, building the logic for a filter function like this is difficult and costly because it requires a divide circuit running at the video clock rate of 135 MHz. The HP Color Recovery filter function is implemented so that this is not a problem.

The implementation details of the filter function are complex. However, if we ignore the high-speed pipeline issues and some minor adjustments required to optimize image quality, we can reduce the implementation of the filter function to the following equation:

$$\sum_{i=0}^{k-1}(\text{Frame\_Buffer\_Data}_i)(W_i) + (\text{Evaluated\_Pixel})(\overline{W}_1) \quad (1)$$

where k is the number of pixels in the filter and W_i is a flag equal to one when a pixel passes an edge compare operation and zero when it doesn't. This flag can be thought of as the output of the comparator shown in Fig. 6.

The idea behind this equation is that if a pixel passes the edge compare, include it in the total. On the other hand, if a pixel fails the edge compare, then substitute the data for the pixel being evaluated for the failing pixel. The overriding assumption is that the pixel being evaluated is a reasonably good guess of the true color data. The worst case is that all the pixels around the sample fail the edge compare and the dithered color is used for that location. Since dithering uses a reasonable sample at each location this extreme case results in a reasonable image being displayed.

To see how this works let's look at two examples. In the first example assume that the pixel being evaluated is a single red dot specified using 01011000 binary (2.75 in our decimal numbering system). This is the same color used in some of the examples described earlier. However, this time let us assume that it is dithered to a value of 011. Also assume that this pixel is surrounded by green. Since the edge compare is done on a per-color basis, all the pixels in the region except the pixel being evaluated will fail the edge compare. In this case we will add a red value of 011 thirty-two times. The result out of the adder tree in Fig. 5 will have a red value of 01100000 (3.00 in decimal). Although this is not exact it will appear as a red dot in the middle of a green region. In other words, a reasonable approximation.

In the second example assume a region that is filled with red is specified with the same eight-bit binary value of 01011000. Also assume the simple dither method described earlier is used. In this case 3/4 of the pixels will be stored as 011. The other 1/4 will be stored as 010. Since none of the pixels fails the edge compare we will send twenty-four pixels with the value of 011 and eight with the value 010 to the adder tree. The results of the adder will be a binary value of 01011000 (2.75 decimal). In this case the output of HP Color Recovery will match the input true color data exactly.

Hardware details. The filtering logic, which was shown in a systems context in Fig. 3, is expanded in Fig. 5. As the frame buffer is scanned, each pixel in the display is sequentially sent to the logic shown in Fig. 5. The left side of the figure shows the path taken as the data for each pixel read from the frame buffer enters the filtering logic. The data is sent both to a pipeline register for immediate use, and to a scan line buffer for use when the next scan line is being evaluated. The 32 registers shown in Fig. 5 store the data for the 2 × 16 region being evaluated. These registers are clocked at the pixel clock rate. Note that the data for each pixel on the display will pass through the location marked with the X. When a pixel is at the location X, it is called the pixel being evaluated. This means that the results of applying equation 1 are assigned to the display at the screen address of X.

The 32 pixels stored in the pipeline registers shown in Fig. 5 are sent through blocks of logic that perform the inner loop evaluation of equation 1. This inner loop is essentially an edge detector. The logic shown in Fig. 6 allows only pixels that have similar numeric values to the pixel being evaluated to be included in the summation. The summation logic is simply an adder tree that sums the results of the pixels passing the edge compare. The filter function is performed in parallel for all the pixels within the filter region.

Given the complexity of the function being performed in the filter circuit, the circuit is surprisingly small. The entire filter circuit is made up of approximately 35,000 transistors. Compared to the number of transistors required to increase the number of color planes, this is very small. For example increasing the number of color planes from 8 to 16 on a typical SVGA (Super VGA) system (1024 × 768-pixel resolution) requires over 8,000,000 transistors, which is 1M bytes of additional frame buffer memory. Because of the small size of the HP Color Recovery circuit, it is inexpensive enough to be included in entry-level graphics systems.

Questions and Answers

Thus far the concepts behind HP Color Recovery have been discussed. It has been shown that HP Color Recovery can supply additional color capabilities to low-end graphics systems while maintaining an interactive windowed environment. The following are answers to the most frequently asked questions about the practical use of HP Color Recovery.

- Question: Is there a difference between a 24-bit true color image and one displayed using HP Color Recovery?

Answer: Yes. If you view a 24-bit image and an HP Color Recovery image side by side there are differences. For example, the back edge of the wing in Fig. 1c has some artifacts in it. At normal size the artifacts can be found but are less noticeable than in Fig. 1c.

- Question: How many colors are reproducible with HP Color Recovery?

  Answer: In the best case HP Color Recovery can provide up to 23 bits of accuracy. However, in typical images about four million colors can be reproduced.

- Question: Are artifacts introduced by HP Color Recovery?

  Answer: In areas of very low contrast, artifacts will show up. Again the back edge of the wing in Fig. 1c is a good example.

- Question: Does HP Color Recovery look the same on all HP products that support it?

  Answer: No. The first implementation was designed for the graphics chip used in the HP 9000 Model 712 workstation. After that design was finished some improvements were made which ended up in the HCRX family of graphics devices. These changes are hidden deep in the details of the implementation, enabling any application using HP Color Recovery on one product to work without change on the other products.

- Question: Do applications need to change to use HP Color Recovery?

  Answer: If the application was written using a 3D application program interface the answer is no. Of course it must be running in an eight-bit visual environment on a device that supports HP Color Recovery. In addition, the application must have been written to use the 24-bit true color model. However, if the application was written using Xlib then it must be changed to do the dithering. Details can be found in reference 1.

- Question: Is there a way to turn HP Color Recovery off?

  Answer: Yes. Set the environment variable `HP_DISABLE_COLOR_RECOVERY` to any value.

- Question: What happens to the color map in the HP 9000 Model 712's graphics chip when HP Color Recovery is enabled?

  Answer: In the graphics chip there are two hardware color maps. By default, the X11 server permanently downloads the default color map into one of these hardware color maps. If HP Color Recovery is enabled the remaining color map is used by HP Color Recovery. See the article on page 43 for more information about these color maps.

- Question: What happens to the color map on HCRX graphics when HP Color Recovery is enabled?

  Answer: On HCRX graphics devices there are two hardware color maps in the overlay planes and two in the image planes. By default, the X11 server permanently downloads the default color map into one of the overlay planes' hardware color maps. This is true in each of the following configurations:
  - The HCRX-8 and HCRX-8Z frame buffer configurations with no transparency have one hardware color map in the overlay planes and two in the image planes that are available. In this configuration the HP Color Recovery color map can be downloaded into any of the available hardware color maps.
  - The HCRX-8 and HCRX-8Z frame buffer configurations with transparency have only one hardware color map in the overlay planes and only one in the image planes. Since the hardware color map for the overlay planes already has the default color map loaded into it, there is only one color map available for HP Color Recovery to choose from. Therefore, in this configuration the HP Color Recovery color map is downloaded into the remaining hardware color map.
  - The HCRX-24 and HCRX-24Z frame buffer configurations with or without transparency have one hardware color map in the overlay planes and two in the image planes that are available. In this configuration, when using an eight-bit visual depth the HP Color Recovery color map can be downloaded into any of the available hardware color maps.

- Question: Does HP Color Recovery work with logical raster operations?

  Answer: Yes. Like any dithered frame buffer system, HP Color Recovery works with raster operations such as `AND`, `OR`, and `XOR`.

- Question: How do image processing applications interact with HP Color Recovery?

  Answer: There are two basic classes of image processing applications: feature finding and image enhancement.
  - Feature finding. Most feature-finding applications are based on edge detection. The results of running one of these types of applications can be displayed using HP Color Recovery. However, as with other dithered frame buffers, any application using the frame buffer as the image source may have problems if it does not account for the dither.
  - Image enhancement. Image enhancement applications are typically used to enhance images for the human visual system. The goal of many of these applications is to bring out low-level features of the image. It is possible to preprocess the image and send it to HP Color Recovery. However, if there is a need for an extremely high-quality image (e.g., medical imaging) a 24-bit frame buffer may be necessary.

- Question: If an image is dithered using a dither method other than the one developed for HP Color Recovery, can it be displayed on a system that supports HP Color Recovery?

  Answer: Yes. One option is to turn HP Color Recovery off. However, the image can be processed with HP Color Recovery on. In this case the image will be viewable. The image quality will be comparable to viewing the image on a typical dithered system, but the dithering artifacts will be replaced with a new set of artifacts.

- Question: Can an image created using the HP Color Recovery dither method be viewed on an eight-bit system that does not support HP Color Recovery?

  Answer: Yes. However, it is important to realize that without the HP Color Recovery back end the dithering artifacts will be visible in the image.

- Question: Can a user read the frame buffer data?

Answer: Yes. However, as with any dithered system there is the issue of precision. For example, if the red data is generated with eight bits of precision, then the readback will give a three-bit dithered value for the data. The data on readback is not the same as the eight-bit value generated by the application.

- Question: Does HP Color Recovery work with multimedia applications?

  Answer: Yes. By removing the dithering artifacts, image quality during MPEG (video) playback is improved.

- Question: Does HP Color Recovery impact application performance?

  Answer: No. The HP Color Recovery dither is implemented in fast hardware in both the Model 712's graphics chip and the HCRX graphics subsystem. When hardware dithering cannot be used, such as with virtual memory double buffering, a software dither is performed by the device driver. Since the dither is the same complexity as common dithers, there is no performance penalty for using HP Color Recovery when compared to using other dithered systems.

  In addition, the DSP circuit in the back end is placed in the path of the data being scanned into the monitor. As such the DSP does get in the path (without affecting application performance) when the system is performing what the user sees as interactive tasks.

- Question: Can an image generated using HP Color Recovery be displayed on output devices other than monitors (e.g., printers)?

  Answer: Many applications generate a print file. In this case the data displayed on the monitor is not used to create the print file. Therefore, HP Color Recovery will not interfere with the output. Another method used to generate hardcopy is a screen dump. Unfortunately, a complete solution for dumping a color-recovered image to a printer is not available yet.

Conclusion

Color recovery brings added color capabilities to entry-level systems. Since the technology is based on dither, these additional color capabilities can be brought to an entry-level system while maintaining an interactive environment that supports many current applications.

Acknowledgments

Many people have helped transform HP Color Recovery from an idea into a reality. The list would be too long to print here. Without the list of names I hope everyone involved knows that I appreciate their efforts. However, there are several people that I must list by name. These people are Paul Martin, Larry Thayer, Brian Miller, and Randy Fiscus. Additionally, a special thanks goes to Dave McAllister who took my notes and turned them into real logic. Along the way, Dave found many innovations that led to a better design.

Reference

1. *HP Color Recovery Technology*, HP publication Number 5962-9835E.

Real-Time Software MPEG Video Decoder on Multimedia-Enhanced PA 7100LC Processors

With a combination of software and hardware optimizations, including the availability of PA-RISC multimedia instructions, a software video player running on a low-end workstation is able to play MPEG compressed video at 30 frames/s.

by Ruby B. Lee, John P. Beck, Joel Lamb, and Kenneth E. Severson

Traditionally, computers have improved productivity by helping people compute faster and more accurately. Today, computers can further improve productivity by helping people communicate better and more naturally. Towards this end, at Hewlett-Packard we have looked for more natural ways to integrate communication power into our desktop machines, which would allow a user to access distributed information more easily and communicate with other users more readily.

We felt that adding audio, images, and video information would enrich the information media of text and graphics normally available on desktop computers such as workstations and personal computers. However, for such enriched multimedia communications to be useful, it must be fully integrated into the user's normal working environment. Hence, as the technology matured we decided to integrate increasing levels of multimedia support into both the user interface and the basic hardware platform.

In terms of user interface, we integrated a panel of multimedia icons into the HP VUE standard graphical user interface, which comes with all HP workstations. These multimedia icons are part of the HP MPower product.[1] HP MPower enables a workstation user to receive and send faxes, share printers, access and manipulate images, hear and send voice and CD-quality stereo audio, send and receive multimedia email, share an X window or an electronic whiteboard with other distributed users, and capture and play back video sequences. The HP MPower software is based on a client/server model, in which one server can service around 20 clients, which can be workstations or X terminals.

In terms of hardware platforms, we integrated successive levels of multimedia support into the baseline PA-RISC workstations.[2,3,4] First, we integrated support for all the popular image formats such as JPEG (Joint Photographic Experts Group)† compressed images.[5] Then, we added hardware and software support for audio, starting with 8-kHz voice-quality audio, followed by support for numerous audio formats including A-law, μ-law, and 16-bit linear mode, with up to 48-kHz mono and stereo. This allowed high-fidelity, 44.1-kHz stereo, 16-bit CD-quality audio to be recorded, manipulated, and played back on HP workstations. At the same time, we supported uncompressed video capture and playback.

In January 1994, HP introduced HP MPower 2.0 and the entry-level enterprise workstation, the HP 9000 Model 712, which is based on the multimedia-enhanced PA-RISC processor known as the PA 7100LC.[6,7,8] The video player integrated in the MPower 2.0 product is the first product that achieves real-time MPEG-1 (Moving Picture Experts Group)[9] video decompression via software running on a general-purpose processor. Typically real-time MPEG-1 decompression is achieved via special-purpose chips or boards. Previous attempts at software MPEG-1 decompression did not attain real-time rates.[10] The fact that this is achieved by the low-end Model 712 workstation is significant.

In this paper, we discuss the support of MPEG-compressed video as a new (video) data type. In particular, we discuss the technology that enables the video player integrated into the HP MPower 2.0 product to play back MPEG-compressed video at real-time rates of up to 30 frames per second.

Digital Video Standards

We decided to focus on the MPEG digital video format because it is an ISO (International Standards Organization) standard, and it gives the highest video fidelity at a given compression ratio of any of the formats that we evaluated. MPEG also has broad support from the consumer electronics, telecommunications, cable, and computer industries. The high compression capability of MPEG translates into lower storage costs and less bandwidth needed for transmitting video on the network. These characteristics make MPEG an ideal format for addressing the need for detail in the video used in technical workstation markets and computer-based training in commercial workstation markets.

MPEG is one of several algorithmically related standards shown in Fig. 1. All of these digital video compression standards use the discrete cosine transform (DCT) as a fundamental component of the algorithm. Alternatives to discrete cosine-based algorithms that we looked at include vector quantization, fractals, and wavelets. Vector quantization

† JPEG is an international digital image compression standard for continuous-tone (multilevel) still images (grayscale and color).

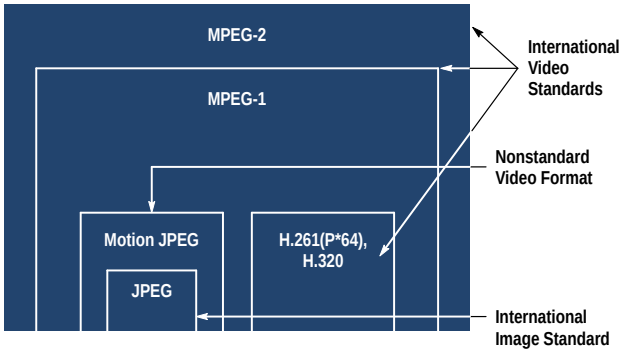


Fig. 1. Digital video standards based on the discrete cosine transform.

algorithms are popular on older computer architectures because they require less computing power to decompress, but this advantage is offset by poorer image quality at low bandwidth (high compression) compared to MPEG for practical vector quantization methods. Algorithms based on wavelet and fractal technology have the potential to deliver video fidelity comparable to MPEG, but there is presently a lack of industry consensus on standardization, a key requirement for our use.

Another advantage of a high-performance implementation of MPEG is the ability to leverage the improvements to the other DCT-based algorithms. Although the relationships shown in Fig. 1 do not represent a true hierarchy of algorithms is useful for illustrating increased complexity as one moves from JPEG to MPEG-2, or from H.261 to MPEG-2.

All of these formats have much in common, such as the use of the DCT for encoding. The visual fidelity of the algorithms was the key selection criterion and not ease of implementation or performance on existing hardware.

Although JPEG supports both lossy and lossless compression, the term JPEG is typically associated with the lossy specification.† The primary goal of JPEG is to achieve high compression of photographic images with little perceived loss of image fidelity. Although it is not an ISO standard, by convention, a sequence of JPEG lossy images to create a digital video sequence is called motion JPEG, or MJPEG.

H.261 is a digital video standard from the telecommunications standards body ITU-TSS (formerly known as CCITT). H.261 is one of a suite of conferencing standards that make up the umbrella H.320 specification. H.261 is often referred to as P*64 (where P is an integer) because it was designed to fit into multiples of 64 kbits/s bandwidth. The first frame (image) of an H.261 sequence is for all practical purposes a highly compressed lossy JPEG image. Subsequent frames are built from image fragments (blocks) that are either JPEG-like or are differences from the image fragments in previous frames. Most video sequences have high frame-to-frame coherence. This is especially true for video conferencing. Because the encoding of the movement of a piece of an image requires less data than an equivalent JPEG fragment, H.261 achieves higher visual fidelity for a given bandwidth than does motion JPEG. Since the encoding of the differences is always based on the previous frames, the technique is called *forward differencing.*

† In lossless compression, decompressed data is identical to the original image data. In lossy compression, decompressed data is a good approximation of the original image data.

The MPEG-1 specification goes even further than H.261 in allowing sophisticated techniques to achieve high fidelity with fewer bits. In addition to forward differencing, MPEG-1 allows backward differencing (which relies on information in a future frame) and averaging of image fragments. (Forward and backward differencing are described in more detail in the next section.) MPEG-1 achieves quality comparable to a professionally reproduced VHS videotape even at a single-speed CD-ROM data rate (1.5 Mbits/s).[9,11] MPEG-1 also specifies encodings for high–fidelity audio synchronized with the video.

MPEG-2 contains additional specifications and is a superset of MPEG-1. The new features in MPEG-2 are targeted at broadcast television requirements, such as support for frame interleaving similar to analog broadcast techniques. With widespread deployment of MPEG-2, the digital revolution for video may be comparable to the digital audio revolution of the last decade.

The approximate bandwidths required to achieve a level of subjective visual fidelity for motion JPEG, H.261, MPEG-1, and MPEG-2 are shown in Fig. 2. Motion JPEG will primarily be used for cases in which accurate frame editing is important such as video editing. H.261 will be used primarily for video conferencing, but it also has potential for use in video mail. MPEG-1 and MPEG-2 will be used for publishing, where fidelity expectations have been set by consumer analog video tapes, computer-based training, games, movies on CD, and video on demand.

MPEG Compression

MPEG has two classes of frames: intracoded and non-intracoded frames (see Fig. 3). Intracoded frames, also called *I-frames,* are compressed by reducing spatial redundancy within the frame itself. I-frames do not depend on comparisons with past or "reference" frames. They use JPEG-type compression for still images.[5]

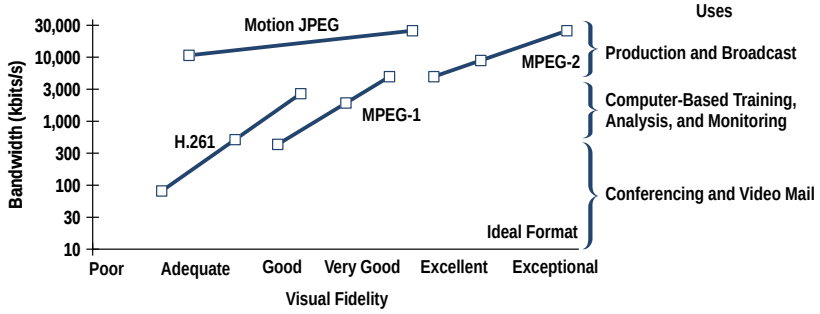


Fig. 2. Compressed video bandwidth versus subjective visual fidelity. The ideal format achieves exceptional visual fidelity at the lowest bandwidth.

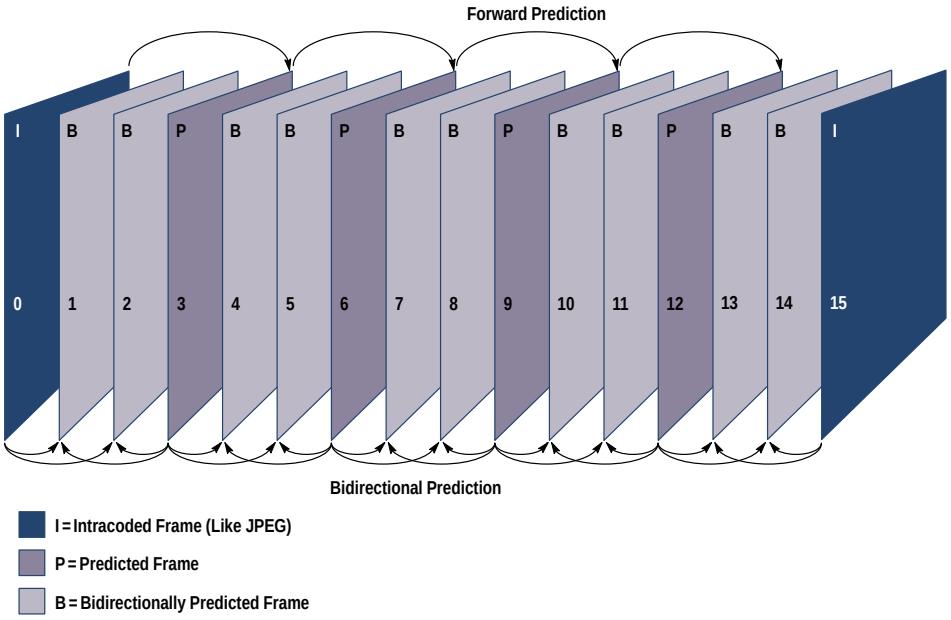


Fig. 3. MPEG frame sequencing.

Nonintracoded frames are further divided into *P-frames* and *B-frames*. P-frames are predicted frames based on comparisons with an earlier reference frame (an intracoded or predicted frame). By considering temporal redundancy in addition to spatial redundancy, P-frames can be encoded with fewer bits. B-frames are bidirectionally predicted frames that require one backward reference frame and one forward reference frame for prediction. A reference frame can be an I-frame or a P-frame, but not a B-frame. By detecting the motion of blocks from both a frame that occurred earlier and a frame that will be played back later in the video sequence, B-frames can be encoded in fewer bits than I- or P-frames.

Each frame is divided into macroblocks of 16 by 16 pixels for the purposes of motion estimation† in MPEG compression and motion compensation in MPEG decompression. A frame with only I-blocks is an I-frame, whereas a P-frame has P-blocks or I-blocks, and a B-frame has B-blocks, P-blocks, or I-blocks. For each P-block in the current frame, the block in the reference frame that matches it best is identified by a motion vector. Then the differences between the pixel values in the matching block in the reference frame and the current block in the current frame are encoded by a discrete cosine transform.

The color space used is the YCbCr color representation rather than the RGB color space, where Y represents the luminance (or brightness) component, and Cb and Cr represent the chrominance (or color) components. Because human perception is more sensitive to luminance than to chrominance, the Cb and Cr components can be subsampled in both the x and y dimensions. This means that there is one Cb value and one Cr value for every four Y values. Hence, a 16-by-16 macroblock contains four 8-by-8 blocks of Y, and only one 8-by-8 block of Cb and one 8-by-8 block of Cr values (see Fig. 4). This is a reduction from the twelve 8-by-8 blocks (four for each of the three color components) if Cb and Cr were not subsampled. The six 8-by-8 blocks in each 16-by-16 macroblock then undergo transform coding.

† Motion estimation uses temporal redundancy to estimate the movement of a block from one frame to the next.

Transform coding concentrates energy in the lower frequencies. The transformed data values are then quantized by dividing by the corresponding quantization coefficient. This results in discarding some of the high-frequency values, or lower-frequency but low-energy values, since these become zeros. Both transform coding and quantization enable further compression by run-length encoding of zero values.

Finally, the nonzero coefficients of an 8-by-8 block used in the discrete cosine transform can be encoded via variable-length entropy encoding such as Huffman coding. Entropy encoding basically removes coding redundancy by assigning the code words with the fewest number of bits to those coefficients that occur most frequently.

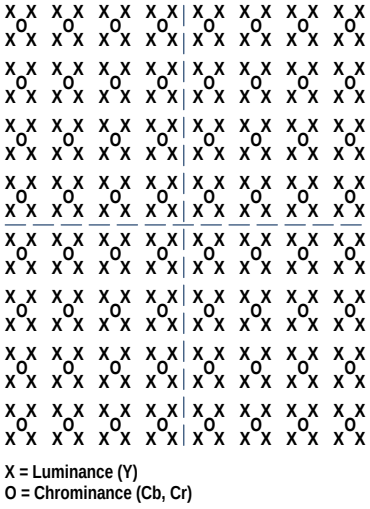


Fig. 4. Subsampling of the chrominance components (Cb, Cr) with respect to the luminance (Y) component.

MPEG Decompression

MPEG decompression reverses the functional steps taken for MPEG compression. There are six basic steps involved in MPEG decompression.

1. The MPEG header is decoded. This gives information such as picture rate, bit rate, and image size.

2. The video data stream is Huffman or entropy decoded from variable-length codes into fixed-length numbers. This step includes run-length decoding of zeros.

3. Inverse quantization is performed on the numbers to restore them to their original range.

4. An inverse discrete cosine transform is performed on the 8-by-8 blocks in each frame. This converts from the frequency domain back to the original spatial domain. This gives the actual pixel values for I-blocks, but only the differences for each pixel for P-blocks and B-blocks.

5. Motion compensation is performed for P-blocks and B-blocks. The differences calculated in step 4 are added to the pixels in the reference block as determined by the motion vector for P-blocks and to the average of the forward and backward reference blocks for B-blocks.

6. The picture is displayed by doing a color conversion from YCbCr coordinates to RGB color coordinates and writing to the frame buffer.

Methodology

Our philosophy was to improve the algorithms and tune the software first, resorting to hardware support only if necessary. We set a goal of 10 to 15 frames/s for software MPEG video decompression because this is the rate at which motion appears smooth rather than jerky.

We started by measuring the performance of the MPEG software we had purchased. This software initially took two seconds to decode one frame (0.5 frame/s) on an older 50-MHz Model 720 workstation. This decoding was for video only and did not include audio. Profiling indicated that the inverse discrete cosine transform (step 4) took the largest chunk of the execution time, followed by display (step 6), followed by motion compensation (step 5). The decoding of the MPEG headers was insignificant.

With this data we set out to optimize every step in the MPEG decompression software. After we applied all the algorithm enhancements and software tuning, we measured the MPEG decode software again. While we had achieved an order of magnitude improvement, the rate of 4 to 5 frames/s was not sufficient to meet our goal.

Hence, we looked at possible multimedia enhancements to the basic PA-RISC processor and other system-level enhancements that would not only speed up MPEG decoding, but also be generally useful for improving performance in other computations. In addition, any chip enhancements we added could not adversely impact the design schedule, complexity, cycle time, and chip size of the PA-RISC processor we were targeting, the PA 7100LC, which was already deep into its implementation phase at the time. The PA 7100LC is described in detail in the article on page12.

We approached this problem by studying the distribution of operations executed by the software MPEG decoder. Then, we found ways to reduce the execution time of the most frequent operation sequences. The application of algorithm enhancements, software tuning, and projected hardware enhancements was iterated until we attained our goal of being able to decompress at a rate greater than 15 frames/s via software.

Algorithm and Software Optimizations

In terms of MPEG video algorithms, we improved on the Huffman decoder, the motion compensation, and the inverse discrete cosine transform. A faster Huffman decoder based on a hybrid of table lookup and tree-based decoding is used. The lookup table sizes were chosen to reduce cache misses. For motion compensation, we sped up the pixel averaging operations.

For the inverse discrete cosine transform, we use a faster Fourier transform, which significantly reduces the number of multiplies for each two-dimensional 8-by-8 inverse discrete cosine transform. In addition, we use the fact that the 8-by-8 inverse transform matrices are frequently sparse to further reduce the multiplies and other operations required.

The MPEG audio decompression is also done in software. This algorithm was improved by using a 32-point discrete cosine transform to speed up the subband filtering.[12]

In terms of software tuning, we "flattened" the code to reduce the number of procedure calls and returns, and the frequent building up and tearing down of contexts present in the original MPEG code. We also did "strength reductions" like reducing multiplications to simpler operations such as shift and add or table lookup.

The last column of Table I shows the percentage of execution time spent in each of the six MPEG decompression steps after the algorithm and software tuning improvements were made. The first two columns of Table I show the millions of instructions executed in each of the six decompression steps and the percent of the total instructions executed (path length) each step represents. The input video sequence was an MPEG-compressed clip of a football game. The total time taken was 7.45 seconds on an HP 9000 Model 735 99-MHz PA-RISC workstation, with 256K bytes of instruction cache and 256K bytes of data cache.

Table I
Instructions and Time Spent in each MPEG Decompression Step on an HP 9000 Model 735

	Millions of Instructions	Path Length (%)	Time (%)
Header decode	0.6	0.1	0.1
Huffman decode	55.3	10.2	7.5
Inverse quantization	8.7	1.6	2.4
Inverse discrete cosine transform	206.5	38.3	38.7
Motion compensation	79.9	14.8	18.3
Display	188.7	35.0	33.0
Total	539.7	100.0	100.0

The largest slice of execution time (38.7%) and the largest chunk of instructions executed (38.3%) were still the inverse discrete cosine transform. We studied the frequencies of generic operations in this group and attempted to execute them faster. This resulted in new PA-RISC processor instructions for accelerating multimedia software.

PA-RISC Processor Enhancements

The new processor multimedia instructions implemented in the PA 7100LC processor allow simple arithmetic operations to be executed in parallel on subword data in the standard integer data path. In particular, the integer ALU is partitioned so that it can execute a pair of arithmetic operations in a single cycle with a single instruction. The arithmetic operations accelerated in this way are add, subtract, average, shift left and add, and shift right and add. The latter two operations are effective in implementing multiplication by constants.

PA-RISC Multimedia Extensions 1.0. The PA 7100LC PA-RISC processor chip contains some instructions that operate independently and in parallel on two 16-bit data fields within a 32-bit register. These operations are independent in that bits carried or shifted out of one of the fields never affects the result in the other field. These operations occur in parallel in that a single instruction computes both 16-bit fields of the result. Table II summarizes these instructions.

HADD does two parallel 16-bit additions on the left and the right halves of registers ra and rb, placing the two 16-bit results into the left and right halves of register rt.

HSUB does two parallel 16-bit subtractions on the left and right halves of registers ra and rb, placing the two 16-bit results into the left and right half of register rt.

Both HADD and HSUB perform modulo arithmetic (modulus 2^{16}), that is, the result wraps around from the largest number back to the smallest number and vice versa. This is the usual mode of operation of twos complement adders when overflow is ignored.

HADD and HSUB also have two saturation arithmetic options. With the signed saturation option, HADD.ss, both operands and the result are considered signed 16-bit integers. If the result cannot be represented as a signed 16-bit integer, it is clipped to the largest positive value (2^{15}–1) if positive overflow occurs, or it is clipped to the smallest negative value (-2^{15}) if negative overflow occurs.

With the unsigned saturation option, HADD.us, the first operand (ra) is considered an unsigned 16-bit integer, the second operand (rb) is considered a signed 16-bit integer, and the result (in rt) is considered an unsigned 16-bit integer. If the result cannot be represented as an unsigned 16-bit integer, it is clipped to the largest unsigned value (2^{16}–1) if positive overflow occurs, or it is clipped to the smallest unsigned value (0) if negative overflow occurs.

The signed saturation and unsigned saturation options for parallel halfword subtraction are defined similarly.

HAVE, or halfword average, gives the average of each pair of halfwords in ra and rb. It takes the sum of parallel halfwords and does a right shift of one bit before storing each 16-bit result into rt. During the one-bit right shift, the carry is shifted in on the left and unbiased rounding* is performed on the least-significant bit on the right. Because the carry is shifted in, no overflow can occur in the HAVE instruction.

HSLkADD, or halfword shift left and add, allows one operand to be shifted left by k bits (where k is 1, 2, or 3) before being added to the other operand.

HSRkADD, or halfword shift right and add, allows one operand to be shifted right by k bits (where k is 1, 2, or 3), before being added to the other operand.

Both HSLKADD and HSRKADD use signed saturation.

* Unbiased rounding means that the net difference between the true averages and the averages obtained after unbiased rounding is zero if the results are equally distributed in the result range.

Table II
PA-RISC Multimedia Instructions in PA 7100LC
ra contains a1; a2
rb contains b1; b2
rt contains t1; t2

Instruction	Parallel Operation
HADD ra,rb,rt	t1 = (a1+b1) mod2^{16}; t2 = (a2+b2) mod2^{16};
HADD.ss ra,rb,rt	t1 = IF (a1+b1) > (2^{15}–1) THEN (2^{15}–1) ELSEIF (a1+b1) < -2^{15} THEN (-2^{15}) ELSE (a1+b1); t2 = IF (a2+b2) > (2^{15}–1) THEN (2^{15}–1) ELSEIF (a2+b2) < -2^{15} THEN (-2^{15}) ELSE (a2+b2);
HADD.us ra,rb,rt	t1 = IF (a1+b1) > (2^{16}–1) THEN (2^{16}–1) ELSEIF (a1+b1) < 0 THEN 0 ELSE (a1+b1); t2 = IF (a2+b2) > (2^{16}–1) THEN (2^{16}–1) ELSEIF (a2+b2) < 0 THEN 0 ELSE (a2+b2);
HSUB ra,rb,rt	t1 = (a1–b1) mod2^{16}; t2 = (a2–b2) mod2^{16};
HSUB.ss ra,rb,rt	t1 = IF (a1–b1) > (2^{15}–1) THEN (2^{15}–1) ELSEIF (a1–b1) < -2^{15} THEN (-2^{15}) ELSE (a1–b1); t2 = IF (a2–b2) > (2^{15}–1) THEN (2^{15}–1) ELSEIF (a2–b2) < -2^{15} THEN (-2^{15}) ELSE (a2–b2);
HSUB.us ra,rb,rt	t1 = IF (a1–b1) > (2^{16}–1) THEN (2^{16}–1) ELSEIF (a1–b1) < 0 THEN 0 ELSE (a1–b1); t2 = IF (a2–b2) > (2^{16}–1) THEN (2^{16}–1) ELSEIF (a2–b2) < 0 THEN 0 ELSE (a2–b2);
HAVE ra,rb,rt	t1 = (a1+b1)/2; t2 = (a2+b2)/2;
HSLkADD ra,k,rb,rt	t1 = (a1≪k) + b1; t2 = (a2≪k) + b2; (for k = 1, 2, or 3)
HSRkADD ra,k,rb,rt	t1 = (a1≫k) + b1; t2 = (a2≫k) + b2; (for k = 1, 2, or 3)

ss = signed saturation option
us = unsigned saturation

Saturation Arithmetic. In saturation arithmetic a result is said to have a positive overflow if it is larger than the largest value in the defined range of the result. It is said to have a negative overflow if it is smaller than the smallest value in the defined range of the result. If the saturation option is used for the HADD and HSUB instructions, the result is clipped to the maximum value in its defined range if positive overflow occurs and to the minimum value in its defined range if negative overflow occurs. This further speeds up the processing because it replaces using about ten instructions to check for positive and negative overflows and performs the desired clipping of the result for a pair of operations in one instruction.

Saturation arithmetic is highly desirable in dealing with pixel values, which often represent hues or color intensities. It is undesirable to perform the normal modulo arithmetic in which overflows wrap around from the largest value to the smallest value and vice versa. For example, in 8-bit pixels, if 0 represents black and 255 represents white, a result of 256 should not change a white pixel into a black one, as would occur with modulo arithmetic. In saturation arithmetic, a result of 256 would be clipped to 255.

Effect on MPEG Decoding. These parallel subword arithmetic operations significantly speed up several critical parts of the MPEG decoder program, especially in the inverse discrete cosine transform and motion compensation steps. More than half of the instructions executed for the inverse transform step are these parallel subword arithmetic instructions. Their implementation does not impact the processor's cycle time, and adds less than 0.2% of silicon area to the PA 7100LC processor chip. Actually, the area used was mostly empty space around the ALU, so that these multimedia enhancements can be said to have contributed to more efficient area utilization, rather than adding incremental chip area. See "Overview of the Implementation of the Multimedia Enhancements" on page 66.

Since the PA 7100LC processor has two integer ALUs, we essentially have a parallelism of four halfword operations per cycle. This gives a speedup of four times, in places where the superscalar ALUs can be used in parallel. Because of the built-in saturation arithmetic option, speedup of certain pieces of code is even greater.

System Optimization

The second longest functional step (see Table I) in MPEG decompression was the display step. Here, we leveraged the graphics subsystem to implement the color conversion step together with the color recovery already being done in the graphics chip.[7] Color conversion converts between color representations in the YCbCr color space and the RGB color space. Color recovery reproduces 24-bit RGB color that has been color compressed into 8 bits before being displayed. Color compression allows the use of 8-bit frame buffers in low-cost workstations to achieve almost the color dynamics of 24-bit frame buffers. This leveraging of low-level pixel manipulations close to the frame buffer between the graphics and video streams also contributed significantly to the attainment of real-time MPEG decompression. Color recovery and the graphics chip are described in the articles on pages 51 and 43, respectively.

Other PA 7100LC processor enhancements streamline the memory-to-I/O path. By having the memory controller and the I/O interface controller integrated in the PA 7100LC chip, overhead in the memory-to-frame-buffer bandwidth is reduced. Overhead in the processor-to-graphics-controller-chip path is also reduced for both control and data.

Path Length Reduction

Table III shows the same information as Table I but for the low-end Model 712 workstation which uses the multimedia-enhanced PA 7100LC processor and the graphics chip mentioned above.

Table III
Instructions and Time Spent in each MPEG Decompression Step on a Model 712 Workstation

	Millions of Instructions	Path Length (%)	Time (%)
Header decode	0.60	0.2	0.3
Huffman decode	55.0	16.1	14.5
Inverse quantization	8.9	2.6	4.5
Inverse discrete cosine transform	138.5	40.6	34.4
Motion compensation	74.8	21.9	25.6
Display	63.0	18.5	20.7
Total	340.8	100.0	100.0

The Model 712 executes consistently fewer instructions than the Model 735 for the same MPEG decompression of the same video clip. It is also faster in MPEG decompression even though it operates at only 60% of the 99-MHz rate of the high-end Model 735 and has only one eighth of the cache size. This shows the performance benefits from the path length reduction enabled by the PA-RISC processor and system enhancements for multimedia acceleration.

Performance

The performance of the PA-RISC architectural enhancements and the leveraging of the graphics subsystem for video decompression can be seen in Fig. 5. This data is for a

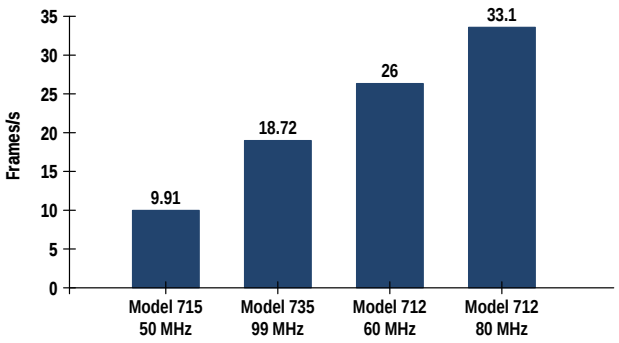


Fig. 5. Maximum MPEG decode frame rates for different models of HP 9000 Series 700 workstations. These rates are for a 352-by-240-pixel clip that was encoded at 30 frames/s.

Overview of the Implementation of the PA 7100LC Multimedia Enhancements

One goal in adding the multimedia instructions was to minimize the amount of new circuits to be added to the existing ALUs and to minimize the impact on the rest of the CPU. This goal was accomplished. The only circuit changes to the CPU were in the ALU data path and decoder circuits. These instructions reuse most of the existing functionality and very small modifications and additions were required to implement them.

All of the new instructions implemented require two 16-bit adds or subtracts to be done in parallel. The existing ALU adder was modified to provide this functionality. These instructions required that the existing 32-bit adder be conditionally split into two 16-bit halves without sacrificing the performance of the 32-bit add. Conceptually this is equivalent to blocking the carry from bit 16 to bit 15 in a ripple-carry adder. To accomplish this, we made the following modifications.

The ALU adder is similar to a carry lookahead adder. The first stage of the adder calculates a carry generate and a carry propagate signal for each single bit in the adder. In this case, 32 single-bit generate and 32 single-bit propagate signals are calculated. These single-bit carry generate and carry propagate signals are used in subsequent stages of the carry chain to calculate carry generate and carry propagate signals for groups of bits.

The 32-bit adder was divided into two 16-bit halves between bits 15 and 16 by providing alternate signals for the carry generate and carry propagate signals from bit 16 (Fig. 1). The new generate and propagate signals from bit 16 are created with a two-input multiplexer. When a 32-bit addition or subtraction is being performed, the multiplexer selects the original generate and propagate signals to be passed onto the next stage of the carry chain. When 16-bit addition or subtraction is being performed the multiplexer selects the value for generate and propagate from the second input which is false (logical 0) for additions and true (logical 1) for subtractions.

The new generate and propagate signals can be forced to be false for instructions requiring halfword addition. This stops the carry from being generated by bit 16 or propagating from bit 16 to bit 15, even if this generate and propagate signal is not used directly to calculate the carry signal (as is the case in this adder). The generate and propagate signals can also be forced to be true for instructions requiring halfword subtraction. This will force a carry into the more significant halfword of the adder by generating a carry from bit 16 into bit 15. This technique is used along with the ones complement of the operand to be subtracted to perform subtraction as twos complement addition.

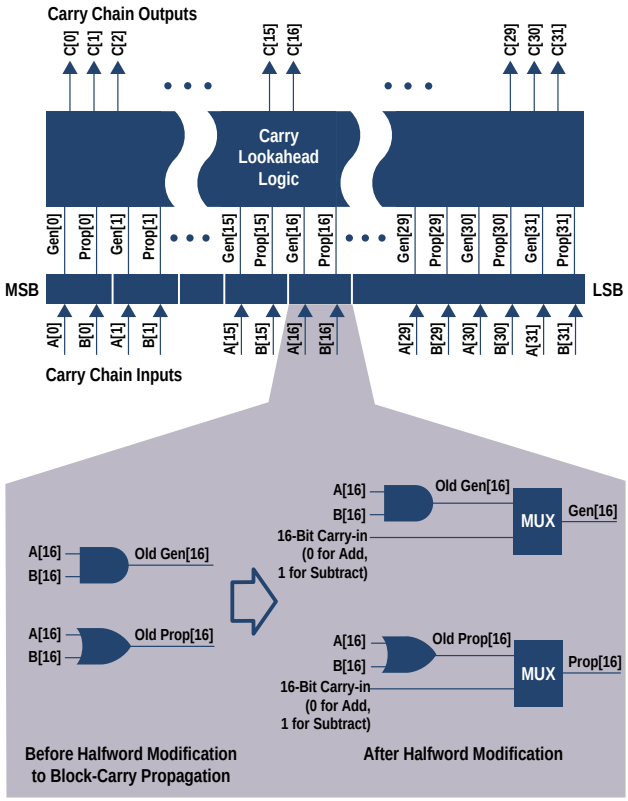


Fig. 1. Modifications to the carry lookahead adder to accommodate the halfword instructions.

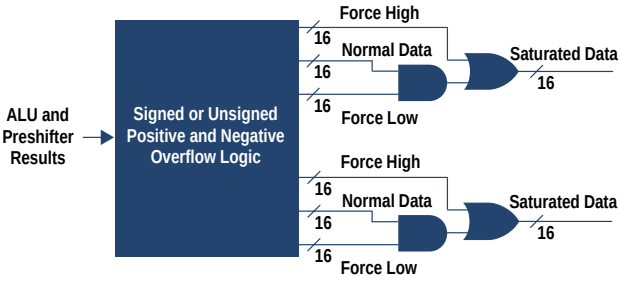


Fig. 2. Saturation logic. There is one of the these circuits for each halfword.

The original carry generate and propagate signals from bit 16 are still generated to calculate overflows from the less significant halfword addition. This overflow is used by the saturation logic, which can be invoked by some of these instructions.

Saturation requires groups of bits of the result to be forced to states of true or false, or passed unchanged. This is accomplished with an AND-OR gate (Fig. 2). The AND function can force the output of the gate to be false and the OR function can force the output of the gate to be true. Thus, the output is either forced high, forced low, or forced neither high nor low. It is never simultaneously forced high and low. The key is to determine when to force the result to a saturated value.

The saturation circuit is added at the end of the ALU's data path after the result selection multiplexer selects one of the results from the adder after it performs additions, subtractions, or logical operations such as bitwise AND, OR, or XOR (Fig. 3). The saturation circuit does not impact the critical speed paths of the ALU because it is downstream from the point where the cache data address is driven from the adder and where the test condition logic (i.e., logic for conditional branch instructions) obtains the results from which to calculate a test condition.

If signed saturation is selected, the ALU will force any 16-bit result that is larger than 0x7fff to 0x7fff (2^{15}–1) and any 16-bit result that is smaller than 0x8000 to 0x8000 (-2^{15}). These conditions represent positive and negative overflow of signed numbers. Positive and negative overflow can be detected by examining the sign bit (the MSB) of each operand and the result of the add. If both operands are positive and the result is negative then a positive overflow has occurred and the result in this case is saturated by forcing the most-significant bit to a logical 0 and the rest of the bits to a logical 1. If both operands are negative and the result is positive then a negative overflow has occurred and the result in this case is saturated by forcing the most significant bit to a logical 1 and the rest of the bits to a logical 0. Unsigned saturation is implemented in a similar way.

The average instruction, HAVE, requires manipulating the result after the addition is finished. Before the implementation of the halfword instructions the ALU selected between the results of a bitwise AND, a bitwise OR, a bitwise XOR, or the sum of the two input operands. The halfword average instruction adds an additional choice. The average result is the sum of the two input operands shifted right one bit position with a carry out of the most-significant bit (MSB) becoming the MSB of the result. To perform rounding of the result, the least-significant bit (LSB) of the result is replaced by an OR of the two least-significant bits before shifting right one bit.

The shift right and add and the shift left and add functions were added by modifying the x-bus preshifter in the operand selection logic of the ALU. The original ALU was capable of shifting 32-bit inputs left by zero, one, two, or three bits. To implement the 16-bit shift left and add instructions, the left-shift circuits had to be broken at

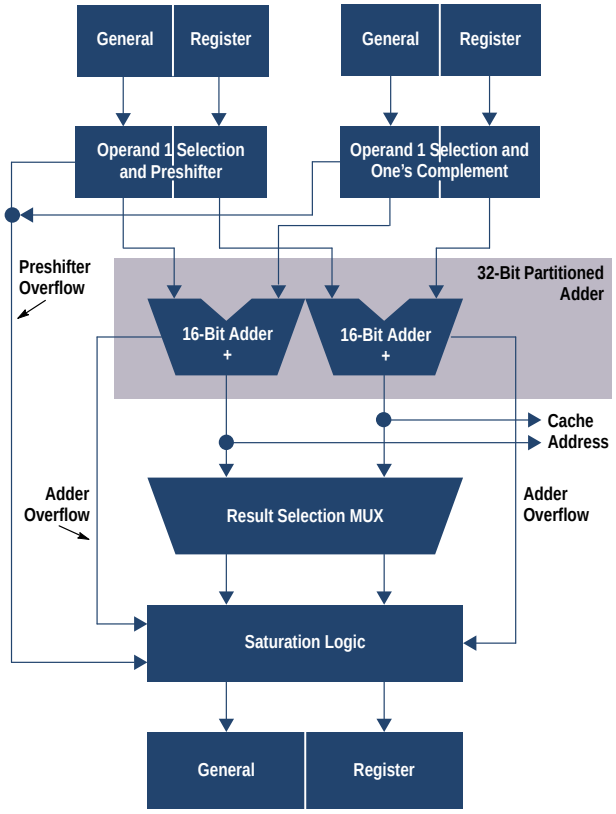


Fig. 3. Flow of halfword instructions showing the location of the saturation logic in relation to the ALU.

the halfword boundary. This was done by ANDing the bits shifted from the least-significant halfword to the most-significant halfword with a control signal that indicates when a 32-bit shift is being done. The 16-bit shift right and add instructions were implemented by adding the ability to shift one, two, or three bits right. This shift is always broken at the halfword boundary.

One challenging aspect of implementing the 16-bit shift left and add instructions was detecting when the results of shifting an operand left by one, two, or three bits causes a positive or negative overflow. A positive overflow occurs when the unshifted operand is positive and a logical one is shifted out of the left, or when the result of the shift is negative. A negative overflow occurs when the unshifted operand is negative and a logical zero bit is shifted out of the left, or when the result of the shift is positive. These overflow conditions are combined with the overflows calculated by the adder and used to saturate the final result. The final result is saturated if either the left shift or the adder causes an overflow.

The result of selecting instructions that can provide the most useful functionality while costing the least to implement was a relatively small increase in the area of the ALU. About 15% of the ALU's area is devoted to halfword instructions. Since the ALU's circuits were the only ones modified on the processor chip, only about 0.2% of the total processor's chip area is devoted to halfword instructions.

video clip that was compressed at 30 frames/s. The Model 715 and Model 735 are based on the PA 7100 processor. The Model 712 is based on the PA 7100LC processor, which is a derivative of the PA 7100. The PA 7100LC contains the multimedia enhancements and system integration features and is described in the article on page 12. The older, high-end Model 735 running at 99 MHz achieves 18.7 frames/s while the newer entry-level Model 712 achieves 26 frames/s at 60 MHz and 33.1 frames/s at 80 MHz. These frame decompression rates are quoted for MPEG video only (no audio) with

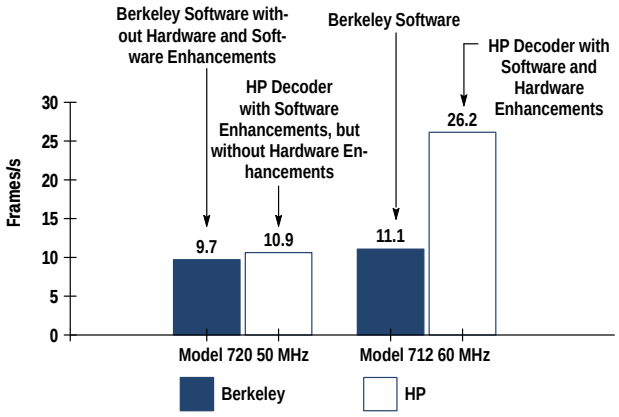


Fig. 6. Comparison between the performance of the enhanced Berkeley MPEG decoder and the HP MPEG decoder (without audio).

no constraints on how fast the decoding can proceed. In other words, the decoding rate is not constrained by the rate at which the MPEG stream has been compressed. Hence, although the video clip used was MPEG compressed at 30 frames/s, the 80-MHz Model 712 can decode it faster than 30 frames/s in unconstrained mode. This implies that there is some processor bandwidth left after achieving real-time software MPEG video decoding.

In the video player product in HP MPower 2.0, frames are skipped if the decoder cannot keep up with the desired real-time rate. This results in a lower effective frame rate, since skipped frames are not counted, even though execution time may have been used for partial decoding of a skipped frame.

Fig. 6 shows a comparison between the enhanced Berkeley software MPEG decoder and the HP software MPEG decoder running on the older HP 9000 Model 720 (with no hardware multimedia enhancements) and the newer Model 712 workstation (with hardware multimedia enhancements). The fourth column in Fig. 6 illustrates the performance obtainable with synergistic software and hardware enhancements.

In the Model 720, the Berkeley and HP software decoders have comparable performance. For the Model 712, the performance of the HP decoder was 2.4 times greater than the Berkeley decoder because of the synergistic coupling of the algorithms and software optimized with the PA-RISC multimedia instructions and the system-level enhancements in the Model 712.

Fig. 7 shows the performance when MPEG audio of various fidelity levels is also decompressed by software running on the general-purpose PA 7100LC processor. The highest-fidelity audio is stereo with no decimation. This means that every audio sample comes as a pair of left and right channel values, and every sample is used. Half decimation means that one out of every two audio samples is used. (3/4 decimation means that only one out of every four audio samples is used.) Mono means that every audio sample is a single value (channel) rather than a pair of values.

While software decompression of MPEG audio degrades the performance in terms of frames decoded per second, the PA 7100LC-based workstations achieved rates of 15.1 frames/s at 60 MHz, 24.2 frames/s at 80 MHz, and 27.4 frames/s at

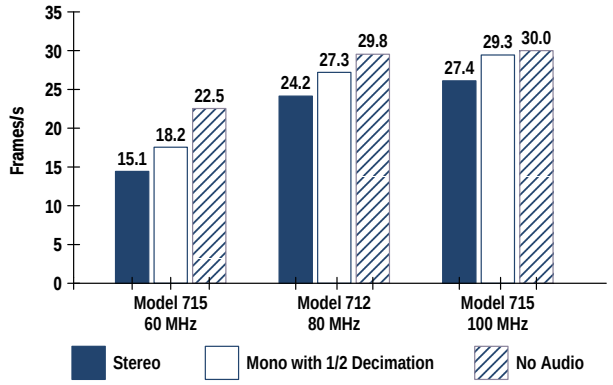


Fig. 7. Performance when MPEG video and MPEG audio are decoded in software.

100 MHz even with the highest-fidelity 44.1-kHz stereo 16-bit linear audio format with no decimation. With further enhancements of audio decoding and audio-video synchronization, we should be able to do even better.

Conclusion

We wanted a software approach to MPEG decoding because we felt that if video is to be useful it has to be pervasive, and to be pervasive, it should exist at the lowest incremental cost on all platforms. With a software video decoder, there is essentially no additional cost. In addition, the evolving standards and improving algorithms pointed to a flexible solution, like software running on a general-purpose processor. Using special-purpose chips designed for MPEG decoding, or even for JPEG, MPEG, and H.261 compression and decompression, would not allow one to take advantage of improved algorithms and adapt to evolving standards without buying and installing new hardware.

Furthermore, since the performance of general-purpose microprocessors continues to improve with each new generation, we wanted to be able to leverage these improvements for multimedia computations such as video decompression. This approach also allows us to focus hardware design efforts on improving the performance of the general-purpose processor and system without having to replicate performance efforts in each special-purpose subsystem, such as the graphics and video subsystems. The PA-RISC multimedia instructions are also useful for graphics, image, and audio computations, or any computations requiring arithmetic on a lot of numbers with precision less than 16 bits.

The net result is that we achieve real-time MPEG decoding of video streams at 30 frames/s with a software decoder. This was achieved by a synergistic combination of algorithm enhancements, software tuning, PA-RISC processor multimedia enhancements, combining video and graphics support for color conversions and color compression, and system tuning. The PA-RISC multimedia enhancements allow parallel processing of pixels in the standard integer data path at an insignificant addition to the silicon area. The total area used is less than 0.2% of the PA 7100LC processor chip with no impact on the cycle time or the control complexity.

The real-time software MPEG decoding rate of the final video player product exceeds our original goal of 10 to 15 frames/s for a software-based MPEG video decoder. It is also significant that MPEG video decoding at 30 frames/s is achieved by an entry-level rather than a high-end workstation. This is in the context of a full-function video player on the HP MPower 2.0 product. With MPEG audio decoding (also done by software), the frame rate is usually above 15 frames/s, even for the low-end Model 712/60 workstation, and around 24 frames/s for the Model 712/80 workstation.

We expect to see continuous improvement in the MPEG decoding rate as the performance of the general-purpose processors increases. With PA-RISC processors, there has been roughly a doubling of performance every 18 to 24 months. This would imply that larger frames sizes, multiple video streams, or MPEG–2 streams may be decoded in the future by such multimedia-enhanced general-purpose processors.

Acknowledgments

The success of the MPEG software decoder performance tuning depended on close, interdivisional teamwork distributed across four HP sites at Chelmsford, Cupertino, Palo Alto, and Fort Collins. We would like to thank all the other members of the multimedia architecture team, especially Vasudev Bhaskaran, Peter Kaczowka, Behzad Razban, Pat McElhatton, Larry Thayer, Konstantine Konstantinides, and Larry McMahan. We would also like to thank the HP MPower team, the PA-RISC extensions team, especially Michael Mahon, the PA 7100LC team, especially Mark Forsyth and Charlie Kohlhardt, the Model 712 team, the performance lab, especially the late Tian Wang, and the compiler lab, especially Pat Kwan, for their support.

References

1. *Hewlett-Packard Journal,* Vol. 45, no. 2, April 1994.

2. R. Lee, "Precision Architecture," *IEEE Computer,* Vol. 22 no. 1, January 1989, pp. 78-91.

3. R. Lee, M. Mahon, and D. Morris, "Pathlength Reduction Features in the PA-RISC Architecture," *Proceedings of IEEE Compcon,* February 1992, pp. 129-135.

4. L. McMahan and R. Lee, "Pathlengths of SPEC Benchmarks for PA-RISC, MIPS and SPARC," *Proceedings of IEEE Compcon,* February 1993, pp. 481-490.

5. *Digital Compression and Coding of Continuous-Tone Still Images,* CCIT REC. T.81 0918-1, July 1992.

6. P. Knebel, et al, "HP's PA7100LC: A Low-Cost Superscalar PA-RISC Processor," *Proceedings of IEEE Compcon,* February 1993, pp. 441-447.

7. S. Undy, et al, "A VLSI Chip-Set for Graphics and Multimedia Workstations," *IEEE Micro,* Vol. 14, no. 2, April 1994, pp. 10-22.

8. L. Gwennap, "New PA-RISC Processor Decodes MPEG Video," *Microprocessor Report,* Vol. 8, no. 1, January 1994, pp . 16-17.

9. *Coding of Moving Pictures and Associated Audio for Digital Storage Media up to 1.5 Mbit/s*, ISO/IEC JTCI CD 11172, 1991.

10. K. Patel, B. Smith, and L. Rowe, "Performance of a Software MPEG Video Decoder," *Proceedings of First ACM International Conference on Multimedia,* August 1993, pp. 75-82.

11. D. LeGall, "MPEG – A Video Compression Standard for Multimedia Applications," *Communications of the ACM,* April 1991, Vol. 34 no. 4, pp 46-58.

12. K. Konstantinides, "Fast Subband Filtering in MPEG Audio Coding," *IEEE Signal Processing Letters,* Vol. 1, no. 2, February 1994, pp. 26-28.

HP TeleShare: Integrating Telephone Capabilities on a Computer Workstation

Using off-the-shelf parts and a special interface ASIC, an I/O card was developed that provides voice, fax, and data transfer via a telephone line for the HP 9000 Model 712 workstation.

by S. Paul Tucker

Integration of the telephone and the computer workstation is a natural step in the evolution of the electronic office. It allows the user to perform telephone transactions without having to change from the keyboard and mouse environment to the telephone and handset environment and vice-versa. This capability provides obvious benefits to a wide customer audience, especially those dealing with customer service and support. The HP TeleShare option card for the HP 9000 Model 712 workstation represents HP's first integrated telephony product. Coupled with multimedia technologies such as audio, video, and HP SharedX,[1] HP TeleShare provides the user with a powerful arsenal of communications tools. This article will focus mainly on the hardware aspects of the HP TeleShare product.

Features provided by HP TeleShare include:

- Two-line support, with each line configurable for voice, fax, or data
- Workstation audio support and mixing (stereo headset with built-in microphone included)
- Dual-tone multifrequency (DTMF) tone generation and detection
- Telephone line status and control
- Call progress support
- Caller-ID support
- V.32bis modem (14,400 bits/s) with V.42bis and MNP5 (Microcom Networking Protocol) compression and error correction
- Fax Group 3 Class II up to 14,400 bits/s.

Background

HP TeleShare began as an experimental interface card for the HP 9000 Series 300 workstations. It had simple voice-only telephone capabilities, including single-source audio record and playback, and it was perceived as useful (and entertaining) to those engineers who were fortunate enough to have the opportunity to use it. At some point, further investigation was needed and the HP TeleShare project team was formed. It was determined that fax and data modem capabilities were needed with close coupling to the workstation's audio capability. Dual telephone lines were included so the user could talk on one line and at the same time use the other line for faxing or data. The standard analog phone line interface was chosen over digital (i.e., ISDN) because of the relatively insignificant number of digitally equipped PBX systems.

The first incarnation of the current product was an external RS-232-driven box with stereo inputs for computer, line-in, and microphone audio, and stereo outputs for computer, line-out, and headphones. It provided dual-line operation and employed two DSP (digital signal processor) subsystems for maximum flexibility and performance in voice and data modes. Audio mixing was provided by dedicated analog hardware, and any combination of audio inputs could be sent to any output complete with treble and bass control. The audio capabilities were so good that HP TeleShare engineers always had their CD players plugged into the box and their headphones on. This forced development of an automatic audio mute feature when an incoming call was detected. Since modem functionality was a primary goal, the command interface for the box used a partial modem AT command set, along with some proprietary extensions for new functionality like setting audio gains, setting audio mix values, telling a DSP to reboot, and so on. These commands were delivered over the RS-232 interface and received the typical OK and error responses.

While the external box was well received in the lab and by customers, it was postponed indefinitely in favor of a lower-cost internal version with a proprietary interface available on a single workstation, the Model 712. The same DSP subsystem used in the external box was carried over to the Model 712 option card and some effort was exerted to leverage as much as possible of the external box's software interface and feature set into the new design.

Architecture

HP TeleShare is made up of two independent DSP subsystems that communicate with the workstation host through an interface chip called *XBAR* (see Fig. 1). Each DSP is coupled to a hybrid chip called a *data access arrangement,* which provides direct connection to a standard two-wire analog telephone line. HP TeleShare is tightly coupled with the workstation audio system to provide the highest degree of audio flexibility. For instance, line-in audio (perhaps from a CD player) could be sent to telephone line 0 while the party on that line is on hold. Simultaneously, the workstation

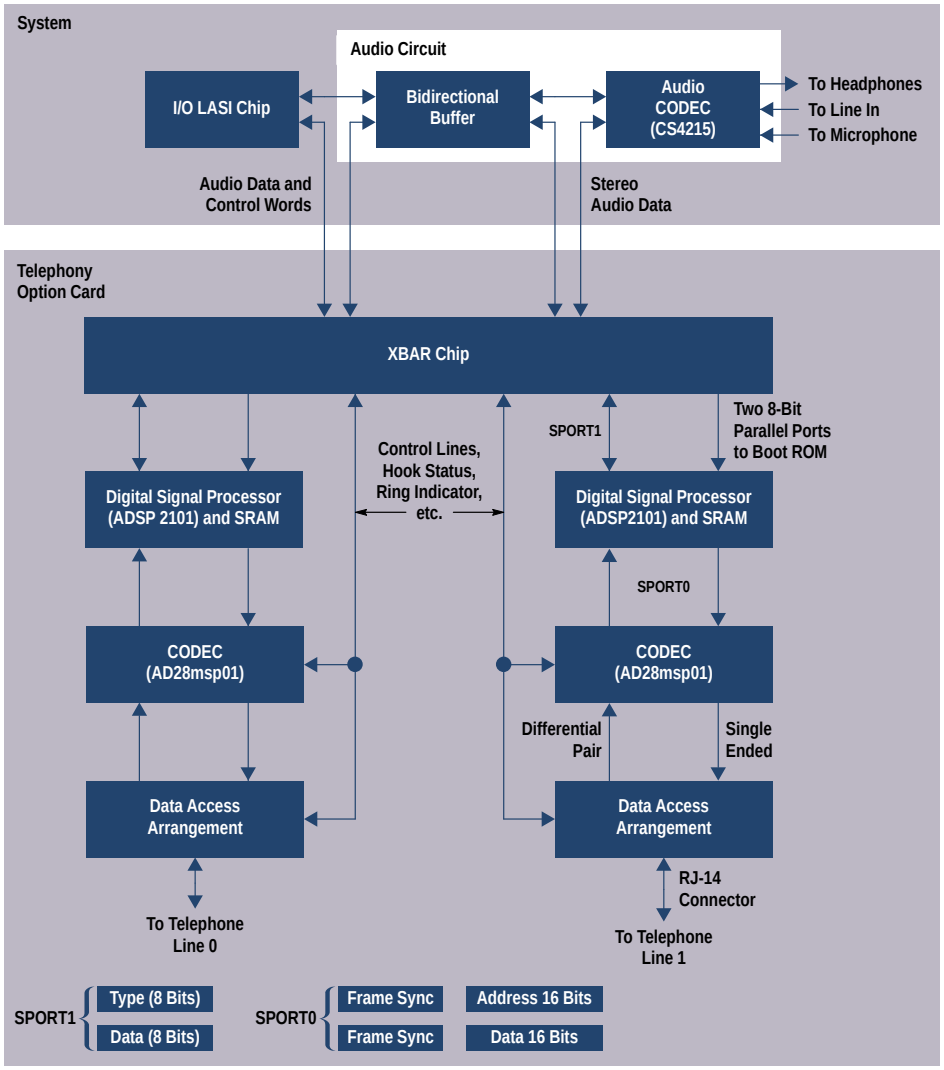


Fig. 1. Block diagram of HP TeleShare.

user could be conversing with or faxing a message to the party on telephone line 1. During this time comments from the party on line 0 could be recorded to disk for later playback.

XBAR. The XBAR ASIC (application-specific integrated circuit) is a custom VLSI part packaged in a low-cost 80-pin QFP (quad flat pack). It was designed by the HP TeleShare team specifically for use with the Model 712 workstation and performs all of the interface functions required by the HP TeleShare card. The XBAR chip communicates with the system I/O chip (LASI) and the audio CODEC (coder/decoder) through a pair of proprietary serial interfaces. If HP TeleShare is not present in the system, audio data and CODEC control words pass through the bidirectional buffer between the system I/O chip and the audio CODEC. When the HP TeleShare card is installed, XBAR is effectively placed between the system I/O chip and the CODEC, forcing all audio to be routed through XBAR in either direction. The serial interface between XBAR and the audio CODEC carries 16-bit stereo audio data.

The serial interface between XBAR and the I/O chip multiplexes 16-bit system audio data (to and from disk) and control words for XBAR. In addition, this interface is used for modem data, voice-mode AT commands and responses, and DSP application code downloaded from the host system. On the DSP side, XBAR has two 13.824-MHz serial ports, each designed specifically for interfacing with the DSPs. These ports are used for passing audio samples, modem data, application programs, and commands and responses to and from the DSPs.

XBAR's configuration can be changed by writing to the control registers in the XBAR-to-LASI I/O serial interface address space. In voice mode, XBAR is configured to pass each audio data sample from the LASI I/O chip and CODEC (coder/decoder) to a DSP, whereupon the DSP will return responses to each of those sources. XBAR can also send audio data samples from DSP to DSP for conferencing between lines when both lines are in voice mode. In data and fax modes, XBAR sends appropriately formatted data to the DSP and receives similar data in return.

Although XBAR supports stereo audio at up to a 48-kHz sample rate, DSP bandwidth limitations require all audio data to and from the telephone lines to be left-channel only, sampled at 8 kHz. This is not a serious limitation, since telephone-quality audio only requires a sample rate around 7.2 kHz for full reproduction and is inherently a single-channel signal.

In addition to the DSP serial ports, XBAR also has a pair of byte-wide parallel ports that connect to the DSPs' boot ROM

ports. This allows DSP boot code (not to be confused with DSP application code) to be downloaded from the host system. This provides additional flexibility and eliminates the cost and board-space limitations associated with external ROMs.

XBAR has several asynchronous control signals that are connected to downstream hardware, including reset lines for the DSP subsystems and hook control and telephone status (such as the ring indicator) signals to and from the data access arrangement chips.

The biggest challenge in XBAR's design was purely logistical because we had a lot of different data types (e.g., stereo data and telephone command and status data) to handle and very little time to implement them in XBAR. There are no less than 52 separate data types that XBAR must recognize and generate for the two DSP serial interfaces alone, with a slightly smaller number required for the system I/O interface. To provide these types, each transaction between XBAR and a DSP consists of 16 bits, with the upper eight bits providing type information and the lower eight bits providing the associated data. To prevent data overruns, XBAR requires a data acknowledge (Ack) word back from the appropriate DSP for every transaction.

Audio data samples in HP TeleShare are 16 bits long. Since XBAR sends eight bits of data at a time, audio samples must be broken into two pieces: an upper half, or most-significant byte (MSB) and a lower half, or least-significant byte (LSB). Using this model, it requires two transfers from XBAR and two Acks back to send one sample of audio data to a DSP. Sending one set of stereo system and line-in audio samples to a DSP requires eight output transfers (four transfers for the system sample and four transfers for the line-in sample), with an Ack back after each transfer. The DSP will then send mixed audio samples back for the system and the CODEC, requiring an additional eight transfers, for a total of 24 transfers per sample. This has to happen at an 8-kHz sample rate (once every 125 microseconds). Fortunately, XBAR can handle these transactions, but order must be maintained exactly or audio quality will suffer. Other data types, such as AT commands and responses, are given lower priority during audio frames and are queued until audio transfer is finished.

Digital Signal Processor. The DSP used by TeleShare is an Analog Devices ADSP2101. This is a programmable single-chip microcomputer optimized for digital signal processing, and operates at 16.67 MHz. The 2101 operates on 16-bit data and uses a 24-bit instruction word. It has 1024 words of data RAM and 2048 words of program RAM on the chip. The part has two data address generators and a program sequencer, which allows program and data accesses to occur simultaneously in a single cycle. Dual data operand fetches can also occur in a single cycle since program memory can also be used to store data. The part can address up to 16K words of data and 16K words of program memory, both of which are supplied on the HP TeleShare board in the form of six external SRAMs.

The DSP has two independent serial ports, SPORT0 and SPORT1, which support multiple data formats and frame rates and are fully programmable. In the HP TeleShare design, SPORT1 on each DSP is dedicated to communication with XBAR, while SPORT0 is dedicated to communication with an AD28msp01 telephone line CODEC. Each full transfer to or from one of the SPORT lines triggers an associated interrupt in the 2101, allowing programs to act on the incoming data as it arrives.

Analog Devices supplies a complete set of software development tools for the 2100 microprocessor family, including a C compiler with a DSP function library.

CODEC. The Analog Devices AD28msp01 provides HP TeleShare with a multiple-sample-rate CODEC specifically designed for use in modem designs. This device supports sample rates of 7.2 kHz, 8.0 kHz, and 9.6 kHz, and has an 8/7 mode* for sampling at 8.23 kHz, 9.14 kHz and 10.97 kHz. For voice mode operation, 7.2 kHz and 8.0 kHz are all that is required, but the sophisticated algorithms used by modern modem standards often require the other rates. The CODEC uses 16-bit sigma-delta conversion technology and includes resampling and interpolation filtering along with transmit and receive phase adjustments.

Each CODEC has one serial port which is connected directly to SPORT0 on the associated DSP. This port operates in free-running mode once it is properly initialized and continually sends 16-bit data samples from the telephone line to the DSP. All transfers to the DSP consist of a serial output frame sync followed by a 16-bit address word, then a second frame sync followed by a 16-bit data word. These address and data pairs are transmitted at the selected sample rate and trigger SPORT0 receive interrupts in the DSP. The DSP transfers data to the CODEC using the same mechanism as just described (in the other direction, of course). The address portion of each transfer coming from the DSP identifies the data as a control word (for programming the part) or as a data word to be sent through the on-chip digital-to-analog converter (DAC) and transmitted to the data access arrangement chip. Data from the CODEC to the DSP is identified as either a control word or as a data word from the on-chip analog-to-digital converter (ADC).

The AD28msp01 CODEC is attached to the telephone line through the data access arrangement chip. Transmit data outputs are differential for noise reduction, while the receive data input is single-ended.

Data Access Arrangement. HP TeleShare uses the TDK 73M9002 data access arrangement chip as its telephone line interface. This part provides all the necessary line monitoring, filtering, isolation, protection, and signal conversion functions for connection of high-performance analog modem designs to the PSTN (public switched telephone network) in the United States, Canada, and Japan. The 73M9002 incorporates, on a two-to-four-wire hybrid, ring detection circuitry, off-hook relay, and on-hook line monitoring for caller-ID support (see "Caller-ID" on page 72).

The 73M9002 comes with FCC (Federal Communications Commission) part 68 DOC CS-03 and JATE (Japan Approvals Institute for Telecommunications Equipment) protection circuitry built in, and is compliant with UL 1459 2nd Edition

* The 8/7 mode is a capability required by some modem applications. It simply adds some sampling bandwidth. For example, in 8/7 mode the normal 8-kHz sample rate becomes 9.14 kHz ($8 \times 8/7$).

Caller-ID

Caller-ID information is sent between the first and second power ringing signals. The data is sent a minimum of 500 milliseconds after the first ring and ends at least 200 milliseconds before the second ring begins. This leaves 2.9 to 3.7 seconds of time for data transmission. The data is sent at 1200 baud using frequency shift keying (FSK) modulation. All data is 8-bit ASCII.

Two standard formats exist for Caller-ID information: single message format and multiple message format. In general, both formats can be described using Fig. 1.

The message type is 0x4 (hexadecimal 4) for single message format. The message length is variable and indicates the number of message words in the message body. The final word is a checksum word, used for error checking. Single message format provides the receiver with date, time, and calling number data.

The message type is 0x80 (hexadecimal 80) for multiple message format. The message length is variable as before, but provides the receiver with date, time, calling number, and calling name data if available. In the absence of calling name data, a P indicating private or an O indicating out of area or unavailable will be sent.

Caller-ID detection requires on-hook line monitoring, which the HP TeleShare data access arrangement chip fully supports. HP TeleShare can detect and display both message formats.

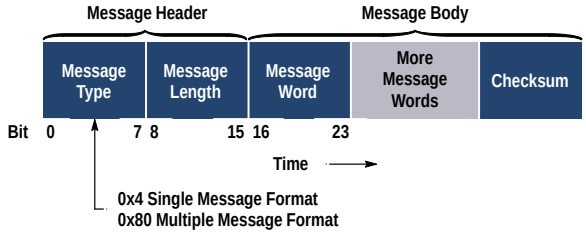


Fig. 1. Caller-ID message format.

with the addition of an external slow-blow fuse. The off-hook relay is controlled by a TTL-level input from XBAR. This relay determines when the phone is off the hook and can be pulsed for use as a pulse dialer. Another TTL-level input from XBAR is used to enable on-hook line monitoring.

The ring detection circuitry is capable of detecting ringing signals that comply with Ringing Type B from the FCC Part 68 regulations. The detected ring signal appears at a pair of differential outputs which are also connected to XBAR.

The 73M9002 provides telephone connectivity to the DSP subsystem through the CODEC's analog receive and transmit lines and is attached to the telephone line through a standard RJ-14 connector.

Operating Modes

HP TeleShare is capable of operating in three modes: voice, fax, and data. The modes are selected through a graphical user interface by the workstation user. The mode application software is downloaded through XBAR to a DSP as needed and runs continually until a reset of that DSP is performed. The voice mode code was developed solely by HP and can be run on one or both lines simultaneously. The fax and data modem code was developed with a third party and because of licensing restrictions, only one line can be configured as a fax or data modem at a time. Combinations of voice and fax or data are fully supported.

Voice Mode Operation. When configured in the voice mode, HP TeleShare essentially operates like an enhanced telephone. Digital mixing of microphone, line-in, telephone, and recorded audio (from system disk) is supported for both playback and recording. This capability allows numerous interesting audio configurations including placing a line on hold with music, recording conversations, playing back recorded audio over the phone, and so on. While in voice mode, HP TeleShare provides the user with caller-ID information if it is available. In addition, DTMF (dual-tone multi-frequency) tone and pulse dialing are supported, along with DTMF tone detection for unattended phone functions like answering machines or voicemail (see "Call Progress, DTMF Tones, and Tone Detection" on page 73).

Dialing and hook manipulation actions are performed through the GUI (graphical user interface), but at the lowest level these actions are sent to the DSP as standard AT commands like ATDT (attention dial tone) and ATH n (n = 0 is on-hook or hang up, and n = 1 is take telephone off hook). Special functions like audio mixing are also controlled with low-level AT-type commands, but are manipulated using sliders in the GUI.

The voice mode application firmware is driven primarily by DSP SPORT interrupts. Every incoming 16-bit SPORT0 word from XBAR triggers an interrupt, which in turn causes the SPORT1 interrupt service routine to execute. Likewise, every 16-bit SPORT0 word from the CODEC causes the SPORT0 interrupt service routine to execute. The SPORT1 interrupt service routine is responsible for audio I/O with XBAR and queueing AT commands as they arrive. Commands arrive asynchronously, that is, they can arrive at any time, while audio arrives in 8-piece bundles every 125 microseconds (one frame) as described earlier. Normally, every piece of data received by SPORT1 causes an interrupt, but the firmware disables these interrupts for the rest of a frame once it recognizes the first piece of audio data. Otherwise, at least eight context switches would occur every frame, which would render the system useless. Once the SPORT1 interrupt service routine has received all of the audio samples, it is responsible for transmitting the new audio back to XBAR for routing to the workstation (i.e., headphones and/or disk).

The SPORT0 interrupt service routine is responsible for receiving and transmitting telephone-line audio and mixing all audio data, including DTMF tones. Before mixing can occur in the DSP, all of the LSBs must be appended to the MSBs. Remember that each 16-bit sample transferred between XBAR and the DSP is divided so that the most-significant byte contains the data type and the least-significant byte contains the data. Thus, all the data from XBAR is put back into 16-bit linear format before transfer to the CODEC.

The audio input and output amplitude matrices, built by the user via the GUI, are used to determine what the final mix will sound like. The DSP firmware processes each output in sequence by adding together any inputs that are on to create a total value for each output. Any gain adjustments are made at this time as well. When this is completed for all outputs, the resulting 16-bit values are broken into MSBs and LSBs, if required.

Audio data that is meant for XBAR is transmitted during the next XBAR audio frame. Audio data meant for output to the

Call Progress, DTMF Tones, and Tone Detection

HP TeleShare's voice mode firmware has the ability to detect a number of tones used commonly in telephone communications, including DTMF tones and call progress tones like busy, ringback (the ringing sound you hear when you call someone), and dial tone.

DTMF Tones

Dual-tone multifrequency (DTMF) tones are made up of two separate tones, as the name suggests, and can be accurately generated using easily understood principles. The DTMF standard specifies two sets of distinct tones, called row frequencies and column frequencies (see Fig. 1). The row frequencies correspond to the horizontal rows on a standard telephone touchpad. The column frequencies correspond to the vertical columns on the touchpad, plus an additional column to the right of the last touchpad column.

This makes eight separate frequencies, which combine for a total of sixteen DTMF tones (see Fig. 2).

Generation of a DTMF tone is accomplished by creating a sinusoid for each of the two frequencies, row and column, and then adding the results. In a digital implementation, the sinusoids are computed and added on a sample-by-sample basis. HP TeleShare uses a five-coefficient Taylor series approximation for the sinusoid generation. The sinusoid samples are updated and added at 8 kHz, or every 125 microseconds, and the sum of the sinusoid samples is used as the current DTMF sample.

Tone Detection

Tone detection is accomplished through the use of a 512-point fast Fourier transform (FFT), which is implemented in the ADSP2101 C-language run-time library. The FFT, when given a set of samples of an input signal over some time interval, returns the frequency spectrum of the signal during the interval. This can be done in almost real time with a DSP, making it very useful for detecting incoming tones. The following important rules and relationships should be noted concerning sample rate, input points, output points, time, frequency, and the FFT in general:

- The FFT requires complex (real and imaginary) data for input (two arrays).
- The imaginary input array may be filled with zeros if unused.
- The output data is complex (two arrays).
- The frequency spectrum returned covers half of the sampling frequency.
- Only the first half of output data is used, and the other half is a mirror image.
- The output frequency resolution is equal to (sampling rate)/(number of input points).

Using an 8-kHz sampling rate and 512 points causes the FFT to return a spectrum from 0 to 4 kHz, with 512 complex output points. The second 256 output points can be ignored since they are the mirror image of the first 256. The output will have a resolution of 15.625 Hz per point, using the formula above. These output points will be referred to as bins since they include spectral data on either side of each point.

HP TeleShare calculates magnitude-squared values for each bin by squaring the real and imaginary values at each point and adding them. The magnitude-squared values correspond roughly to the power of the signal in each bin. Once the powers are known for each bin in the spectrum, they can be analyzed to see if any DTMF or call progress tones are present.

	Column Frequencies			
Row Frequencies	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

Fig. 1. Dual-tone-multifrequency digits and the frequencies associated with them.

	Tone	Frequency (Hz)	512-Point FFT Index
Call Progress Tones	Dial Tone	350 + 440	22.4 + 28.16
	Busy	480 + 620	30.72 + 39.68
	Ringback	440 + 480	28.16 + 30.72
16 DTMF Tones	DTMF 1	697 + 1209	44.61 + 77.38
	DTMF 2	697 + 1336	44.61 + 85.5
	DTMF 3	697 + 1477	44.61 + 94.53
	DTMF 4	770 + 1209	49.28 + 77.38
	DTMF 5	770 + 1336	49.28 + 85.5
	DTMF 6	770 + 1477	49.28 + 94.53
	DTMF 7	852 + 1209	54.53 + 77.38
	DTMF 8	852 + 1336	54.53 + 85.5
	DTMF 9	852 + 1477	54.53 + 94.53
	DTMF *	941 + 1209	60.22 + 77.38
	DTMF 0	941 + 1336	60.22 + 85.5
	DTMF #	941 + 1477	60.22 + 94.53
	DTMF A	697 + 1633	44.61 + 104.51
	DTMF B	770 + 1633	49.28 + 104.51
	DTMF C	852 + 1633	54.53 + 104.51
	DTMF D	941 + 1633	60.22 + 104.51

Fig. 2. Call progress and DTMF tones recognized by HP TeleShare.

As an example, suppose the telephone has been taken off-hook in preparation for dialing and HP TeleShare is configured to check for dial tone. 512 samples of the input signal would be stored in the real input array, while the imaginary array is filled with zeros. Next, the FFT function is called, returning the real and imaginary arrays. The magnitude-squared values of the first 256 bins are computed using the two output arrays. The two frequencies that make up a dial tone are 350 and 440 Hz (see Fig. 2), so FFT indexes (or bin numbers) must be computed for these frequencies:

350/15.625 = 22.4 440/15.625 = 28.16

An effective method of checking for the existence of a particular frequency is to compare the power present at that frequency with the total power of the spectrum. This is done quite easily with magnitude-squared values since they represent power in each bin already. Total power is simply the sum of all the magnitude-squared values for the first 256 FFT return values. Divide this into the power of the frequency being checked for, and the result is the percentage of total power for that frequency. For example, when checking for 350 Hz, compute the sum of the power values for bins 22 and 23 since the real index (22.4) falls between them, and then divide by the total power. The result is the percentage of the total power present around 350 Hz. The same can be done for 440 Hz, using bins 28 and 29.

Once the percentage of total power is calculated, a comparison can be made to see if the power in each frequency meets *match criteria.* The HP TeleShare firmware typically uses 35% of total power as a match condition. In other words, if the power present at the desired frequencies is 35% or more of the total power, dial tone has been detected. Otherwise, no dial tone is found.

The number of bins used in the comparison and the match criteria can be fine-tuned for a particular application. The match criteria can include other tests and can be relaxed or tightened as needed. The number of bins used can be influenced by the total number of points in the FFT and by a preprocessing tool that does windowing. Windowing is used to create a finite-length sequence from a continuous sequence. It is basically a digital filter that truncates an infinite-length input sequence while preserving its frequency characteristics. Since we are grabbing finite pieces (sequences) of data, we need to window the data.

telephone line is immediately sent to the CODEC without being split in half. Since both interrupt service routines run at 8 kHz, there is no need to worry about sample rate changes. DTMF audio data is only available for mixing when a tone is being generated. A new DTMF sample is generated during every SPORT0 interrupt and is based on the sample rate (always 8 kHz) and the time elapsed since the tone began.

All of these interrupts and audio manipulations require almost all of a DSP's processing bandwidth and can effect some areas of system performance. Because of DSP bandwidth limitations, DTMF detection can have a slight, but noticeable effect on the audio quality heard by the user. However, in unattended modes like answering machines or voicemail (where DTMF detection could be used for such things as navigation), this should not be a concern. The default configuration has DTMF detection disabled, since the typical user will never use it, and the current GUI does not support it.

Fax and Data Modem Operation. The fax and data modem functionality was codeveloped by HP and Digicom Systems Incorporated and uses their SoftModem technology. The fax mode allows transfers up to 14,400 bits/s and covers Group 3 Class II and all fallbacks. Data mode supports transfers up to 14,400 bits/s (V.32bis) and can reach peak rates of 57,600 bits/s with compression.

Conclusion

HP TeleShare effectively combines telephone communications capability with a low-cost computer workstation. Context switches between the display and the telephone are minimized by integrating the telephone into the computer system and providing an easy-to-use graphical user interface. Voice, fax, and high-speed data modes are supported using flexible digital signal processing technology.

References

1. D. Garfinkel, B. Welti, and T. Yip, "HP SharedX: A Tool for Real-Time Collaboration," *Hewlett-Packard Journal,* Vol. 45, no. 2, April 1994, pp. 23-36.

Product Design of the Model 712 Workstation and External Peripherals

A product design without fasteners and the use of environmentally friendly materials and low-cost parts with integrated functions provides excellent manufacturability, customer ease of use, and product stewardship.

by Arlen L. Roesner

The HP 9000 Model 712 workstation and and the three new peripherals that go with the product are an excellent example of computer integration and simplicity. The new workstation, while providing a new class of performance with HP's new PA-RISC PA 7100LC processor, pushed the envelope of product design by using relatively few and inexpensive parts. In addition to simplicity and low cost, the product promotes good product stewardship by making parts easy to identify and recycle. Customers find the hardware easy to manage because there are no fasteners to deal with, and all the components snap or drop into place. The main workstation product is a small compact size that fits easily under a monitor or stands vertically on the desk, and the external peripherals can be positioned on the desktop where they are most convenient to the user. Fig. 1 shows the Model 712 workstation and its three peripherals.

Outward Simplicity

Several assemblies of the Model 712 workstation products have high levels of functional integration. This functional integration tends to make components more complex, but yields an outer simplicity by reducing the number of physical parts and the methods necessary to work with them. Once configured, the only accessible components of the Model 712 workstation include the chassis, system board, option boards (including memory), disk drive, flexible disk drive, and top cover. All of these components are accessed through quick removal of the cover and the manipulation of a few snap or drop-in fits, which require a minimum of time and effort. Fig. 2 shows the workstation and one of the peripherals with their covers removed. Benefits of this resulting simplicity include better manufacturability, easier customer use and configuration, and serviceability.

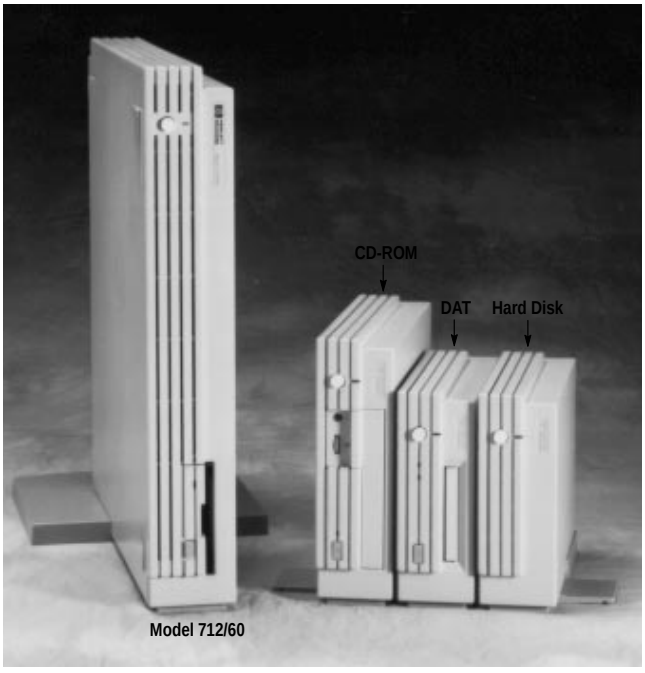


Fig. 1. HP 9000 Model 712 workstation and related external peripherals.

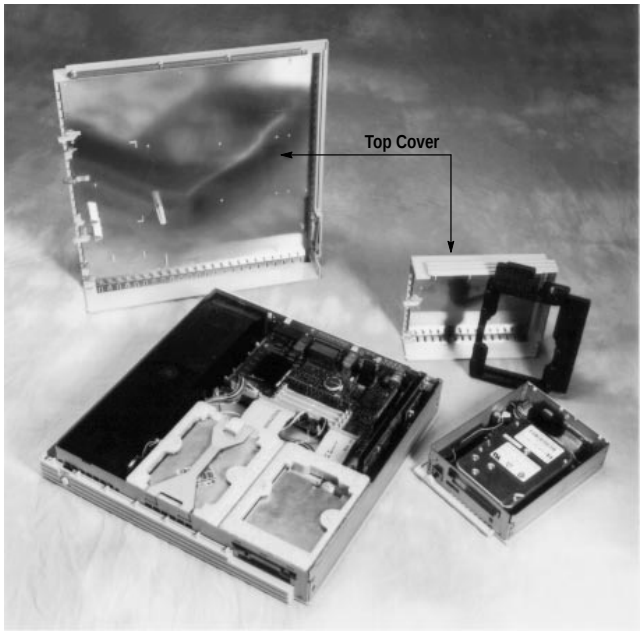


Fig. 2. The Model 712 workstation and hard disk peripheral with top covers disassembled.

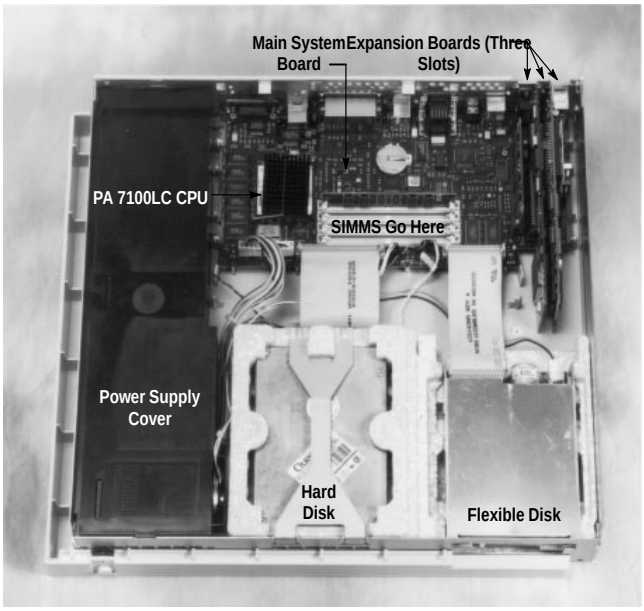


Fig. 3. Top view of the Model 712 without top cover.

Electronics

The system electronics is the place where integration is most likely to be first noticed in the Model 712 product. Electronic assemblies consist of one main system board, a power supply, three optional circuit boards, and up to four memory SIMMs. The main system board is relatively small, and all of the core electronics is incorporated onto this board through integration of functionality into relatively few VLSI components. (Fig. 3 in the article on page 9 shows the main system board). The main system board uses dual-sided surface mount construction, with I/O connector space being provided mostly by double-high (stacked) bulkhead connectors. Optional boards are provided for telephony, extra I/O, and high-resolution graphics. Compared to today's personal computers, the Model 712's system board functions are usually found on a personal computer's motherboard, backplane (if any), and two to three expansion boards. This level of integration on the Model 712 exceeds the density of personal computer functionality, while providing current workstation performance.

Chassis

The chassis assembly consists of a plastic base, a metal chassis, a metal liner for EMI containment of the rear I/O connectors, and a plastic rear dress panel (see Fig. 4). The dress panel includes silkscreened graphics to identify the connectors and state necessary regulatory information, eliminating the need for information labels. The chassis has a variety of holes and embossments to assist in joining the plastic parts to it. The plastic base provides outer air venting and cosmetic appeal to the product while also containing several snaps and guides for mating parts. The metal liner provides EMI finger contact to all connectors in one part, whereas previous products often required many different clips for such functionality. Held together via plastic heat stakes, the plastic base, the metal chassis, the metal liner, and the plastic dress panel make up the main assembly chamber of the product. The main circuit board, power supply and cover, disk brackets, and top cover all snap or drop into this chassis. Option boards are also easily installed into the chassis on top of the main system board, with integral bulkheads that mate vertically to chassis cutouts (also without fasteners).

Power Supply Cover

The power supply cover is another example of integration. Many parts were "designed out" by this single plastic part that performs six functions. The main function is to protect end users from dangerous voltages by shrouding the exposed power supply. The cover snaps into the chassis from front to rear and is removable only by using a screwdriver to disengage the snap that holds it in place. In addition to shrouding the power supply, the cover secures the power supply board in place, houses the fan and speaker, channels air flow, and provides structural support for the monitor. The fan simply snaps down inside the cover and seals to the sides and top of the cover. The speaker slides down and press fits into a simple pocket, which provides acoustic baffling. After the cover is installed, cables from these devices are routed to the main system board for electrical connection.

HP-PAC Disk Brackets

The disk brackets are made of HP's newly patented HP-PAC material.[1] This material is made of expanded polypropylene beads, and is used most often to produce shipping carton cushions for many types of products. Instead of placing this material around a finished product to cushion it in a shipping carton environment, it is instead formed to fit inside a product with integral recesses to embed internal components. For the Model 712 workstation, the HP-PAC material is used to hold the hard disk and flexible disk mechanisms in place. The HP-PAC used in the workstation consists of three parts: a bottom shell which provides a recess for both flexible and hard disk, and two separate top pieces for covering each disk mechanism (see bottom portion of Fig. 4). Because of the cushioning properties of the HP-PAC material, the disk drive mechanisms benefit from reduced shock and vibration levels. The HP-PAC material also provides integral air channels for inlet air to be drawn across hot areas of the disk drive mechanisms. The interesting feature of HP-PAC is that no screws are needed to install the mechanisms. The devices simply drop into recesses inside of the cushioning material, and cables can be connected directly to the embedded mechanisms. Once in place, the chassis enclosure then retains the top and bottom shells of HP-PAC around each device.

Top Cover

The top cover includes a configurable bezel for the flexible disk area, a plastic top shell, and a thin metal liner to complete the EMI enclosure. The liner is held to the cover via plastic heat stakes and has a series of fingers on each side of the cover to contact the chassis and contain EMI radiation. The flexible disk bezel is designed to snap into the front of the cover, which then configures the frontal appearance of the product. The cover assembly drops vertically onto the chassis and then slides rearward until alignment hooks and snaps in the cover engage to hold the cover in place.

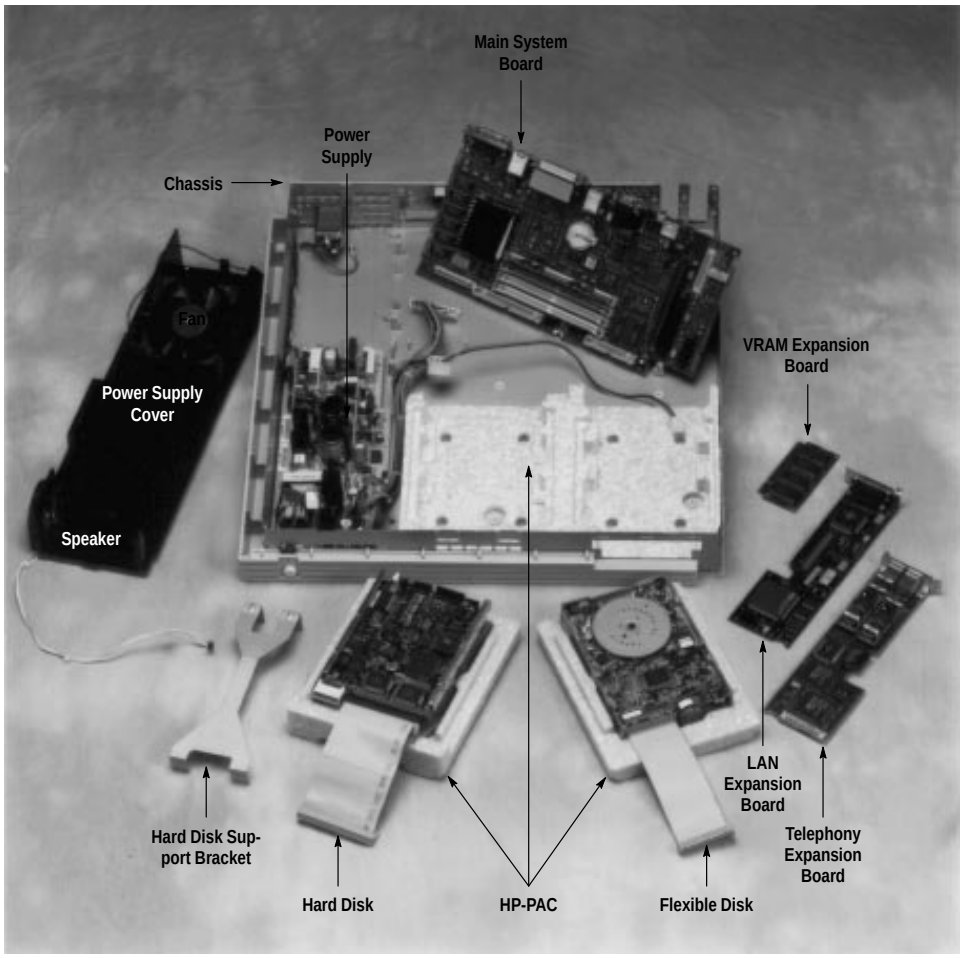


Fig. 4. The Model 712 workstation showing components disassembled from the chassis.

External Peripheral Products

The product design of the three external peripherals also includes a large degree of functional integration. Each of these boxes is designed as a miniature Model 712 workstation, with HP-PAC cushions providing location and support for the drive mechanism, a printed circuit board (for power conversion), power switch plunger, and cabling. The plastic cover for each product includes any necessary doors, light pipes, and buttons. The chassis assembly of each product integrates a plastic base, metal chassis, spring clip, dress panel, and SCSI signal cable (attached with screws by the vendor). Thus, final assembly parts involved in the manufacturing of the box include only the chassis assembly, internal power cable, printed circuit board, plunger rod, HP-PAC, disk mechanism, and top cover. Like the workstation, there are no fasteners for manufacturing or the customer to deal with, and the top cover snaps into place to retain all parts inside.

Low Cost for Entry-Level Pricing

To command lower material costs for mechanical components, all custom plastic and sheet-metal parts were hard-tooled for mass production. The chassis of each product was designed with a minimum of folded features to reduce part complexity and the cost associated with that complexity. All major sheet-metal parts use progressive tooling for the lowest price.

To reduce the amount of final assembly time (and labor costs) involved in the product, components were designed with a high degree of functional integration. Integrated components (such as chassis or top cover assemblies) are assembled by vendors, placing the burden of labor on these non-HP processes and thus achieving lower pricing of the final product. This functional integration of components also lowers cost by reducing part count and related inventory management.

Because of the no-fastener design, final assembly takes under four minutes for the workstation product and comparable times are achieved for the external peripherals. This ease of manufacturing lowers manufacturing costs because of reduced assembly time and overhead costs. It also makes the product much better suited to indirect market channels, which prefer to configure products themselves and often do this at the last possible moment before shipment.

Environmentally Friendly

The Model 712 workstation and peripherals also conform to HP's new guidelines for product stewardship. Virtually every component of the workstation and peripheral products can be easily disassembled, identified, and recycled. Each plastic part contains engraved information that identifies the type of plastic used, and only four different types of plastic are used within the entire family of products. To assist the disassembly process, the products use plastic heat staking to join parts together, which can easily be cut away during the disassembly process. The new HP-PAC material can be recycled as well, either by grinding to pellet size and reusing in other shipping cushion parts, or by melting the material

down to solid plastic. And again, because there are virtually no fasteners to deal with, disassembly is quick and thus more parts are given to recycling. Materials with bromide compositions have been avoided, except for the HP-PAC parts, which require a bromide flame-retardant treatment to meet safety requirements.

Other product stewardship features include:

- No painted components (all plastics with molded colors)
- No plated plastics
- No adhesives
- Required labels can be recycled along with plastic base material
- Reusable aftermarket components (flexible and hard disk, power supply, CPU, and fan)
- Bulk packaging of final assembly components implemented on larger parts (reduces manufacturing waste)
- Printed circuit boards built in approved non-ODS (ozone-depleting substance) processes
- Embedded fan (low acoustic noise).

Reference

1. J. Mahn, et al, "HP-PAC: A New Chassis and Housing Concept for Electronic Equipment," *Hewlett-Packard Journal,* Vol. 45, no. 4, August 1994, pp. 23-28.

Development of a Low-Cost, High-Performance, Multiuser Business Server System

Using leveraged technology, an aggressive system team, and clearly emphasized priorities, several versions of low-end multiuser systems were developed in record time while dramatically improving the product's availability to customers.

by Dennis A. Bowers, Gerard M. Enkerlin, and Karen L. Murillo

The HP 9000 Series 800 Models E25, E35, E45, and E55 (Ex5) and the HP 3000 Series 908, 918, 928, and 938 (9x8) business servers were developed as low-cost, performance-enhanced replacements for the HP 9000 F Series and low-end G Series and the HP 3000 Series 917, 927, 937, and 947. The development of the PA-RISC PA 7100LC processor chip and the LASI (LAN/SCSI) I/O interface and the evolution of DRAMs for main memory enabled the development of these low-end servers. The PA 7100LC and the LASI I/O interface are described in the articles on pages 12 and 36 respectively.

The priorities for the Models Ex5 and Series 9x8 server project were short time to market, low cost, and improved performance. The functionality and quality of the new servers were to be as good as the products they were replacing, if not better. The challenge was to get these new servers to market as soon as possible so that HP could continue to be competitive in the business server market and our customers could benefit from better performance at a lower price. We were able to get the first versions of these systems completed, released, and shipping on time with all new VLSI components.

Low-Cost, Higher-Performance Features

The principlal reason for achieving high integration and low cost for the Model Ex5 and Series 9x8 servers was the development of the PA 7100LC processor chip, which was being developed at the same time as our servers. Integrating the floating-point unit, the 1K bytes of internal instruction cache, the external cache interface, the TLB (translation lookaside buffer), the memory controller, and the general system connect (GSC) I/O interface inside the PA 7100LC processor chip allowed the Model Ex5 and Series 9x8 designers to condense the CPU and main memory onto the same board.

Also, at the same time as our new servers were being developed, DRAM densities doubled (in some cases quadrupled) to allow more memory to be put into a smaller space. The Model Ex5 and Series 9x8 servers use the same industry-standard ECC (error correction coded) SIMM modules used in the HP 9000 Model 712 and other HP workstations. The Model Ex5 and Series 9x8 servers use 16M- and 32M-byte SIMMS which must be inserted in pairs to provide 32M to 256M bytes of main memory. ECC memory was chosen because it carries two additional address lines making it possible to put four times the memory capacity on one SIMM while staying compatible with industry-standard modules. The 64M-byte SIMM was designed several months after first introduction of the new low-end servers to boost their maximum memory to 512M bytes. This larger SIMM is not available as an industry standard.

Four versions of the Model Ex5 and Series 9x8 processor have been developed, differentiated by clock speed, cache size, and cost. Each version is fully contained on the system board (which also contains cache, main memory, processor dependent hardware and firmware, and 802.3 LAN connect) and is easily installable and upgradable. Table I lists the technical specifications for the different Model Ex5 systems and summarizes the HP-UX* performance characterizations. The Series 9x8 MPE/iX systems have equivalent CPU hardware, and their specifications are close to those given in Table I.

Table I
Technical Specifications for HP 9000 Model Ex5 Systems Running the HP-UX Operating System

Processor Performance	Models			
	E25	E35	E45	E55
Clock (MHz)	48	64	80	96
SPECint92	44	65	80	104
SPECfp92	66	98	120	156
OLTP Transactions/s	80	125	155	180
Standard memory/cache (M bytes)	16	16	16	16
Maximum memory (M bytes)	512	512	512	512
Cache size (K bytes)	64	256	256	1000
Cache SRAM speed (ns)	15	12	10	7.5

Architecture

Fig. 1 shows a block diagram for the Model Ex5 and Series 9x8 servers.

The general system connect (GSC) bus was designed as a new, more powerful system bus for higher performance. The Model Ex5 and Series 9x8 servers only use the GSC bus for the processor, main memory, and 802.3 LAN through the LASI chip. The midrange and high-end server systems also support the GSC bus as their high-performance I/O bus. All PA-RISC systems support the HP-PB[1] (HP precision bus) as the common I/O bus because multiple functionality (hardware and drivers) currently exist for this bus. The interface from the GSC bus to the HP-PB is accomplished in a chip called the HP-PB bus converter.

The HP-PB bus converter chip is a performance-improved version of the bus converter that was used in the HP 9000 F and G Series and HP 3000 Series 9x7 machines. This chip allows the Model Ex5 and Series 9x8 servers to leverage HP-PB I/O functionality from the systems they are replacing.

The HP-PB bus converter implements transaction buffering† as an HP-PB slave, gaining performance improvements of 10% to 28% over its predecessor. The chip supports GSC to HP-PB clock ratios ranging from 3:1 to 5:1 in synchronous mode when the GSC bus is operating under 32 MHz. It switches to asynchronous mode when the GSC bus operates in the 32-to-40-MHz range. These ratios and the asynchronous feature of the HP-PB bus converter allow fair flexibility in CPU and GSC operating frequencies while maintaining a constant 8-MHz HP-PB frequency. The bus converter also provides an interface to the access port used for remote support, and the control signals used for the chassis display and status registers. The chip is designed for the HP CMOS26B process and comes packaged in a 208-pin MQFP (metal quad flat pack).

The other key VLSI chip used in the I/O structure for the Model Ex5 and Series 9x8 servers is the LASI chip. The LASI

† With transaction buffering, during reads from disk, data is buffered so that HP-PB transactions can continue at maximum pace.

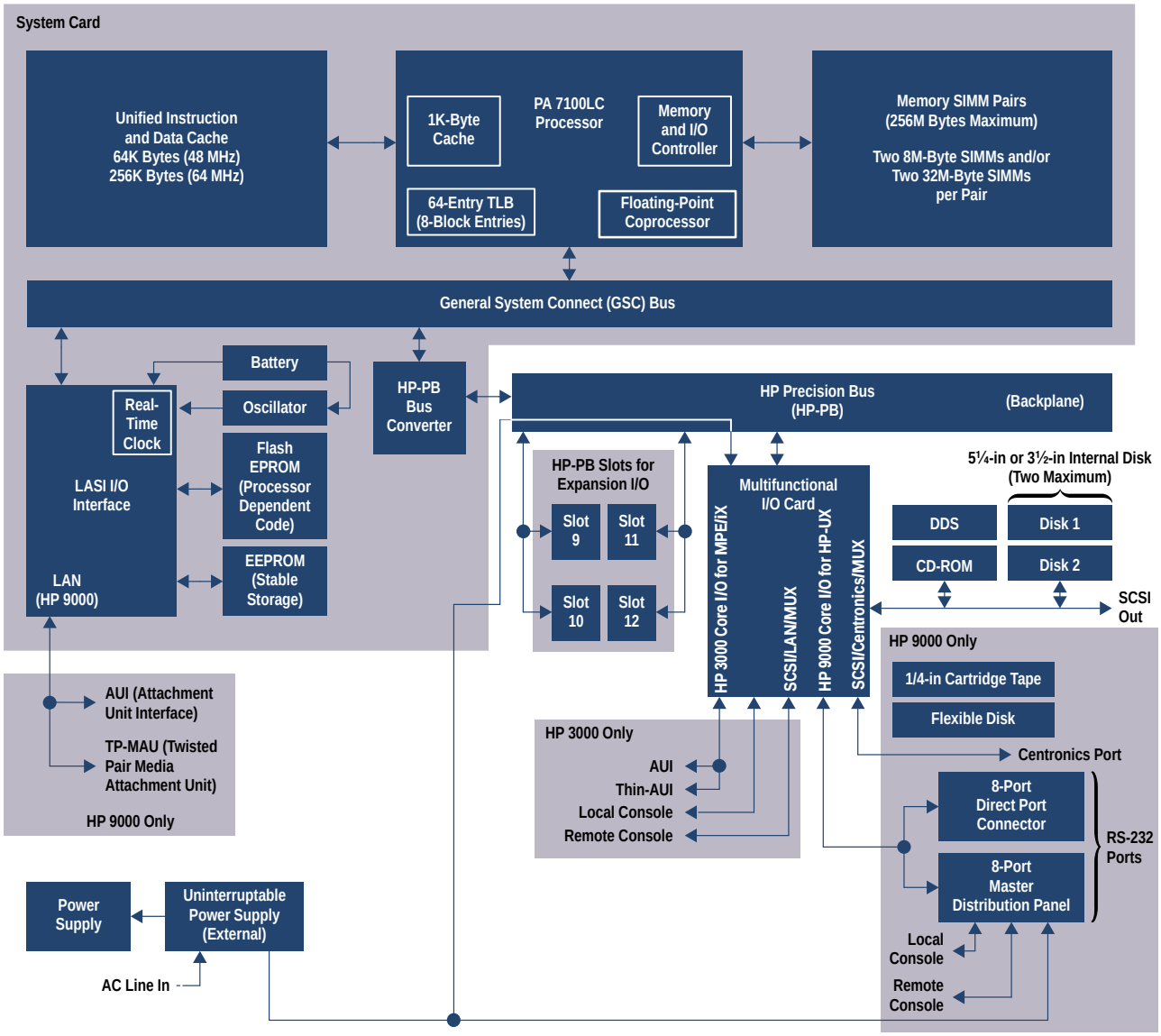


Fig. 1. Block diagram for the HP 9000 Series 800 Models Ex5 and the HP 3000 Series 9x8 business servers.

chip is designed to have the same integration impact on core I/O as the PA 7100LC had on the CPU and GSC bus interface. The workstation products are able to take advantage of this (see article on page 6), but the multiuser server systems were not able to take advantage of LASI functionality.

LASI functionality includes interfaces to IEEE 802.3 LAN, SCSI, processor dependent code, Centronics, RS-232, audio, keyboard, flexible disk, and GSC bus arbitration logic and the real-time clock. Because HP-UX and MPE/iX software drivers could not be made available in time for our release, only a small subset of LASI functionality could be used on the new servers. Thus, the decision was made to continue using the core I/O card from the previous versions of low-end servers because it provides all the functionality needed.

For the 96-MHz version of the Model Ex5 and Series 9x8 servers, a chip with a subset of the functionality of LASI was used. This was developed as a cost reduction for those applications that use only the LAN, GSC bus arbitration, and processor dependent code path. The 96-MHz version had to add a real-time clock on the system board to have equivalent functionality to what was needed from LASI.

In addition to the above VLSI chips and printed circuit boards, the Model Ex5 and Series 9x8 servers have the internal capacity for two disk drives (4G bytes), two removable media devices, and up to four I/O slots. The packaging and power supplies for the new servers are highly leveraged from the previous low-end server systems.

Meeting Fast Time-to-Market Goals

Meeting deadlines for any program is always a challenge. Too often it is believed that a few extra hours a week is all that is needed to keep the project on track. But many well-intentioned programs soon lose time with unexpected delays even when the project team is made up of industrious folks willing to do whatever it takes to stay on schedule.

At large corporations like HP, where releasing a product to market may span several divisions, the task is even more daunting. With our lab's mission of providing world-class low-end commercial business systems and servers, time to market is always expected to be a key objective. In the case of the Model Ex5 and Series 9x8 program, it was the primary objective. Additionally, we were challenged to keep cost projections in line with the set goals, and to meet or exceed the quality of the versions of the low-end servers that we were replacing. Quality is consistently a key objective on all HP products.

The main challenge for the Model Ex5 and Series 9x8 program was to achieve (on schedule) an order fulfillment cycle time† of 10 or fewer days for the entire product family. With the existing product family averaging order fulfillment cycle times four to five times larger than our 10 or fewer days goal, it was evident that for the new servers a well-orchestrated program that involved the entire system team was necessary to meet this challenge.

Fig. 2 shows a spider chart of the overall metrics for the Model Ex5 and Series 9x8 program. Note that the program achieved or exceeded all planned manufacturing release goals. Even the factory cost goal was exceeded, which was at risk when the hardware team added existing material into the design over less expensive functionality, reducing the software development schedule. The order fulfillment cycle time objective was not only achieved but exceeded! For the first three months of production, order fulfillment cycle time averaged under nine days.

† Order fulfillment cycle time is measured from when HP receives a customer's order to the time when the order is delivered at the customer's dock.

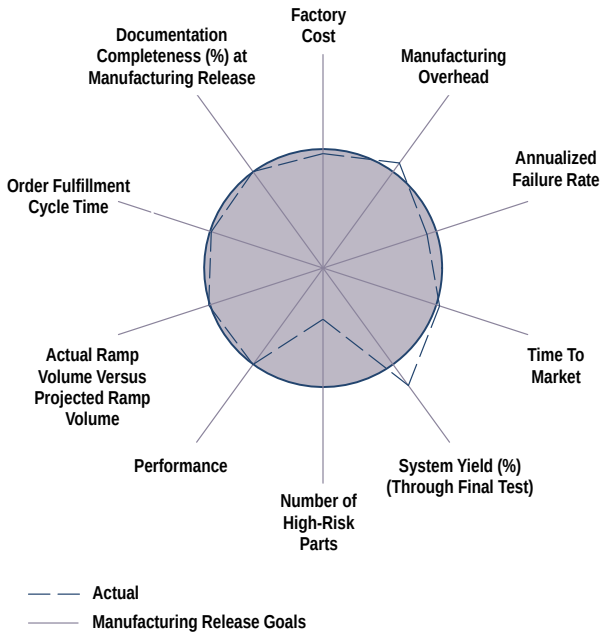


Fig. 2. Spider diagram showing how well the project team for the Models Ex5 and Series 9x8 servers met their project goals.

The following sections summarize the reasons we met or exceeded our cost, quality, time-to-market, and manufacturing release goals.

Consolidation of Project Team. When the Model Ex5 and Series 9x8 program was in its early stages of design the development team was dispersed in two different geographic locations. The remote organization was eliminated and the project development and management were consolidated in one location under one manager. With this organization, technical decisions regarding system requirements could be made quickly and effectively.

Ownership of Issues. A system team composed of representatives from the different organizations involved in the development of the Models Ex5 and Series 9x8 servers was organized. Weekly one-hour meetings were held with the main focus on issues or concerns that impacted the project schedule. Communication was expected to be limited to discussions that affected everyone. Issues were captured and assigned an owner with a date assigned for resolution of the issue. Representatives at the meeting were expected to own the issues that were presented to their organization. No issue was closed until the team agreed upon it. This ensured that technical problems did not "bounce" around looking for an owner.

Interdivisional Communication. Effective interdivisional teams establish good working relationships to ensure timely response to actions and issues. An example was the decision to change the core I/O functionality. While the hardware team improved their factory cost by incorporating new, less costly hardware, the software team would have realized a

longer schedule to provide new software features to support the new hardware. After reviewing the plans, the hardware team, aware of the critical time-to-market objective, recommended a return to the existing I/O feature implementation at an impact to factory cost for the sake of the software development team's ability to improve their schedule. The end result was that the hardware team still achieved their factory cost goals (by making adjustments elsewhere), and the software development team achieved their schedule goals.

Leverage Design Where Possible. When time to market was established as the key objective for the project, the development teams realized that leveraging from as many existing products as possible would greatly benefit achieving this goal. The following components were leveraged from new or existing products:

- Product package. Sheet metal was leveraged from the existing low-end business servers with minor changes to accommodate new peripherals and a different processor and main memory partitioning scheme. Plastic changes were kept to a minimum in an effort to use tools already established. (Only one new tool was required.)
- Base system configuration. The base system was established using the I/O printed circuit boards and several peripherals available on the existing low-end servers.
- Memory. The memory design was leveraged from the memory configuration used in the HP 9000 Model 712 workstation, which uses SIMM modules for the base memory system. Higher-density memory was designed specifically for the Model Ex5 and Series 9x8 servers after first release to increase their maximum memory capacity.
- Power supply. The power supply was leveraged from the existing servers.
- Printed circuit boards. The core I/O boards from the existing servers were used with only minor firmware changes to the HP-UX version. The processor board and backplane were new designs based on ideas shared with the Model 712 development team.
- VLSI. The PA 7100LC processor chip and the LASI core I/O chip were leveraged from the Model 712 workstation, which was being designed at the same time as our server systems.
- Firmware. Some of the firmware and I/O dependent code was codeveloped with the Model 712 development team.

Fast Time to Manufacturing Release. The use of concurrent engineering played a key role in reducing the back-end schedule. The back end of the schedule consists largely of manufacturing activities (including final test and qualification) aimed at achieving a release of the product for volume shipment. In the case of the Model Ex5 and Series 9x8 servers, with the individual boards being built in two geographically different manufacturing facilities, it was imperative that communication between these entities receive ample attention.

To facilitate this communication, a coordination team consisting of new product introduction engineers and new product buyers and logistics people were located in close proximity with the R&D development team. Everyone attended the system team meetings, which were led by the hardware lab, to ensure that the most current information was applied to the overall system schedule. In addition, production build meetings were held before, during, and after each prototype run to discuss build results. Ensuring that all manufacturing personnel realized that these systems were engineering prototypes, with a high potential for problems, was a difficult task. Most people were not used to seeing lab prototypes being built in a production process. Since the line was shared with currently shipping products, it was extremely important to ensure that building the prototypes did not impede shipping other products.

Prototype Management. Two operating system environments were required for the new servers, the HP-UX operating system release 9.04 and the MPE/iX operating system version 4.0. Since these environments were under development at the same time as our products, it was essential that hardware prototypes be delivered efficiently and be of sufficient quality to ensure expedient use by the software development groups. Thus, three key objectives were considered essential by the development groups. First, units had to be of the highest quality. Second, delivery of the units had to be on time. Finally, downtime because of hardware problems had to be minimized.

To accomplish the first goal, all prototypes were built using the entire production process. No prototypes were handcrafted in the lab. This ensured that units were built with the same quality standards as are applied to released systems. Additionally, each customer was assured of receiving the latest revision of materials released to production. Even new parts not covered under manufacturing release criteria were guaranteed to be of the same revision level. All revision levels were tracked on each unit for the life of the project.

For the second objective, a customer priority list was generated based on customer orders and needs. After the orders were submitted to the manufacturing systems, build priorities were set based on the critical needs being supplied first. From functional prototypes to production prototypes, upgrade kits were structured and made available. In cases where a new system was not required, customers had the option of moving immediately to an upgrade. Also, performance upgrades were designed to require a swap of the processor card only.

Tracking the revision level of all hardware was essential to achieving the third objective of minimizing downtime because of hardware. Another key point was being able to react to a customer's problem quickly. We used a prerelease support team at another HP division to ensure timely response. Spare material was purchased by the support team and defective parts were returned to the lab for analysis.

Using all these methods, we were able to achieve the goal of having all operational prototype units upgraded to manufacturing release equivalence before manufacturing release. This guaranteed test partners use of the machines for future development without the "not-quite-final-product" concerns.

We were not without our share of problems in terms of effectively managing the prototypes. For instance, several units were placed inside an environmental test chamber for weekend testing. During the early morning hours on a

Sunday, the temperature controller of the chamber went out of control, ramping the temperature to beyond 70 °C. The additional heat caused the fire sprinkler system in the chamber to turn on, flooding the chamber at a rate estimated at 10 gallons per minute. The units were standing in four feet of water, but with the disk drives external to the chamber, the test continued. When the chamber was finally shut down, the water mopped up, and the results checked, it was discovered that two of the seven units,which were on the top rack, out of the standing water, continued to operate without failure throughout the test. This test was affectionately named the "bathtub test."

Time-to-Market Focus. Establishing the time to market as the key objective for the program was not enough to ensure its success. The teams involved required constant reminders to stay focused on this objective and make trade-offs accordingly. Once the schedule was confirmed and accepted, it was important to acknowledge the progress. Any activities that appeared in danger of jeopardizing the schedule were reviewed and tackled accordingly.

However, the project team realized that in the past changes to system requirements had a big impact on meeting project schedules. Changes to system requirements to modify or include a feature that might improve sales or could be easily implemented at the cost of another metric might result in significant changes to the hardware or operating system design. In the case of the Model Ex5 and Series 9x8 servers, the system team implemented a process that was also used by the software development teams to control design changes. This process is called *change control,* which requires the change requester to provide a specific level of information to determine whether a particular change is viable. While this is not a new idea, the Model Ex5 and Series 9x8 development team elected to make one additional rule change. Each change request submitted would be briefly looked at to determine how the change would affect the base system. In other words, we wanted to ensure that a change was critical enough that it needed to be added to the products planned for the first release.

The hardware system team put on hold all change requests that were determined not to be required for the first release. To avoid causing lots of changes to the software after first release, some of the critical enhancements that were considered crucial to future sales were briefly reviewed and included in the initial software release. In some cases this meant no changes were required after the first software release. However, there were some instances of patches required for full functionality.

Customer Order Fulfillment Cycle Time

For the Model Ex5 and Series 9x8 servers to stay competitive, cost and performance were not the only items that played an important role. During 1993, it was clear that HP had an order fulfillment cycle time problem, which of course made our customers unhappy and affected our competitiveness. A task force was formed to address HP's order fulfillment cycle time problems. We found out that results from this task force would not arrive in time to help us with our new products. Thus, we formed a team seven months before introduction to ensure that the reduced order fulfillment cycle time process for the Model Ex5 and Series 9x8 servers was in place when the products were ready to be shipped to customers.

Our goal was to reduce the time between the receipt of a customer purchase order for a system and the time when the system is delivered to the customer site. We wanted to reduce this time by 75% of what it was for our existing servers. To accomplish this goal, the following changes were made before product introduction:

- The product structure was made much simpler and it includes fewer line items.
- Product offerings to distributors were unbundled.
- Product numbering for distributors' orders had a single SKU (stock keeping unit) for ease of ordering.
- The rules for our factory configuration system and field configuration system were mirrored.
- Early and proactive material stocking was performed before introduction to ensure that plenty of material was on hand to meet customer demand immediately.
- Factory acknowledgments were automated for clean orders.
- Intensive training was given to order processing personnel in the field and the factory about the Model Ex5 and Series 9x8 servers two months before introduction.
- Consignment, demonstration, and distributor units were stocked before introduction.
- More capacity was added to the factory, and assembly processes were streamlined.
- All new processes were tested intensively before introduction.

With these steps we were able to meet and exceed our order fulfillment goal.

Conclusion

The real success of the Model Ex5 and Series 9x8 server program was that the goals for fast time to market and reduced order fulfillment cycle time were achieved. These were major accomplishments considering the events that took place throughout the whole project including the development of a major VLSI component, consolidation of the design team from different divisions and locations, communication between different manufacturing entities, and a stream of last-minute catastrophes such as flooding prototypes in the environmental test ovens and several eleventh-hour VLSI bugs that had to be fixed.

Acknowledgments

Although one of our key success factors was the small dedicated team, many key people and teams contributed to the development of the Model Ex5 and Series 9x8 low-end servers. We would like to acknowledge the team in Germany (both lab and manufacturing) led by Bernd Buggerman that began and nurtured the initial idea for the new servers. They brought forth many of the design and process improvements that made the products successful. Once the project moved to Roseville, the assembled design team took hold quickly and stepped up to the challenge. We would like to thank John Spofford for his tenacity and Scott Stallard and John Adelsbach for their support. Karen Murillo would like to thank her lab team and especially Loren Koehler, Matthias Meier, and Don Zimmer for providing material for this article. In addition to these three, Robert Dobbs, Delfina Mooney, Ken Tang, Linda VanderWegan, Darrell Walker, and Charles Zacky were key to the success of the the Model Ex5 and Series 9x8 servers' timely implementation, as were John

Connolly and his team who developed the HP-PB bus converter chip. We would also like to thank the teams responsible for the PA 7100LC and LASI chips, including Craig Simpson, Tom Meyer, and Tom Spencer. Dennis Bowers would like to thank the members of his system team, Wolfgang Boehm, Hank Cureton, Mark Driscoll, Gerard Enkerlin, Phil Knutson, Barb McGillivary, Lisa Mitchell, Marty Nichols, Craig Nunes, Kevin Smith, and Dave Snow (and their teams) as they were consistent links to our program partners and allowed us to push harder on our goals than anyone thought achievable. A special thanks goes to Elaine Calomeni for keeping our prototypes up and hardware flowing to and from our remote testing partners.

Thanks also go to the order fulfillment team of Carol Talley, Mauricio Cori, Mary Marcus, Cheryl Marshman, Doug Anderson, John Duff, Tom Hack, Jordan Levin, Wil Pomeroy, Ray Barrett, Dawn Williams, Don Leigh, Steve Wehr, Oanh Phuong, Joe Barrington, Marty Rothenberg, Bob Littlejohn, Kyle Christensen, Bill Schaefer, Laura McMullen, and many other individuals who helped make this program successful.

References

1. T. Alexander, et al, "Corporate Business Servers: An Alternative to Mainframes for Business Computing," *Hewlett-Packard Journal,* Vol. 45, no. 3, pp. 8-30.
2. *Ibid*, p. 17.

HP Distributed Smalltalk: A Tool for Developing Distributed Applications

An easy-to-use object-oriented development environment is provided that facilitates the rapid development and deployment of multiuser, enterprise-wide distributed applications.

by Eileen Keremitsis and Ian J. Fuller

HP Distributed Smalltalk is an integrated set of frameworks that provides an advanced object-oriented environment for rapid development and deployment of multiuser, enterprise-wide distributed applications. Introduced in early 1993, and now in its fourth major release, HP Distributed Smalltalk leverages the ParcPlace Smalltalk language and the VisualWorks development environment. Together, HP Distributed Smalltalk and VisualWorks enable rapid prototyping, development, and deployment of CORBA-compliant applications.†

In the global marketplace, corporate information technology needs are increasingly demanding because worldwide competition requires geographically dispersed operations, changing markets require agility to remain competitive, pressure to improve return on investment requires strong cost controls, timely access to complete information is crucial for business success, and finally, corporate users require access to both legacy and newly developed information sources and applications.

HP Distributed Smalltalk helps answer these business needs by supporting:

- Easy on-demand access to information and services across the enterprise
- Dynamic interaction of distributed people and resources
- Greater application flexibility and ease of use
- Insulation from differences in operating environments
- An architecture that supports an evolutionary approach including legacy system integration
- Industry standards that will allow application interoperability across languages, high productivity, and code reuse.

Customers can take advantage of HP Distributed Smalltalk's easy-to-use development environment to create distributed solutions to compete effectively in the global marketplace. For example, with HP Distributed Smalltalk, customers might build on the sample Forum application (described later) so that their geographically dispersed users can simultaneously annotate a shared document. Also, customers might use HP Distributed Smalltalk to create three-tiered database access applications that extend the advantages of existing client-server architectures for better isolation between user interfaces, data manipulation models, and legacy and new data.

† CORBA, or Common Object Request Broker Architecture, defines a mechanism that enables objects to make and receive requests and responses. HP Distributed Smalltalk's implementation of this architecture is described later in this article.

Three-tiered applications are the most efficient and scalable form of software design for building complex applications. They carefully separate the user interface (tier one) from the business rules governing the application (tier two) and the persistent storage for the information in a database (tier three). Each tier can reside on a different machine in a network, making best use of the network resources. HP Distributed Smalltalk contains objects that enable the straightforward construction of these applications.

Using HP Distributed Smalltalk

An application written in HP Distributed Smalltalk is able to respond to service requests from remote systems. Remote entities that request services of an application do not have to be written in HP Distributed Smalltalk as long as they are in a system that implements the standard ORB (object request broker) and common object services from the Object Management Group (OMG). See "Object Management Group" on page 86 for a description of these items.

In many cases an HP Distributed Smalltalk application's component objects are distributed across several systems. These distributed objects can interact seamlessly so that end users are unaware of where the objects are located.

An overview of the process of running an HP Distributed Smalltalk application is shown in Fig. 1. For incoming requests to the service provider, the ORB translates requests from the implementation-neutral Interface Definition Language (IDL) to the local language (ParcPlace Smalltalk) and forwards them to the correct local object for processing. To complete the request, the service provider's ORB takes return values, translates them to IDL and forwards them to the remote ORB from which the request was received.

Not only does HP Distributed Smalltalk support distributed application delivery but it also provides an environment for distributed application development, which includes:

- A complete implementation of the Object Management Group's latest standards
- A rich suite of tools for application development and administration including simulated remote test support, a remote debugger, and an IDL interface browser and generator
- A user interface environment and sample applications that developers can reuse or extend, or simply use to become familiar with the system.

Object Management Group

The Object Management Group, or OMG, is a nonprofit international corporation made up of a team of dedicated computer industry professionals from different corporations working on the development of industry guidelines and object management specifications to provide a common framework for distributed application development.

OMG publishes industry guidelines for commercially available object-oriented systems, focusing on areas of remote object network access, encapsulation of existing applications, and object database interfaces. By encouraging industrywide adoption of these guidelines, OMG fosters the development of software tools that support open architecture, enabling multivendor systems to work together.

To define the framework for fulfilling its mission, in 1992 OMG published its Object Management Architecture Guide. This guide provides a foundation for the development of detailed interfaces that will connect to the elemental components of the architecture. Fig. 1 shows the four main components of this architecture:

- The object request broker (ORB) enables objects to make and receive requests and responses in a distributed object-oriented environment.
- Object services is a collection of services with object interfaces that provide basic functions for creating and maintaining objects.
- Common facilities is a collection of classes and objects that provide general-purpose capabilities useful in many applications.
- Application objects are specific to particular end-user applications.

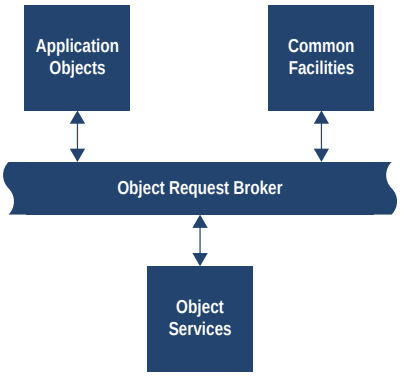


Fig. 1. The object management architecture.

The application objects, object services, and common facilities represent groupings of objects that can send and receive messages. The software components in each of these primary components have application programming interfaces that permit their participation in any computing environment that is based on an object technology framework.

In addition, because HP Distributed Smalltalk is an extension of VisualWorks, developers are able to do their programming in a language they already know (ParcPlace Smalltalk) using the VisualWorks application builder.

VisualWorks is an implementation of the Smalltalk programming language and environment. It provides an excellent environment for building standalone and simple client/server applications that are 100% portable between many of the major computing platforms and operating systems. HP saw an opportunity to enhance the capabilities of VisualWorks to be the basis for next-generation applications by adding objects that enable VisualWorks systems to communicate directly using a standardized set of communications facilities.

Framework

The HP Distributed Smalltalk framework is an environment that encompasses everything from communication with other systems through database access to the object-oriented ParcPlace Smalltalk language and a rich suite of developer's tools, all seamlessly integrated to facilitate distributed application development.

The major components of HP Distributed Smalltalk are shown in Fig. 2 and briefly defined below:

- HP Distributed Smalltalk ORB. This is a full implementation of the Object Management Group's Common Object Request Broker Architecture (CORBA).
- Remote Procedure Call (RPC) communication. This component supports efficient and reliable transfer of messages between systems.
- HP Distributed Smalltalk object services. This includes all standard object services required by distributed systems, as well as support for creating and maintaining objects and the relationships between them.

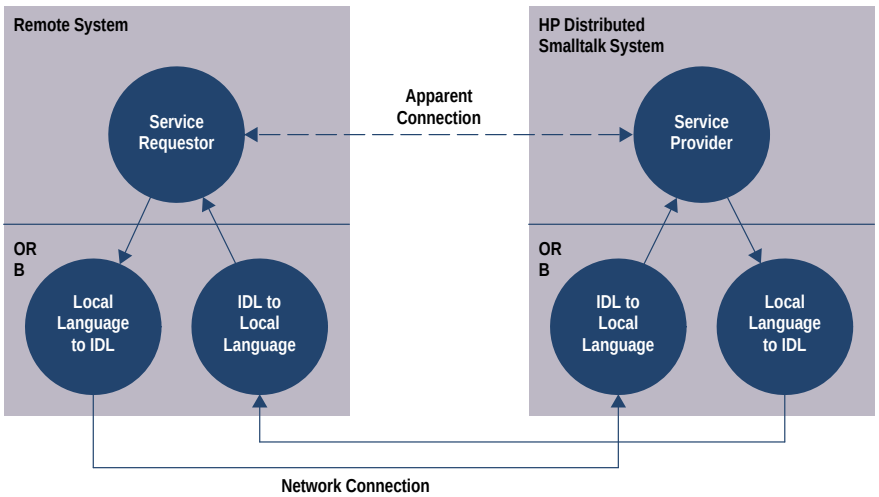


Fig. 1. Overview of the HP Distributed Smalltalk process.

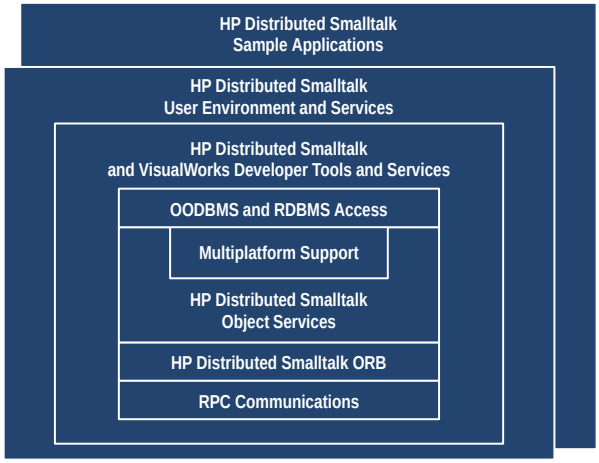


Fig. 2. The major components of HP Distributed Smalltalk.

- Multiplatform support. HP Distributed Smalltalk applications that run on one platform (hardware and operating system combination) can run, without porting, on any other supported platform.
- OODBMS and RDBMS access. HP Distributed Smalltalk provides database access directly to HP's Odapter† and Servio's GemStone as well as to Sybase and Oracle (via VisualWorks). HP Odapter can be used to provide access to a variety of other database systems.
- HP Distributed Smalltalk developer tools and services. This level of the framework provides support specifically designed for developing, testing, tuning, and delivering distributed applications. HP Distributed Smalltalk incorporates a rich development environment, application builder support, and the ParcPlace Smalltalk language.
- HP Distributed Smalltalk user environment and services. These services include a reusable demonstration user interface and desktop environment support for users' work sessions and normal desktop activity.
- HP Distributed Smalltalk sample application objects. These objects provide developers with example code that can be reused or extended, or can provide a source of ideas for developing alternate applications.

The following sections provide more detailed descriptions of the components that make up HP Distributed Smalltalk.

HP Distributed Smalltalk Object Request Broker

HP Distributed Smalltalk is a complete implementation of CORBA, the Object Management Group's specification of an object request broker. HP Distributed Smalltalk's compliance provides the basis for object and application interoperability.

CORBA specifies core services that are required of an object request broker to support interoperable distributed computing. The CORBA specification includes the following core services.

Interface Definition Language Compiler. OMG has defined the Interface Definition Language, or IDL, to be independent of other programming languages. Interfaces for objects that can provide distributed services are written in IDL so that they are accessible to service requesters that might be written in Smalltalk, C, C++, or another language.

OMG recently approved the IDL-to-Smalltalk language binding proposed by HP and IBM. This is important because it allows users to build distributed systems using multiple languages where appropriate, allowing a Smalltalk object to be able to request services of a C++ object or vice versa.

Interface Repository. This service provides a registry of distributable object interfaces for a given system. Any object that remote objects can access has an interface in the interface repository. For example, when objects on two or more systems at different locations collaborate in an application, they interact by sending messages to their interfaces. Since external clients have access to an object's services only through the object's interface, the implementation of the object is private. This privacy provides a variety of benefits, including security, language independence, and freedom to modify the implementation of how a service is performed without external repercussions.

HP Distributed Smalltalk ORB Support. The object request broker (ORB) is the key to providing support for distributed objects. By providing an ORB on each system, HP Distributed Smalltalk makes the location of any object transparent to clients requesting services from the object.

When a message is sent to a local object, the activity is handled normally. When a message is sent to a remote object, the remote object's local surrogate (created automatically by the ORB) intercepts the message, then uses the ORB to locate the remote object and communicate with it (see Fig. 3). Results returned to the calling object appear exactly the same, whether the message went to a local or remote object.

An ORB's responsibilities include:

- Marshalling and unmarshalling messages (translating objects to and from byte streams for network transmission)
- Locating objects in other images or systems
- Routing messages between surrogates and the objects they represent.

While a request is active, both client and server ORBs exchange packet information to track the course of the request

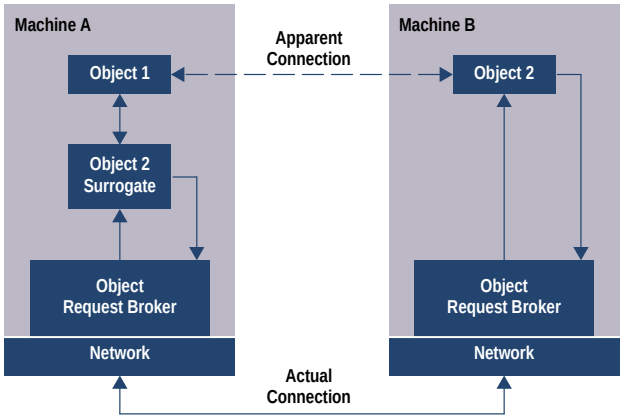


Fig. 3. HP Distributed Smalltalk handles remote access so that a request to a remote object appears the same as a request to a local object.

† HP Odapter is a complementary product from Hewlett-Packard that provides an efficient and scalable link between objects implemented in an object-oriented language such as Smalltalk or C++ and the entities in an Oracle relational database.

and resolve any network or transmission errors that might occur.

Object Services and Policies

Object services extend the core ORB services to support more advanced object interaction. HP Distributed Smalltalk implements OMG's Common Object Services Specification (COSS), which extends CORBA to provide protocols for common operations like creating objects, exporting and destroying objects (life cycle), locating objects (naming), and asynchronous event notification. Additional object services and policies provide efficient interaction between finer-grained distributable objects.

Naming.† There is a standard for assigning each object a unique user-visible name. Names are used to identify and locate both local and remote objects.

Event Notification.† This is a service that allows objects to notify each other of an interesting occurrence using an agreed protocol and set of objects.

Basic† and Compound Life Cycle. There are standard ways for objects to implement activities such as create and initialize, delete, copy, and move both simple and compound objects, externalize (prepare for transmission to remote systems), and internalize (accept objects transmitted from remote systems). Compound objects, built from simple objects, can include application components, anything that appears on a user's desktop (such as a document, a mail handler, or a graphics toolbox), complete applications, and so on.

Relationships: Containment and Links. Links allow networked relationships among objects. Objects can be linked together with various levels of referential integrity (determining how to handle situations when one of the parties to the link is deleted), and in one-to-one, one-to-many, and many-to-many relationships.

Together with links, containment establishes and maintains relationships between objects. Each object has a specific location within some container. Containers are related hierarchically. HP Distributed Smalltalk provides objects that implement a generic distributed container. Programmers can use these objects to build specific implementations such as an electronic mail envelope (containing components of a message) or a bill of sale (containing information about items in a shipment) with minimal extra programming.

Properties and Property Management. Properties are part of an object's external interface (owner, creation date, modification date, version, access control list, and so on). They are a dynamic version of attributes.

Application Objects and their Assistants. Application objects are relatively large-grained compound objects that end users deal with (e.g., a file folder or an order entry form). Application assistants are lightweight objects that implement most of the policies and participate in most of the services that desktop objects need to participate in. Application assistants function as the developer's ambassador into the object services. Application assistants can be stored and activated efficiently and provide the basis for future transaction support.

† This service is specified in COSS 1.0.

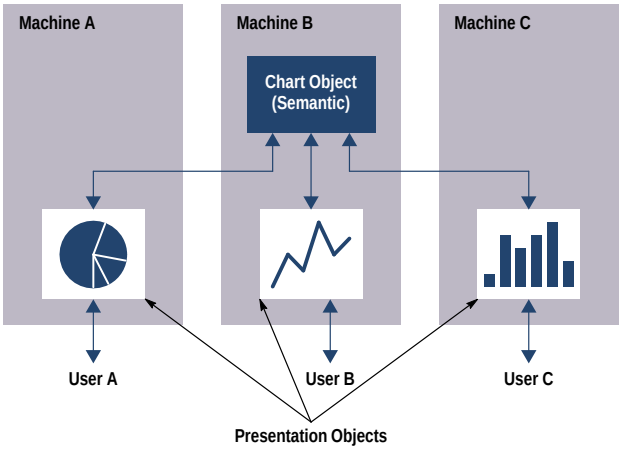


Fig. 4. The bulk of user interaction is with local presentation objects, minimizing and condensing the need to propagate semantically relevant changes over the network. Here for example, a user might choose to look at a chart (semantic object) as a pie, line, or bar chart presentation object.

Presentation/Semantic Split. A logical split between distributed objects, the presentation/semantic split provides an efficient architecture for distributed applications. Local presentation objects handle the bulk of user interaction, while a semantic object (which can be anywhere on the network) holds a shared persistent state of the object (see Fig. 4).

By using the presentation/semantic split, the designer can choose what part of the application should be shared and what should be unique to each user. Applications that might use the presentation/semantic split include a team whiteboard where all behavior is shared but each user can write comments, or a common document with pages that are unique to each user so that all users can read at their own pace. A variety of sample applications included with HP Distributed Smalltalk provide illustrations of how to use the presentation/semantic split.

While use of the presentation/semantic split is optional, it facilitates and optimizes distributed application development and execution. Advantages of using the presentation/semantic split include:

- Acceptable performance levels even over wide area networks
- Association of a single semantic object with multiple presentation objects, a critical feature in distributed computing environments where it is common for many users to work with the same application
- Application access independent of local windowing systems
- Better code reusability.

The HP Software Solution Broker described on page 93 is a good example of using the presentation/semantic split in an application.

Developer Services

HP Distributed Smalltalk also extends VisualWorks with services that support development and test of distributed applications.

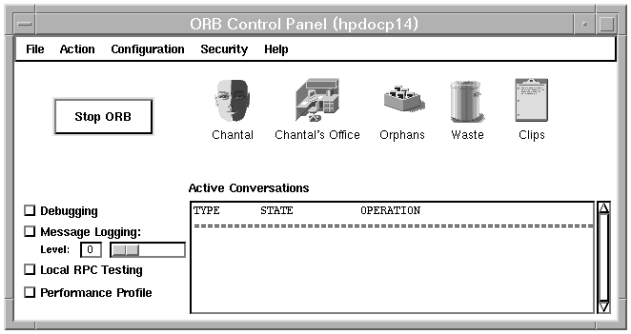


Fig. 5. The control panel provides an easy-to-use interface to administrative and developer services.

Control Panel. The technical user interface to HP Distributed Smalltalk for administrators and developers is invaluable for testing and maintenance (Fig. 5). The control panel provides:

- Controls to start and stop the system cleanly
- Support for local RPC testing (simulated distribution)
- Tracing facilities to log network conversations between objects
- Performance monitoring.

Interface Repository Browser and Editor. The interface repository browser provides an iconic view of the contents of the interface repository where publicly available interfaces are specified (see Fig. 6). It is organized hierarchically so that developers can explore and edit interfaces and construct requests to use the interfaces.

Shared Interface Repository. In HP Distributed Smalltalk, users can share an interface repository on a remote system so they do not have the overhead of keeping a copy of all of the interfaces on every system. The product also supports version management of interfaces, which is very important in large-scale, evolving distributed systems.

Remote Context Inspector and Debugger. This service is an extension that allows debugging on remote images when appropriate. It supports object inspection and debugging for

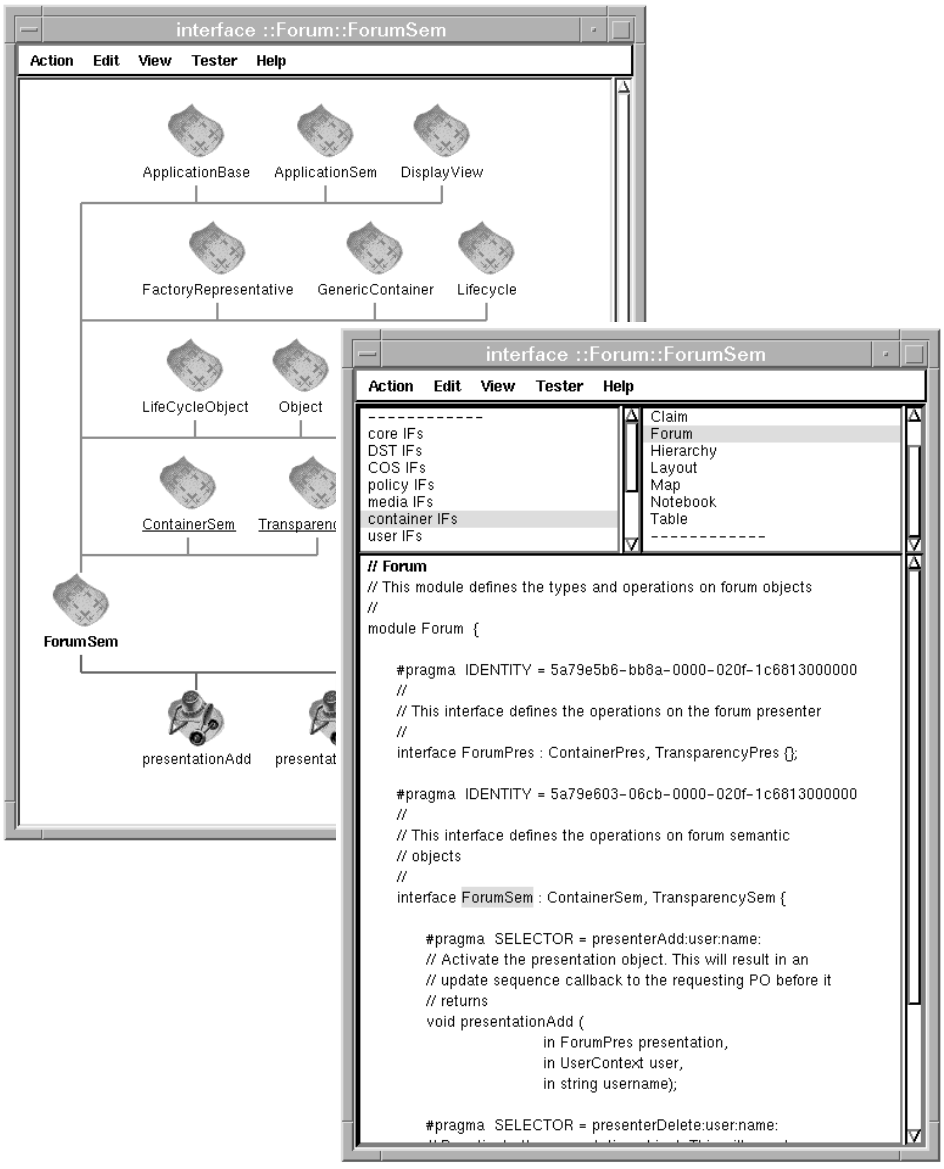


Fig. 6. The interface repository browser can be used to view or edit interfaces that remote clients can use to call local objects.

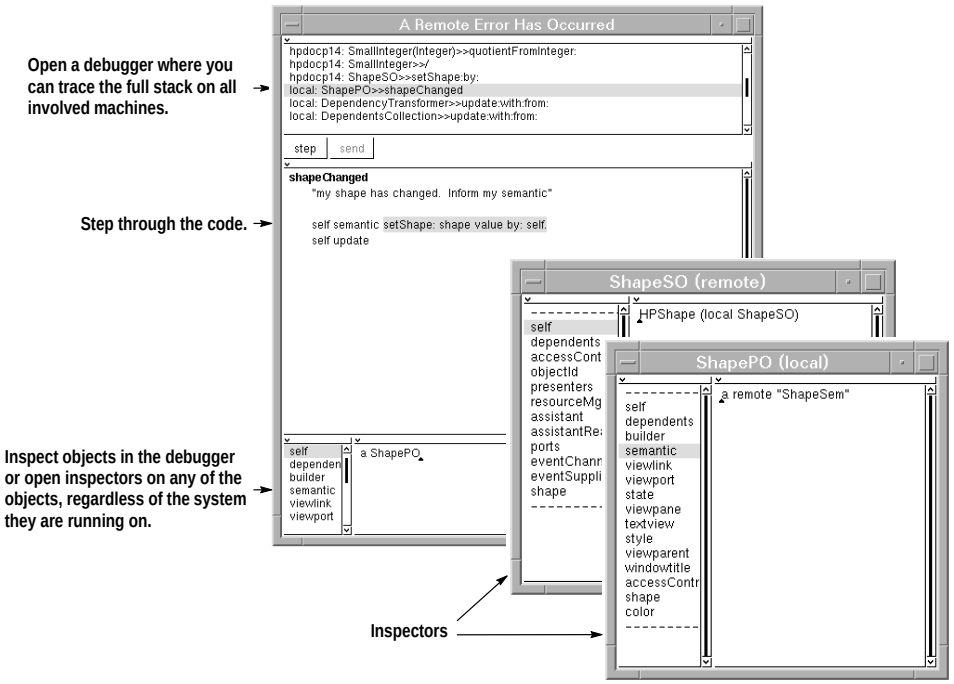


Fig. 7. Screens associated with a remote debugger.

the entire distributed execution context, including communication between images. Fig. 7 shows using the debugger to step through code and inspect objects that might be located anywhere in a distributed environment.

Stripping Tool. To prepare an application for delivery, developers use the HP Distributed Smalltalk stripping tool to remove unneeded classes and interfaces and seal source code when application development is complete. The stripping tool's user interface suggests likely items for removal (see Fig. 8).

User Services

User services allow developers to build a desktop or office environment and control activities during a session.

System Objects. HP Distributed Smalltalk supports a variety of system objects: user, session, clipboard, wastebasket, and orphanage.

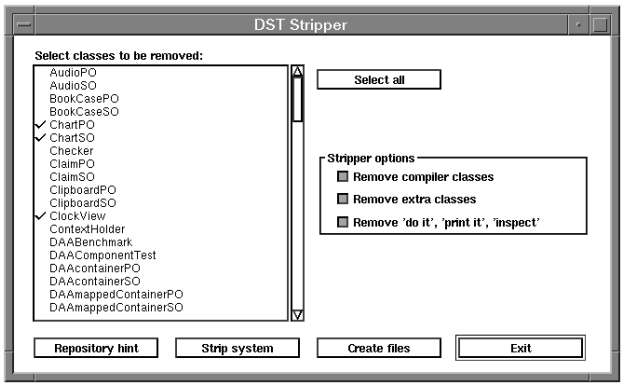


Fig. 8. Interface for the HP Distributed Smalltalk stripping tool.

- User object. This object contains information held about end users of the system including who they are, how to contact them, and so on. User objects may be included by reference in other objects. For example, a user might include a business card in a memo that would enable the receiver to get in touch with the sender.
- Session object. All the information required about the state of a user's environment, including user login, preferences, layout, and so on are contained in a session object. The session object also supports the notion of workspaces, with the potential for developing richer workspace environments. It has no icon on the desktop but it interacts with and supports other application objects.
- Clipboard. This is a container for objects that are being cut, moved, or copied from one location to another.
- Wastebasket. This container receives objects that users throw away. The wastebasket can be cleared when it gets too full.
- Orphanage. This is a container for holding objects that are no longer needed.

Security. Developers can use or extend HP Distributed Smalltalk's access control services in the applications they build, setting controls for host systems, users, or both. Host-system access control lets developers determine whether an image can receive messages from another system. User-level access control lets a developer determine whether a given user has any one of several kinds of privileges (e.g., read or write privilege) for a given object.

Developers can administer access control programmatically or from the default user interface.

Example Code

While all HP Distributed Smalltalk code is available to read, reuse, or extend, the default user interface and certain sample applications may be the best place to start.

Fig. 9. The screen for presenting the office metaphor and some typical objects in an office.

User Interface. HP Distributed Smalltalk uses and provides support for a user interface based on an office metaphor which is designed for easy use and understanding. In the default user interface, all the objects a user works with locally (folders, file cabinets, documents, and so on) are contained in an office. All offices on the same system are in the same building. Users can navigate between buildings to access objects in other offices. Fig. 9 shows a typical office and some of the objects available in an office.

Sample Applications. Sample applications illustrate the use of distributed objects. For example, the Forum (Fig. 10) provides a shared window in which several users can view and annotate a picture or document. The Notebook is a place to store both local and remote objects on a desktop.

Users can also build their own objects from any of the simple objects available, including a table, chart, input field, picture, and text window (see Fig. 11). The sample applications can be extended and customized to create a variety of simple distributed applications.

Creating Applications

HP Distributed Smalltalk allows VisualWorks programmers to create distributed applications quickly and easily. Building on the benefits of Smalltalk and VisualWorks, HP Distributed Smalltalk users can build CORBA-compliant applications either from scratch or by modifying existing applications. Like any Smalltalk application, the distributed development process is iterative and designed for dynamic refinement.

Development. Distributed application development is a four-step process.

1. Design and test the application objects locally.

2. Define the object interfaces and register them in the interface repository.

3. Use HP Distributed Smalltalk's simulated remote testing tools (which actually use the ORB to marshall and unmarshall object requests) to verify the interfaces specified in the interface repository.

4. Track messages and tune performance.

Distribution. Once an application is developed, tested, and tuned locally, it is easy to set it up for distributed use.

5. Copy the application classes to the Smalltalk images they will run on.

6. Update the interface repositories in these images.

The application can then run in the fully distributed environment without further change. Except for actual packet transfer, the distributed application is identical to the simulated remote application developed, tuned, and tested during development.

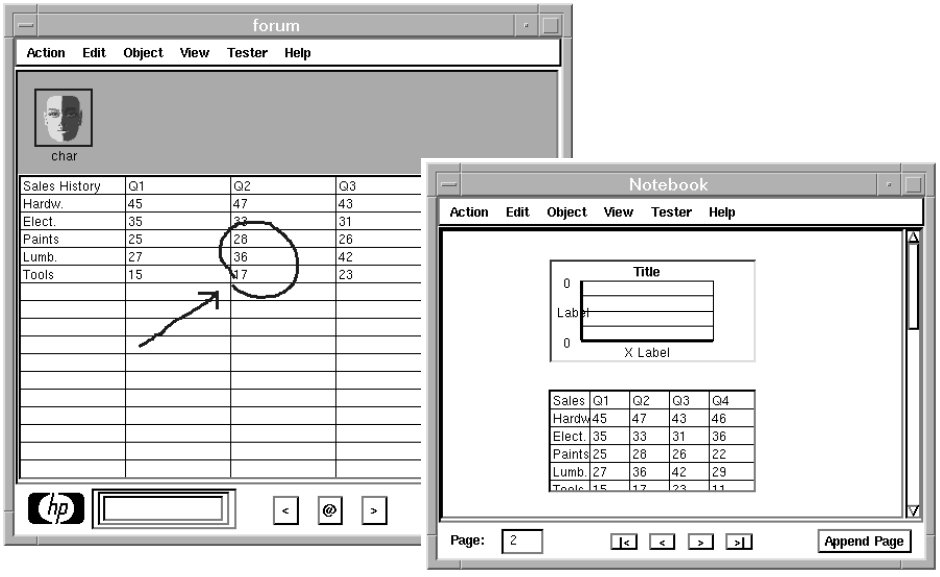


Fig. 10. User interfaces to the sample applications Forum and Notebook.

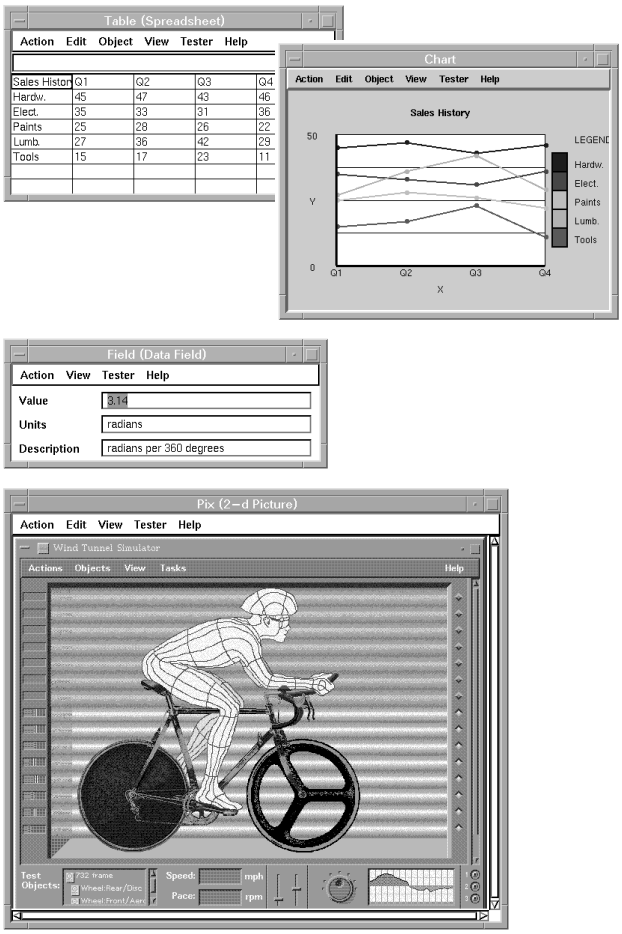


Fig. 11. Sample objects provided with HP Distributed Smalltalk.

Delivery. Once the application is tested, developers can deliver it to their users by stripping the environment of unneeded objects and tools. Once stripped, the application looks exactly the same as applications developed in other languages and can be executed on any supported platform, including: HP-UX,* SunOS/Solaris, IBM AIX, Microsoft® Windows, Microsoft Windows NT, or IBM OS/2. Support for these platforms is available under a run-time license from Hewlett-Packard.

Acknowledgements

We would like to thank the program team manager for HP Distributed Smalltalk, William Woo, for his unflagging support and encouragement. Dr. Jeff Eastman deserves immense credit for designing and building the first implementation of the project. We would also like to thank the HP Distributed Smalltalk program team in Cupertino, California and Fort Collins, Colorado for all their contributions: Lynn Abbott, Jerry Boortz, Jim Borchert, Kevin Chesney, Jürgen Failenschmid, Warren Greving, Robert Larson, Lynn Rowley, Terry Rudd, and Brent Wilkins.

A Software Solution Broker for Technical Consultants

A distributed client-server system gives HP's worldwide technical consultants easy access to the latest HP and non-HP software products and tools for customer demonstrations and prototyping.

by Manny Yousefi, Adel Ghoneimy, and Wulf Rehder

On a typical working day an HP consultant, one of thousands worldwide, sits down with a customer to solve a business problem. The challenge, the customer may tell the consultant, is to move sales data to headquarters more quickly so that management can make timely strategic decisions. For a solution, the consultant might propose a decision support system that integrates the customer's older legacy system where the sales data has been stored traditionally with a faster "warehouse" database and easy access tools that present the information in just the form needed, right on the customer's desktop. "Let me show you what I mean," the consultant says, turning on a laptop computer (which had previously been connected to a LAN or telephone socket). Navigating through the windows on the screen, the consultant invites the customer to look through a virtual shelf filled with databases and access tools, all represented by icons, together with middleware and application development toolkits (see page 98). The consultant clicks on an icon and the tool becomes immediately available for browsing or for self-paced learning. From here the consultant may show one of the demos that are included, or navigate the customer through a hypertext document to more information, alternate products, additional options, and prefabricated software building blocks. No wonder that this virtual software laboratory is called by HP consultants, "the software sandbox." This consultant is actually building—from the tool and product portfolio in front of them—a prototypical decision support system for this customer. How much of this is fantasy and how much reality?

The answer is that it is all reality now. The software sandbox that the consultant was starting to "play in" is called the HP Software Solution Broker (or Broker, for short) and is available now to HP consultants. Defining and creating a decision support system is, of course, not play but serious work. However, the ease and immediacy of the Broker, the ample choices, and many helpful hints make even urgent business problem solving an experimental sport. Best of all, the consultant receives these products and tools, together with support and on-line documentation, free of charge. For this convenience, substantial research efforts had to be poured into building such a virtual software depot, using HP's own hardware platform and the most advanced object technology. Before explaining this implementation more systematically, it is useful to watch our technical consultant and the customer at work.

Using the Software Solution Broker

To get a feeling for how the Software Solution Broker is used we will briefly watch the technical consultant show the customer how to build a prototypical decision support system.

After clicking on the icon in the ORB control panel, which starts the object request broker (an action that in effect opens the lid covering the sandbox), the consultant activates the Software Solution Broker icon. Another window opens offering the Broker's classification of products, either by vendor, by technology, or by product name. (Alternative paths into the Software Solution Broker, such as a classification by business problem, are under development.) Choosing the **i**(nformation request) button for technology, the consultant asks whether the customer wants to see database information first or options for the user interface. As an executive, the customer is eager to see or build a nice GUI. Clicking on the graphic user interface **i** button brings up several choices of which three are shown in Fig. 1. Having heard about VisualWorks the customer selects it and is presented with the VisualWorks Showcase.

The consultant then shows a VisualWorks demonstration to explore with the customer what kind of data display windows, control buttons, analysis tools, and other features would be appropriate. After jotting down these initial requirements the consultant is ready to build a first prototype. The help button launches a palette of GUI building tools, and it takes only minutes to draw an example of a transaction entry tool for the transactions underlying the decision support system the customer wants built (see Fig. 2). Here the customer interrupts and requests that the data be shown in spreadsheet form as well as graphically. They agree on bar chart and pie chart presentations for a first cut and proceed to discuss the requirements for the underlying database. The Software Solution Broker has a "virtual shelf" of relational databases that work with VisualWorks, and among these the customer may have a favorite system, or an already installed legacy database. They again discuss the pros and cons while viewing various product demonstrations.

We meet the customer and the consultant again after another hour or so. By then the VisualWorks front-end tool displays some real data pulled from a database (Fig. 3). At this point we leave the executive's office and describe how the Software Solution Broker is constructed.

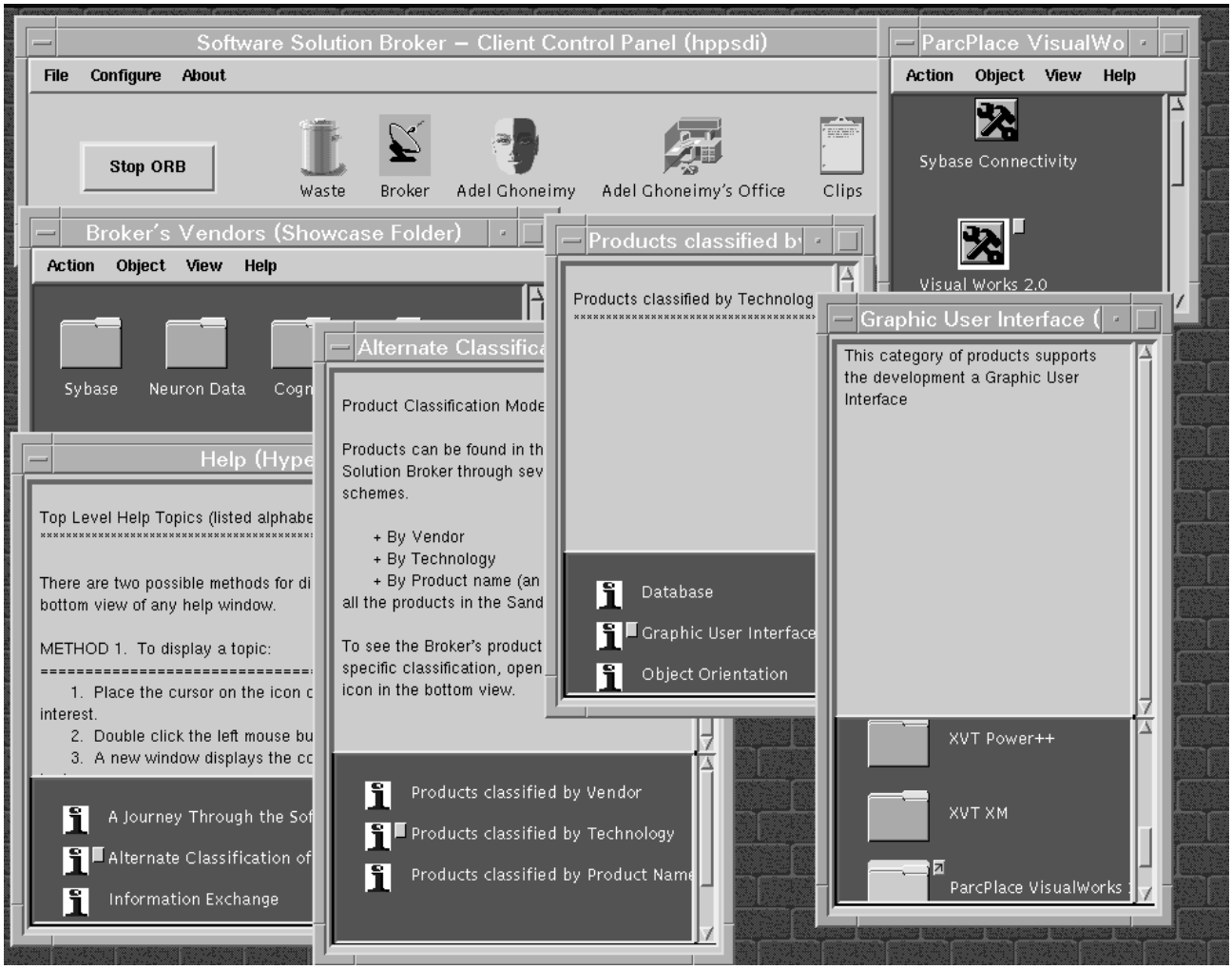


Fig. 1. Software Solution Broker user interface.

Constructing the Software Solution Broker

Two considerations determined the architecture and consequently the implementation of the Software Solution Broker. First, since the products on the Broker have to be accessible worldwide but will be updated and maintained locally, the global partitioning between distributed users and a central server functionality called for a client/server implementation on a wide area network (WAN). Secondly, the need to accommodate many different types of clients and to be able to encapsulate many different products in the software server strongly suggested vendor independence (openness) and adherence to certain industry standards such as the Common Object Request Broker Architecture (CORBA).

Software Substrate

Here we will not focus on the WAN implementation but instead will concentrate on the software substrate on which the Software Solution Broker is built. In the software substrate (see Fig. 4) we include the entire software kit composed of server and client development tools, tools for building the client/server interaction components of the system, and repository tools. Repository tools are essential for the construction of a depot that contains the information in the system, including the logic for accessing this information. After a careful technical analysis of five alternative complete substrate kits, VisualWorks from ParcPlace Systems was chosen as the development software for the PC, UNIX® client, and UNIX server, while HP's Distributed Smalltalk (see article, page 85), which also works with VisualWorks, was the tool of choice to build and manage the client/server interaction. All system information (e.g., documentation) at this time of writing (release 2.0) still resides with the products and a central repository has not yet been chosen. Tools such as Object Lens (working with VisualWorks) or HP Odapter make relational databases look like object databases, so we know that the selection of a repository can be made very quickly when needed.

VisualWorks was the easy winner because it provides a complete environment for the development of true graphic applications that run unchanged on UNIX-system-based, PC, and Macintosh computers under their native windowing systems. Three of VisualWorks' features made it especially appropriate for the Software Solution Broker:

- VisualWorks is built on Smalltalk, a pure object-oriented language designed for fast modular design.
- VisualWorks possesses a tested set of development tools, including browsers for object classes, a thread-safe debugger, and a change manager to track modifications to the code, as well as an inspector for use in testing.

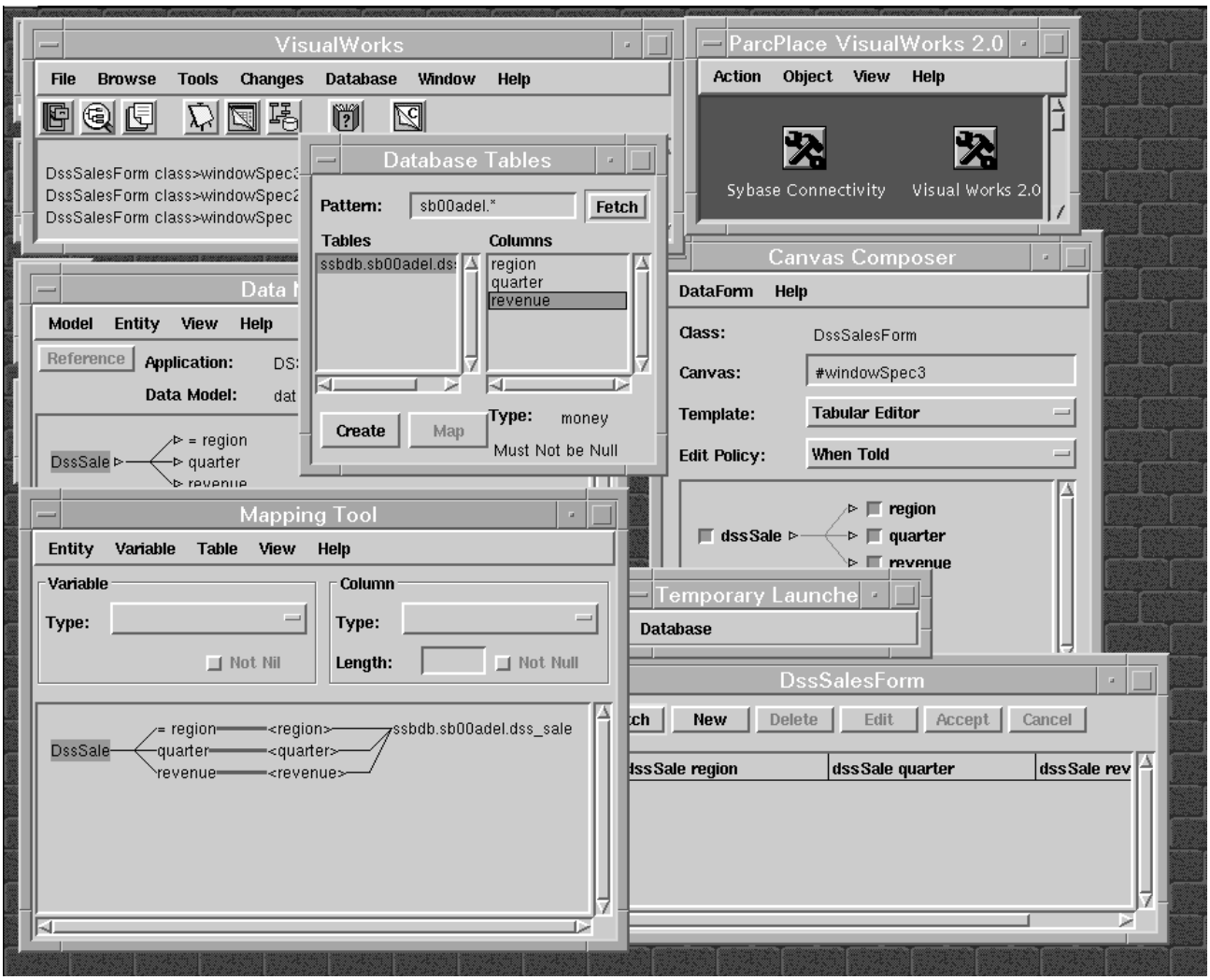


Fig. 2. A window within the Software Solution Broker showing VisualWorks tools for prototyping a customer application.

- VisualWorks has a large class library of more than 350 types of portable objects. These include a rich user interface development toolkit suitable for all major windowing systems.

  HP Distributed Smalltalk extends VisualWorks' capability for developing standalone systems into an environment for creating distributed object systems (see Fig. 5) by adding the following:
- A full implementation of the Object Management Group (OMG) Common Object Request Broker Architecture (CORBA) core services
- Common Object Services for life cycle operations such as creating objects and the relationships between them
- Sample application objects, for example for the modular partitioning of client/server functionality into semantic and presentation objects.

  These objects and services for building distributed applications are portable to all platforms supported by VisualWorks. Furthermore, they are compatible with the OMG CORBA standards. HP's Distributed Smalltalk provides seamless support of client/server interactions between VisualWorks images. CORBA compliance makes our Software Solution Broker implementation open and capable of interoperability, for instance with C++ CORBA-compliant applications, and as soon as HP Distributed Smalltalk is OSF DCE-compliant,* also with DCE remote procedure calls (RPCs). For the current release, TCP/IP or HP Sockets are being used.

* OSF DCE is the Open Software Foundation's Distributed Computing Environment.

Product Encapsulation

Everybody who has worked with spreadsheets, word processors, or CAD systems knows that similar or identical functionality does not mean that the user interfaces and more generally the visual, iconic, and mental models are comparable. For the Software Solution Broker, too, each product has its own artifacts and idiosyncrasies, its own look and personality by which we can identify it when we see it in use or on the shelf of a vendor. This unavoidable fact poses challenges for the "virtual shelf" of the Broker. Without wanting to blot out the individuality of a vendor's offering it was the objective of the development team to minimize the effort needed for the user to get accustomed to this diversity. Generally speaking, the variety has to be hidden behind a simple and consistent, product independent mode of access with uniform and intuitive graphical symbolism. A particular example is the double click used consistently to launch an application.

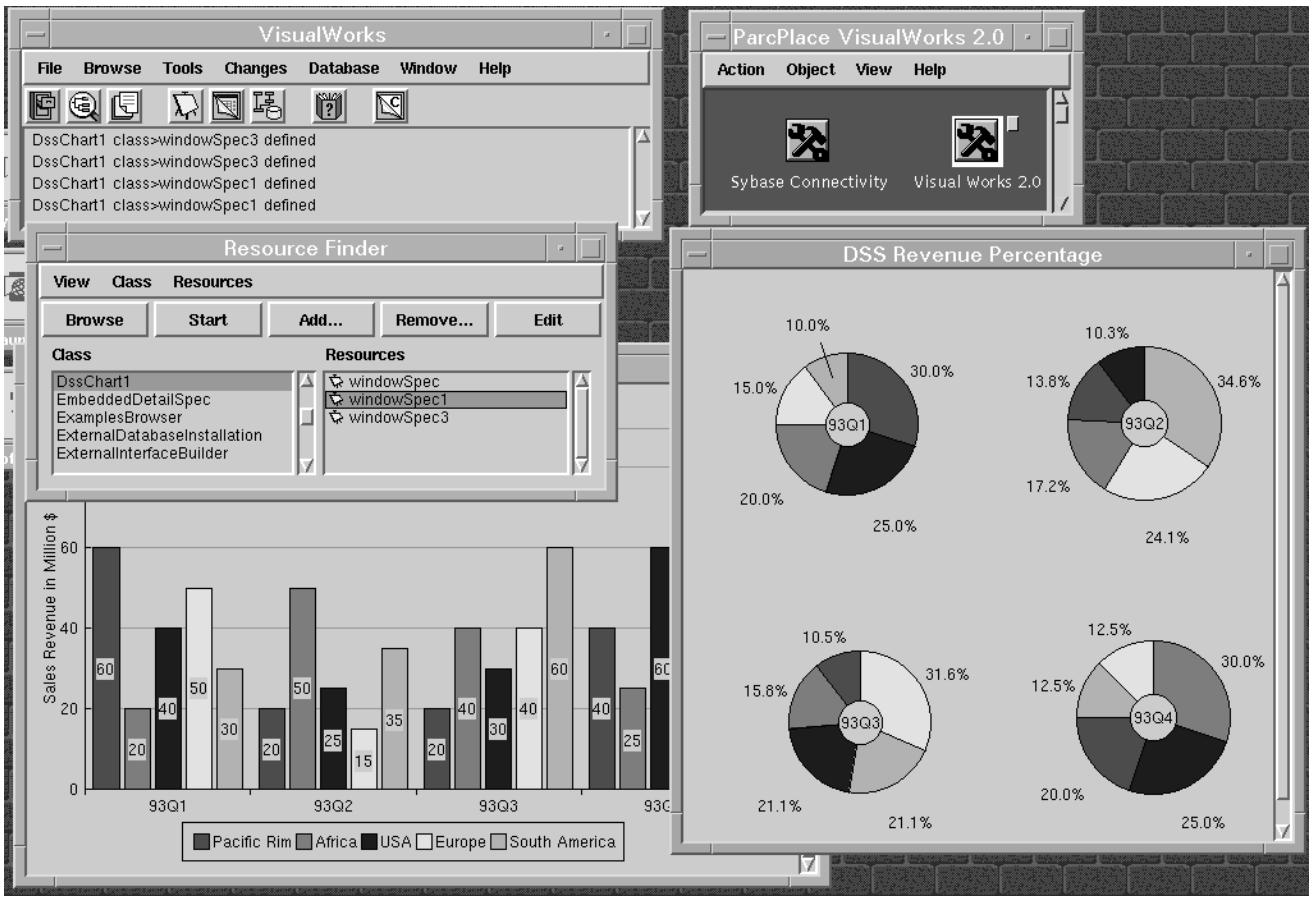


Fig. 3. Prototype display for a customer application constructed using the Software Solution Broker to select user interface and database software.

Encapsulation, in the context of the Software Solution Broker, describes a body of activities and software mechanisms that have two purposes: to integrate each product within the overall product portfolio so that the consultant can use it in its native mode, and to provide a uniform way to access the products, their associated tools, and other artifacts. This accessibility, it should be noted, is restricted to the features and artifacts that are relevant to consulting work with the customer. This means the consultant can access editors, executable code, and documentation, but isn't able to change the internal product configuration, the way it is stored and administered in folders, or the source code. Because of the intrinsic symmetry between Software Solution Broker servers and clients (see Fig. 4) the encapsulation can be done either on the server side or on the client side, provided the

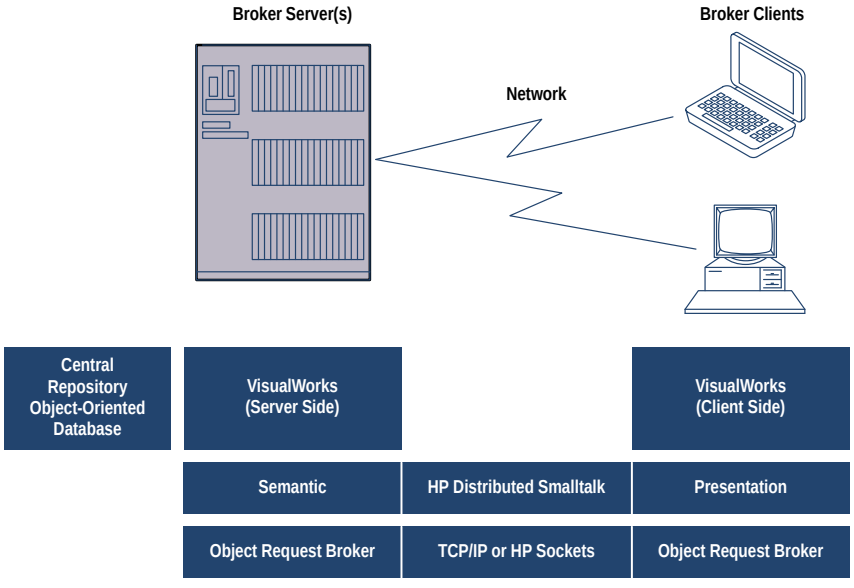


Fig. 4. Software Solution Broker software substrate, showing the client/server architecture, the user interface engine (VisualWorks), and the client/server framework (HP Distributed Smalltalk).

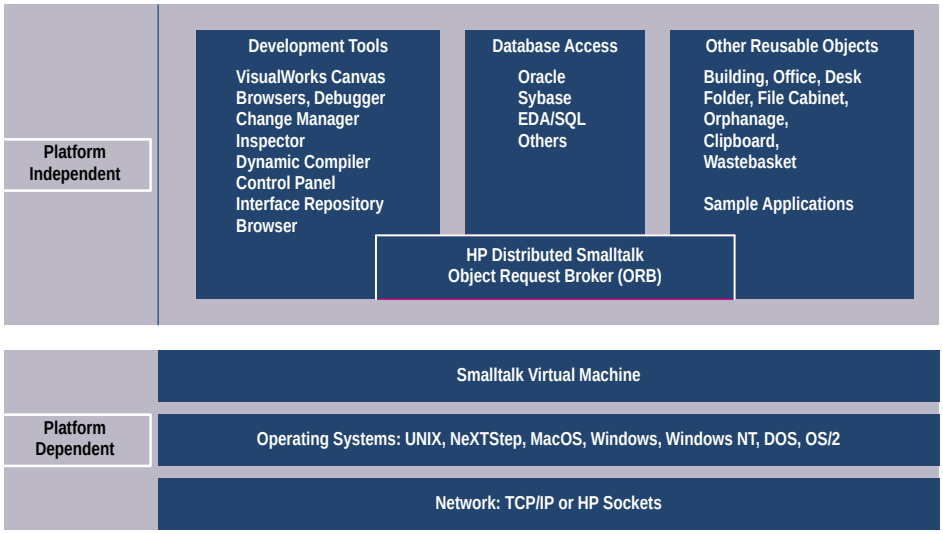


Fig. 5. HP Distributed Smalltalk and VisualWorks provide a full development and run-time environment for distributed computing.

classes FolderPlusPO and EncapsulationDialog are present in the client. These two classes will be discussed below.

It is the already mentioned semantic/presentation split, together with object-oriented features such as *inheritance* and *polymorphism* that make the encapsulation effortless. The semantic/presentation object distribution model is HP Distributed Smalltalk's implementation of a distributed client/server architecture. In this model, classes always appear in logical pairs, one representing the server semantics, the other their presentation in the client. Consequently, the class instances or objects also come in pairs. Take for instance the window object. Every window is composed of two logical parts: its shared (semantic) properties such as its rectangular shape, and its local and personal (presentation) attributes such as color. In general, a semantic object often has (and controls) many different presentation objects, which in the case of the Software Solution Broker handle the remote user interactions, thus reducing network traffic. For instance, one semantic data display object creates and controls different presentations of the data as a bar chart and a pie chart in a decision support system. HP Distributed Smalltalk allows various modes of collaboration between the semantic and presentation objects, including messages that are handled by the object request broker. (For a simple but complete example see the *HP Distributed Smalltalk User's Guide*, chapter 10.)

After this abstract introduction of HP Distributed Smalltalk's semantic/presentation split architecture we will describe in more concrete terms how it works for the encapsulation procedure. As stated above, encapsulation must achieve two goals: it has to present a graphical representation of the artifact (product, tools, demos, documentation) in its native mode to the remote client, and it must allow the remote user to launch the artifact at the server side through this representation. HP Distributed Smalltalk has a pair of classes, MediaSO and MediaPO, that accomplish exactly this. (The suffixes SO and PO imply that semantic and presentation objects, respectively, are spawned by these classes). Tracing the interaction diagram between two objects of these classes we found that there exists a ready-made method called updatePresenter, visible in the MediaSO class, that creates the remote presentation object of a product or other artifact in the server. To customize the generic MediaSO and MediaPO classes and the method updatePresenter for the encapsulation of specific artifacts we first created the narrower subclasses ArtifactSO and ArtifactPO. Then we augmented ArtifactSO with the attributes of artifacts such as vendor and product names. Finally, using overloading, we extended the method updatePresenter to include, among several other administrative tasks, the crucial behavior required for launching the artifacts while exporting their display to the client platform.

Concurrent with this architectural design of the classes and methods that bring about encapsulation in the Software Solution Broker, a few product dependent steps must also be taken. This is done at the instance or object level of every concrete artifact (such as a product) so that it will behave in its expected, native mode. This is a simple matter of inserting the right environment variables and parameters in an encapsulation dialog window. The required information can easily be gleaned from the installation manual of the particular product that is being encapsulated. Finally, products, tools, and other components are put into folders and the encapsulation is done.

Use of Object Technology

The design and building of the Software Solution Broker were characterized by a short development time, a minimal amount of new coding, and a high degree of reuse. The major reason is the application of object-oriented technology. The object-oriented use is pervasive throughout the design, as indicated above, but it is helpful to point to specific examples. We'll give two examples for the object-oriented features inheritance and polymorphism in the context of encapsulation.

One of the examples has just been described: the subclass ArtifactSO of the class MediaSO inherited the method updatePresenter, which in turn, through the feature of polymorphism, was overloaded (that is, extended to include additional functional behavior).

The encapsulation dialog window provides another example. As an administrative tool, it is not available to the user. It is an object built from a subclass of the existing HP Distributed Smalltalk class called SimpleDialog. From this class, the window inherits characteristics such as its property to pop up in front of other windows (it's not obscured), its basic layout

HP Software Solution Broker Accessible Products

Vendors

- Cognos Corp.
- ParcPlace System
- XVT Software
- Itasca System
- Informix
- Neuron Data
- Sybase
- Unison Software
- ProtoSoft
- Oracle
- Dynasty
- NetLabs

Tools

HP-UX*

- Cognos Corp.
  - PowerHouse 4GL 7.23
- ParcPlace System
  - VisualWorks 2.0
  - VisualWorks with Sybase connectivity
- XVT Software
  - XVT-Design (C Developer Kits)
  - XVT/XM (C Developer Kits)
  - XVT-Power++
  - XVT/XM (C ++ Developer Kits)
  - XVT-PowerObject Pak I
- Itasca System
  - ODBMS Server
  - Developer Tool Suite
  - C Interface
  - Lisp Interface
  - API Libraries (C++, CLOS, Ada)
- Informix
  - Informix Online R4GL
  - Informix WingZ
  - Informix SE R4GL
  - Informix SE ISQL
  - Informix Hyperscript Tools
  - Informix Online ISQL
- Neuron Data
  - Smart Elements (Nexpert object)
  - Smart Elements (Openedit)
  - Open Interface Elements (Open edit)
  - CS Elements (Openedit)
- Sybase
  - SA Companion (client & server)
  - SQL Monitor (client & server)
  - SQL Debugger inspector
  - SQL Debugger console
  - SQL Data Workbench
  - SQL APT Edit
  - SQR Workbench (Easy SQR)
  - Open Client/Server
  - ISQL/SQL Server
- Unison Software
  - Maestro
  - Load Balancer
  - Express
  - RoadRunner
- ProtoSoft
  - Paradigm Plus
- Oracle
- NetLabs
  - Net Labs/AssetManager
  - NetLabs/Vision
  - NetLabs/Assist
  - NetLabs/NerveCenter
  - NetLabs/Manager
  - NetLabs/OverLord Manager
  - NetLabs/Discovery

MS Windows

- Cognos Corp.
  - PowerHouse Windows 1.2E
  - Axiant
  - Impromptu
  - PowerPlay
- ParcPlace System
  - VisualWorks 2.0
  - VisualWorks with Sybase connectivity
- XVT Software
  - XVT-Design (C Developer Kits)
  - XVT/Win (C Developer Kits)
  - XVT-Power++
  - XVT/Win (C ++ Developer Kits)
  - XVT-PowerObject Pak I for MS Windows
- Itasca System
  - ODBMS Server
  - API Libraries (C++)
- Informix
  - New Era
- Neuron Data
  - Smart Elements (Nexpert object)
  - Smart Elements (Openedit)
  - Open Interface Elements (Open edit)
  - CS Elements (Openedit)
- Sybase
  - Net-Library
  - Open Client /C
  - SQL Monitor Client
  - SQR Workbench
  - APT Execute
- ProtoSoft
  - Paradigm Plus
- Oracle
- Dynasty Technologies
  - Dynasty
- NetLabs
  - NetLabs/Vision DeskTop

Artifacts

- Cognos Corporation
  - QUICK Application
  - QUIZ Application
  - PDL Application
  - QDESIGN Application
  - QTP Application
  - QUTIL Application
  - PDL And Utilities Reference Manual
  - PowerHouse for UNIX – Primer
- ParcPlace System
  - Product Overview
- XVT Software
  - Product Overview
  - XVT Design Tutorial
  - XVT Database Demo
  - XVT-Power++ Overview
  - XVT-Power++ Demo Guide
  - XVT Power ++ Earth Demo
- Itasca System
- Informix
  - Product Overview
  - Informix R4GL Demo
  - Six demos with source codes
  - Informix ISQL Demo
  - Informix Hyperscript Demo
- Neuron Data
  - Product Overview
  - Notepad Widget example with source files
  - Pack example with source files
  - Print widget example with source files
  - Resize widget example with source files
  - Resource Picker example with source files
  - Scripting example with source files
  - Scroll area usage example with source files
  - Scroll bar usage with source files
  - Sliders usage with source files
  - Special widget example with source files
  - String search example with source files
  - Text edit validation example with source files
  - Windows MD example with source files
  - Alert Windows example with source files
  - Browsex example with source files
  - Browsinc example with source files
  - Cbox example with source files
  - Chart example with source files
  - Clock Widget example with source files
  - C++ Notepad widget example with source files
  - Drag drop example with source files
  - Draw example with source files
  - DropDown pale example with source files
  - File manager example with source files
  - File name translator example with source files
  - File Picker example with source files
  - Floating window example with source files
  - Cantt chart example with source files
  - Help engine example with source files
  - Help viewer example with source files
  - ICON generator example with source files
  - List Box example with source files
  - Local drag drop example with source files
  - Menu example with source files
  - Multiple font text example and source code
  - Notepad example with source files
- Sybase
  - SyBooks
  - APT Demo
  - Compute example with source files
  - Csr_disp example with source files
  - i18n example with source files
  - blktxt example with source files
  - Five other examples with source files
- Unison Software
- ProtoSoft
- Oracle
- Dynasty
- NetLabs

HP-UX is based on and is compatible with Novell's UNIX® operating system. It also complies with X/Open's* XPG4, POSIX 1003.1, 1003.2, FIPS 151-1, and SVID2 interface specifications.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

X/Open is a trademark of X/Open Company Limited in the UK and other countries.

MS Windows is a U.S. trademark of Microsoft Corporation.

with a text box and O.K. and cancel buttons, and its link to a value holder that holds the environmental variables, names, and other information needed for the encapsulation. The only method needed in addition to inherited ones is the one requesting the encapsulation parameters mentioned above.

The same procedure, that is, the use of predefined classes and thus minimal coding, applies to HP Distributed Smalltalk's folders containing the encapsulated product with its tools and other artifacts. The HP Distributed Smalltalk class FolderPO (PO indicates it is the folder class spawning presentation objects) has a method windowMenu, which creates a window with several pop-up menus that have labels such as Action, Edit, and so on. For a subclass of FolderPO called FolderPlusPO, these properties of windowMenu are inherited, but windowMenu is also changed (while keeping the same name), by the addition of a method artifactCreate and its label in one of the pop-up menus of windowMenu. The method artifactCreate is responsible for the inner workings of the encapsulation dialog window mentioned above.

Development Methodology

Funding for the Software Solution Broker project was subject to the condition that the development team find, justify, and implement a design that brings the tools to the consultants in the fastest possible way with the least amount of resources, including development, maintenance, and support resources. At the same time, every released version, even the very first one, had to find immediate user acceptance. Based on these stipulations the team chose a development method that is a hybrid of *iterative prototyping* and the *Fusion method.*

Our reasons for favoring iterative prototyping over a classical software design paradigm that starts with a complete specification (such as the so-called *waterfall model*) were:

- Time constraints. There are never enough engineer-months to write a complete specification, implement and test it into production strength.[1]
- Constraints imposed by the intrinsic nature of the Software Solution Broker tool we were building, that is:
  - Client-side usability. The GUI that was eventually chosen is the result of repeated testing by potential users to achieve maximum ease of use and intuitiveness, and this amount of trial-and-error cannot be specified in advance.
  - Tool accessibility. The different products on the virtual shelf have different behaviors and their own requirements for resources and administration, and creating the encapsulation process again requires much experimentation and gradual maturation based on experience that cannot be specified a priori.
  - Using the object paradigm. The software substrate chosen (HP Distributed Smalltalk with VisualWorks) is well-suited for the rapid development of GUI and client/server applications.

Based on these considerations, our overall approach was that of evolutionary prototyping, in which a fully functional prototype is ushered through repeated refinement steps into a production-strength end product. We realize that often a prototype leads only to an executable specification or a validated model, not a high-quality, stable product. However, in our case the sophisticated framework of HP Distributed Smalltalk with its semantics/presentation split and VisualWorks with its Model View Controller ensured full functionality and high quality at each refinement step because we reused the existing, high-quality code (including the library of classes) and very sparingly added new, thoroughly tested code, preferably as instances (objects) of the existing class library.

Fusion Method

While iterative prototyping can be seen as a software development philosophy that is primarily dictated by business requirements such as time to market, break-even time, or optimal return on investment, the Fusion method[2] was developed with the goal of creating a language independent, comprehensive, software project management method. Being a systematic object-oriented development method, it blends well with our software substrate, which we chose based on openness, compliance with industry standards, ease of use, and the ability to separate the server (semantics) from the remote clients (presentation). The Fusion method emphasizes a modular design process in clearly demarcated phases, so it synchronizes well with the iterative prototyping approach, which requires the repetition and refinement of certain development stages without impacting others. Furthermore, the Fusion method insists that a software development process of the complexity encountered today must cover the entire software development life cycle. The Fusion method's phased development process served as the blueprint for the Software Solution Broker. It can be summarized as follows[2] (our italics):

Starting from a *requirements document*, the *analysis phase* produces a set of models that provide a declarative description of the required system behavior. The analysis models provide high-level constraints from which the design models are developed. The *design phase* produces a set of models that realize the system behavior as a collection of interacting objects. The *implementation phase* shows how to map the design models onto implementation-language constructs.

In our hybrid approach we take an early, loosely defined functional prototype as our initial requirements definition (an executable specification), to be modified and refined in subsequent iterations through the three phases of analysis, design, and implementation. After each of these phases a review of the phase outputs is conducted by the development team in conjunction with users. The results of this audit are prioritized and, if deemed important, incorporated into the prototype which, through several of such review loops, evolves after a full cycle into the production product. (For details about the outputs mentioned and the complete Fusion process breakdown see reference 2, especially Appendix A.)

In summary, the two complementary methods of iterative prototyping and Fusion serve two main purposes. First, at the end of each prototyping cycle a fully functional production-strength product is released. Second, the three Fusion phases—analysis, design, and implementation—of every cycle are independent of the phases in another cycle. Therefore, we are in effect working towards several releases at the same time (see Fig. 6).

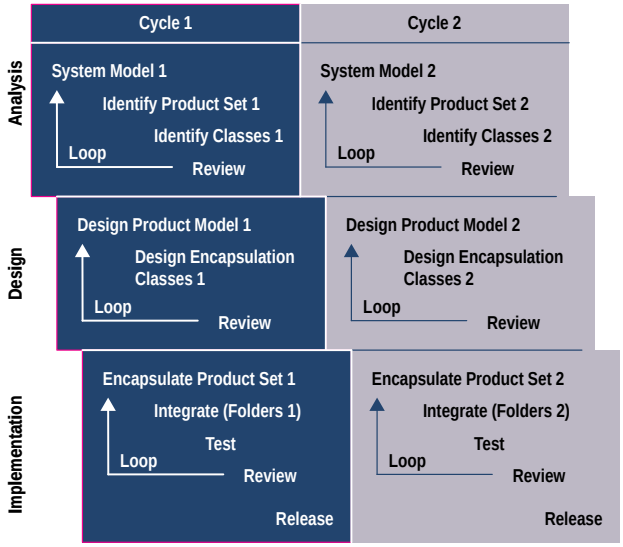


Fig. 6. Software Solution Broker development used iterative prototyping and the Fusion method, resulting in parallel development cycles.

Customizing the Software Solution Broker

In addition to being a productivity tool and a hub of product expertise for HP's technical consultants, the Broker can be customized to meet the business needs of end customers as well. To sketch how such a customization can be done using the object-oriented framework of HP Distributed Smalltalk, imagine a vendor of CAD (computer-aided design) software. Rather than offering shrink-wrapped software packages on the shelves of the store the retailer wants to offer customers an environment where they can, by navigating through virtual shelves, choose interesting products and "test drive" them in the store before deciding what to buy.

For an end customer such as the CAD software vendor, the Broker can be customized by mapping the particular customer requirements into several levels of design complexity. These levels describe in technical terms what level of intervention into the framework of HP Distributed Smalltalk is needed to alter and customize the existing classes and methods. On the lowest level, the requirements fit the HP Distributed Smalltalk framework exactly, and the system can be built from existing classes without change. A higher level of intervention would be needed to construct the Software Solution Broker for the CAD software vendor. Slight modifications of core services (relating to containment and life cycle semantics), in addition to class augmentation and overloading of methods, would be recommended. Working with predefined, well-documented levels of intervention that are necessary to meet a customer's requirements has the advantage of communicating to the customer in advance, during the analysis and before system design begins, how much reuse of the framework is possible, and how much non-framework augmentation is necessary. Intervention levels are thus not only technical assessments but also indicators of the final costs for the system.

Conclusion

The Software Solution Broker was not a typical client/server application development project. We were not primarily concerned about two-tier or three-tier architectures, about objects per se, about the one "right" programming language, or about coding. In fact, we went the opposite route. Based on the working requirements of HP's technical consultants and our own analysis of how consultants work with customers, we resolved to translate these requirements into a system built from distributed objects. The building, however, consisted mainly in the skillful choice of existing classes and the exploitation of HP Distributed Smalltalk's framework. The novelty in our approach lies not in the coding of new structures, but in the extensive application of reuse. In fact, whenever new code seemed required, we took it as a warning that further analysis was needed to look for prefabricated code within the framework of HP Distributed Smalltalk. This simple principle, essential for a fast time to market, also guaranteed a short turnaround time and high quality.

Through its first two releases, 1.0 and 2.0, the Software Solution Broker can be viewed as a distributed productivity tool offering three overlapping types of services. These three types can be described metaphorically as a virtual software shop for the display of individual products, a consultative workbench or simulated classroom for studying and experimenting with several collaborating products, and a virtual demo center with remote satellite offices where the technical consultants can build prototypes and create demos for a customer. Looked at from a broader perspective, however, the Software Solution Broker architecture and implementation are, with small customization, also ideal for other, related applications that require one (or a few) persistent centers and many locally distributed and individually presented clients. One example is software distribution. Another is the establishment of a worldwide software application development lab where each satellite group can develop its own part locally, check it in with a central repository where it is available to the other satellites, and participate remotely in the integration of the parts into a system. Furthermore, object technology, with its concept of containers, makes available compound documents (text, picture, voice, video, etc.) that can be employed also on the nontechnical side of business as vehicles for elaborate project proposals and other communication with business customers—for instance, to propose a solution by showing a video of a prior, successful installation (this would take the place of a paper document of reference sites). In this role, the Software Solution Broker can be a *worldwide business solutions exhibit* and a convenient repository for a portfolio of repeatable solutions from which the customer, advised by a consultant, can select products the way we now choose from mail-order catalogs.

Acknowledgments

A project survives by the benevolent patience of its sponsors, the enthusiasm of the project team, and the acceptance of the receiving customers. In the case of the Software Solution Broker we've tried to justify this benevolence by turning our research ideas into a useful tool in the shortest possible time.

This feat was only possible with the support of our managers David Kirkland and Sherry Harvey. Special thanks are due Chu Chang, general manager of the Professional Services Division, for his encouragement. We are also very grateful for the contributions and for the heated (but object-ive) discussions with friends and partners from many HP's entities. Nothing, however, would have happened without the quick and thorough feedback from our customers, the technical consultants in the field. The Software Solution Broker is dedicated to them.

References

1. F.P. Brooks, *The Mythical Man-Month: Essays in Software Engineering*, Yourdon Press, Englewood Cliffs, 1982.

2. D. Coleman, et al, *Object-Oriented Development—The Fusion Method*, Prentice Hall, 1994.

Bugs in Black and White: Imaging IC Logic Levels with Voltage Contrast

Voltage contrast imaging allows visual tracking of logical level problems to their source on operating integrated circuits, using a scanning electron microscope. This paper presents an overview of voltage contrast and the methods developed to image the failure of dynamic circuits in the floating-point coprocessor circuitry of the HP PA 7100LC processor chip.

by Jack D. Benzel

As pressure for higher performance and higher integration drives integrated circuit design towards increasing complexity, IC designers need an ever-broadening set of analysis and debugging tools and methodologies for tracking down functional bugs and electrical margin issues in their designs.

In developing the new HP PA 7100LC PA-RISC microprocessor chip, the floating-point arithmetic logic unit (FPALU) megacell used design techniques based on the PA 7100 design.[1] The FPALU design is implemented with mostly mousetrap-style dynamic logic[2] with significant use of single-ended dynamic logic in the last pipeline stage.

Past experience in debugging electrical problems in mousetrap designs has shown these problems to be very difficult to find.[3] A failure mechanism that emerged in prototypes of gate-biased PA 7100LC FPALUs proved highly challenging and evasive and required a large engineering effort to get from detection to the root cause identification. The voltage contrast imaging methodology proved useful in analyzing and later confirming the root cause of the failure mechanism. Results from the analysis allowed us to correct the design and verify its quality.

The Wall

The FPALU failure mechanism was named "the wall" because of its appearance on a frequency-versus-voltage shmoo plot depicting regions of passing and failing vectors (see Fig. 1).

Considerable engineering resources were applied toward finding the root cause of the wall using many of the techniques that had proved successful on previous design projects, including but not limited to shmoo plots, failing vector/opcode analysis, clock phase stretching, focused ion beam (FIB) experiments, and simulations of probable circuit failures.[3] These techniques were not providing enough information, and a new methodology was clearly needed.

Why Voltage Contrast?

Another HP design team had recently had success in using an electron-beam prober[4] to track down the root cause of a noise problem on the same CPU chip.

Previous experience with another project several years ago provided insights into a methodology similar to electron-beam probing called voltage contrast, using a scanning electron microscope (SEM). After considering the various trade-offs it was decided to proceed with the voltage contrast imaging while keeping open the option of going to electron-beam probing if further analysis was required.

SEM Fundamentals

The SEM displays objects by sensing and imaging the release of secondary electrons from the surface of a sample which is held in a very high vacuum. A finely focused beam of electrons accelerated from an electron gun with a thousand-volt potential is swept over the surface of the sample in much the same way that a television screen is scanned. As the high-energy electrons in the beam strike the sample, several valence electrons will be "knocked loose" from the sample as the impinging electrons lose energy. These now-free electrons, or secondary electrons, find their way to the surface of the sample and are released from the surface. A highly biased metal screen situated near the sample collects escaping secondary electrons into a detector which generates a signal proportional to the number of electrons collected. The signal from the detector is amplified and displayed on a CRT screen which is scanned in synchronization with the electron beam sweeping the sample.

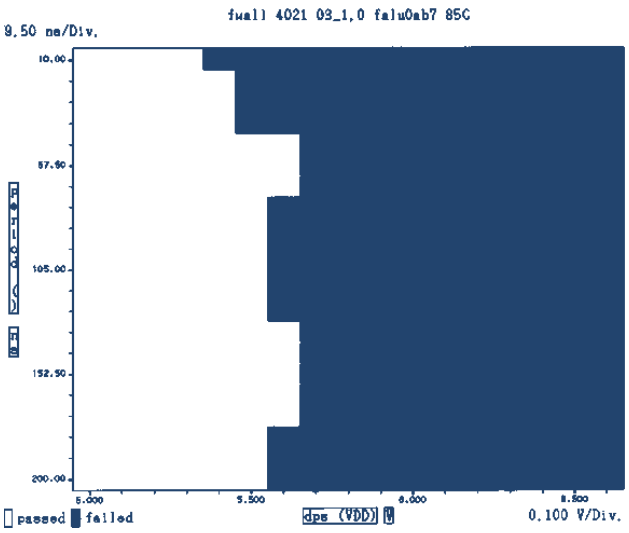


Fig. 1. Shmoo plot of "the wall."

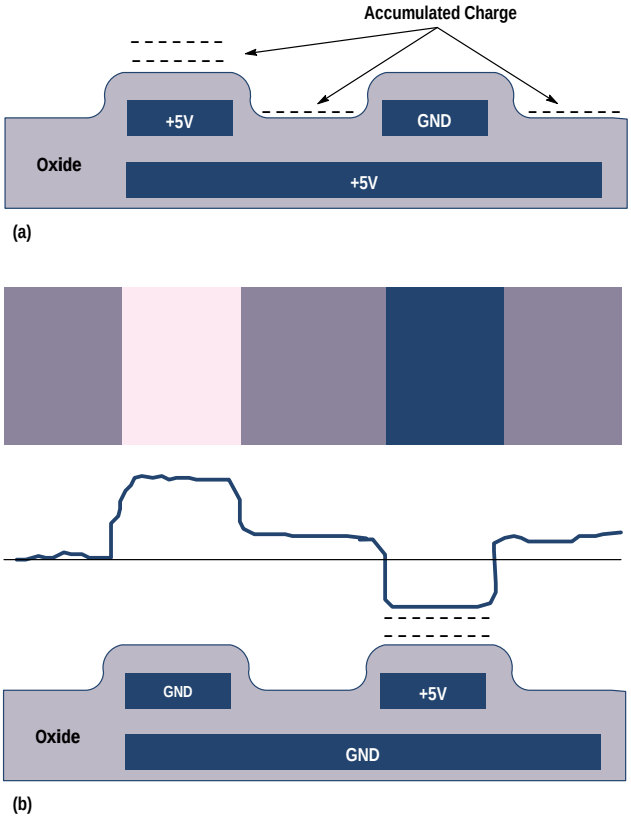


Fig. 2. (a) First pass "charge" of IC surface with secondary electrons. (b) Second pass "read" of charged surface (bottom), resulting video signal (middle), and 2D video image (top).

Voltage Contrast Imaging

Voltage contrast imaging uses the electrical nature of the SEM to view voltage potentials on a sample changing with time. Figs. 2a and 2b show a cross section of the top two metal signal layers of an IC with the metal lines insulated by an oxide.

The imaging is done in two stages: charging and reading. Fig. 2a shows the state of the IC at the end of the charging stage. The positive potential of the buried metal lines attracts and holds the generated secondary electrons on the surface of the oxide above the metal lines. These charges will remain on the surface for long periods of time, basically acting like a capacitor.

Fig. 2b shows the state of the IC at the end of the read stage with the voltage potentials of the metal lines now changed. The resulting detector signal level and the CRT image generated from it are also shown above the cross section. As the electron beam sweeps the surface of the sample, the electrons that were once held by the positive charge of the upper-left and lower metal lines (Fig. 2a) are knocked off the surface and are collected into the detector, generating a bright signal on the CRT. On the other hand, the upper-right metal line is now more positive, and the surface above it will release fewer secondary electrons as the surface capacitively charges, corresponding to a lower number of electrons collected and thus a darker signal on the CRT.

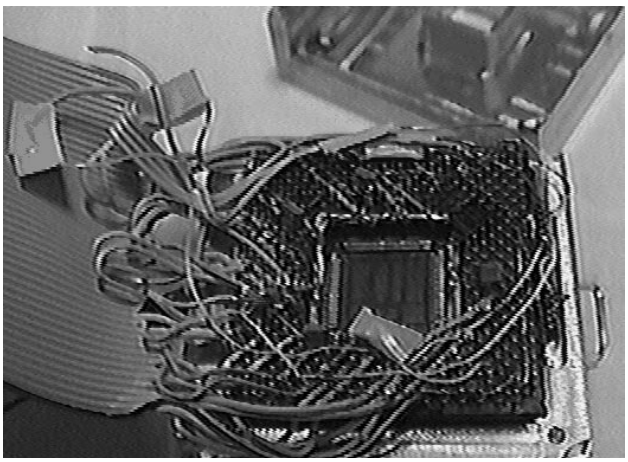


Fig. 3. Video image of DUT fixture for voltage contrast setup with top shield removed.

DUT Preparation

Preparing the IC for the SEM environment required careful attention to several details as follows:

- *Clean Power Environment.* Some previous experiments indicated that the wall was somewhat remedied by a power environment that restricted the V_{DD} current supply. Therefore, careful attention was paid to provide adequate low-inductance power feeds with adequate decoupling capacitance.
- *Simple Vector Stimulus.* Restricted cabling into the SEM chamber and easy portability between two different SEM facilities required a simple method for executing a wall-sensitive floating-point operation (FLOP). A successful method was developed to launch and step through the phases of a FLOP using the JTAG†-conforming serial test port and a serial test board.
- *Image Capture Synchronization.* The capture and imaging of events on the SEM system requires a synchronizing signal generated by the device under test (DUT). Several small surface mount ICs were mounted on the PA 7100LC package to decode the clock signals and derive another synchronizing signal to provide the SEM with an accurate sync pulse that identified the leading clock edge at the starting phase of the failing FLOP.
- *Minimize Outgassing.* To achieve an adequate vacuum in the SEM system, materials that had minimal outgassing were required. This prevented the use of heatshrink tubing and quick-cure epoxies and required careful cleaning of the DUT.
- *Packaging.* The packaging fixture containing the CPU (see Fig. 3) met several requirements. The wall was a high-temperature phenomenon and required heating the part inside of the SEM with large resistors mounted inside the fixture. The metal enclosure shielded all but the die surface from the electron beam, since the beam will positively charge plastics (wiring, capacitors). The shield also prevented electrical signals in the DUT wiring from interfering with the beam's trajectory. The last requirement filled by the fixturing was a compact size to fit inside the small SEM chamber.

† JTAG is the Joint Test Action Group, which developed IEEE standard 1149.1, *IEEE Test Access Port and Boundary-Scan Architecture.*

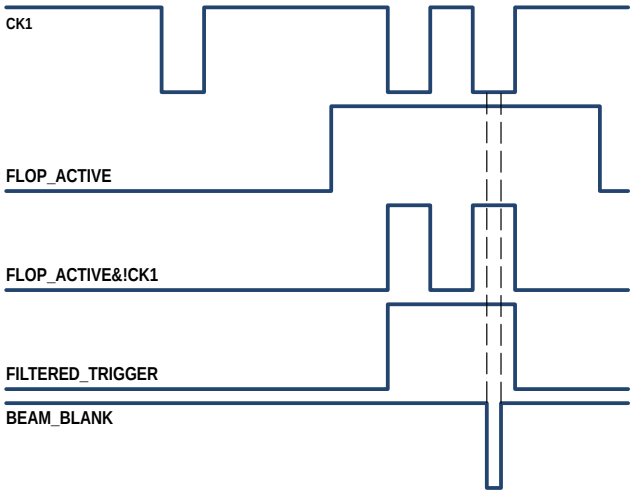


Fig. 4. Beam blanking and synchronization signal generation.

Imaging Dynamic Signals

The electron beam scan is synchronized with the scan of the video display tube and consequently has a slow refresh rate of 1/60 second. This slow refresh rate works well for stationary objects and static electrical signals, but the signals of interest involved in the wall failure are dynamic, typical of mousetrap designs. The imaging of dynamic signals required the development of a new process.

Synchronization and Beam Blanking

The slowest rate at which the DUT could be clocked with reliable operation of the scan path driven through the JTAG port was 2.5 MHz, giving a 200-ns phase or period during which dynamic signals would be active. Connecting a pulse generator to the DUT's sync pulse allowed the generation of a variable-width, variable-delay pulse (see Fig. 4) which was used to blank the electron beam scanning the DUT. Using this blanking signal, the SEM could be controlled to charge or read the IC only during the time of interest when the wall-related signals were active. A 100-ns sample window was chosen for the blank signal, which was centered in the clock phase to reduce possible overlap into adjoining phases.

Once the beam was properly synchronized and blanked, the apparent lack of information in the video image shown in Fig. 5 gave a strong indication that more development was needed.

Image Capture

The next problem to resolve was imaging the brief 100-ns video information successfully. Several ideas were evaluated and tried before an acceptable method was found:

- *Photographic Film Integration.* The SEM focuses the light from a secondary CRT onto the film plane of a Polaroid camera over a period of several minutes while exercising the DUT. This method resulted in either completely black or very indistinct images of the IC.
- *Two-Dimensional Scan.* The SEM can operate with basically a zero-frequency vertical scan rate. This provides an image of a single horizontal slice of the IC surface while improving the refresh rate. Changes in beam intensity were undiscernible in this mode.
- *Two-Dimensional Scan in Oscilloscope Mode.* Using the same two-dimensional scan mode as above, the intensity vector of the SEM's display can be used to drive the vertical component of the video signal. The resulting image is reminiscent of an oscilloscope display showing intensity on the y axis. No discernible changes in intensity were visible in this mode as well.
- *Two-Step Charge/Read.* Instead of trying to charge and read on each or every other FLOP, the process was broken into two steps. The first step involved turning the beam on only during the phase of interest while the part was executing wall FLOPs over a period of three minutes. A long integration time was required because each time the beam turned on it only charged a tiny area of the field of view. At the end of the integration time, the beam was turned off, the IC powered down, and the beam blank removed from the SEM. The IC now had a surface charge that reflected the state of the metal lines during the phase of interest. The second step was to turn the beam on with no blanking to read the surface charge in its first pass over the IC. The resulting video image was clear but brief (one video frame). This process produced an image in which metal lines with a positive voltage were white and metal lines at ground were black. Another small variation in this process was not to power down the part before the read step. The resulting image took a little more thought to interpret because only the metal lines that changed state from the previous step were black or white.
- *Two-Step Charge/Read with VCR Frame Capture.* By adding a VCR to the setup, the resulting video image fed to the CRT could be captured on tape and then freeze-framed for viewing. The purchase of a VCR with a forward and reverse single-frame jog shuttle control greatly aided in isolating the image captured on a single frame. It was apparent from the videotape that the majority of the IC's surface charge was removed in the first sweep of the beam across the die area. This last methodology was used successfully for imaging the dynamic signals in the FPALU.

Results

Once the methodology was established, over 120 images were captured and catalogued on video tape over a four-week period. Several days were spent at the outset trying to understand why an active clock line in the imaged phase

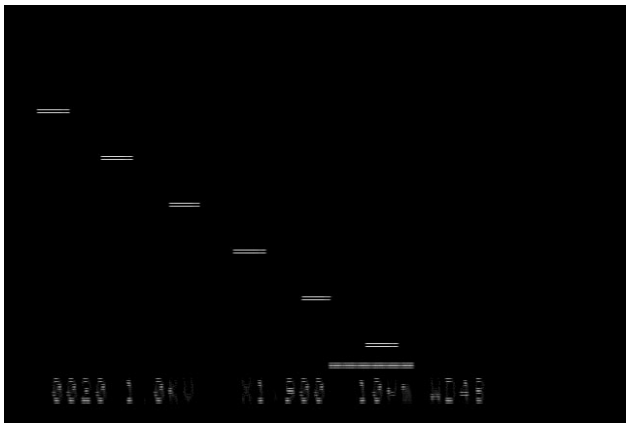


Fig. 5. Video image of the first-pass imaging attempts.

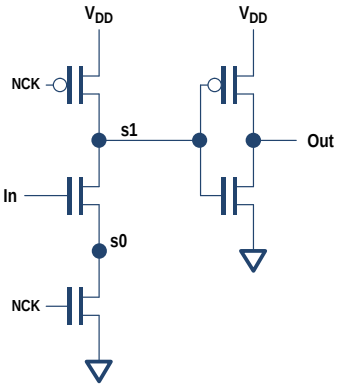
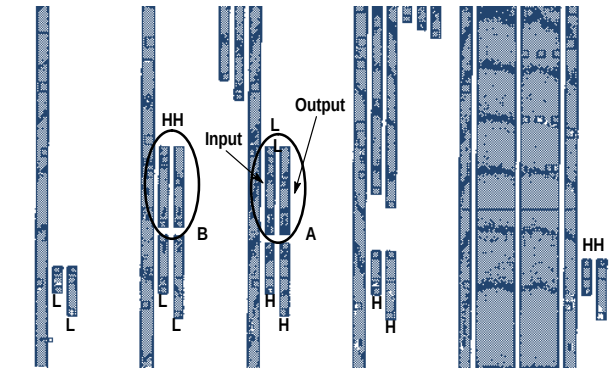


Fig. 6. Zero_topH mousetrap buffer.

was not showing activity, a key indicator that the proper phase of the FLOP was being captured. This issue was never satisfactorily resolved, yet phase-by-phase clock gating in the FPALU ensured that the signals would only be active and thus visible in the phase of interest.

Figs. 6, 7a, and 7b show the schematic, artwork, and voltage contrast image of probably the clearest failure identified. The circuit in Fig. 6 shows a mousetrap buffer whose storage node, s1, was somehow being compromised, possibly through a ground differential problem or a noise spike on the input.



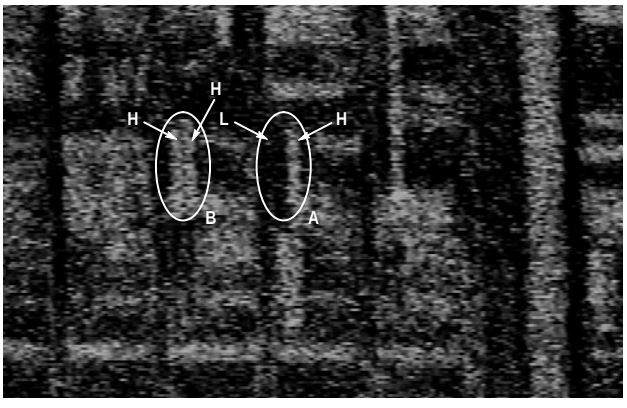


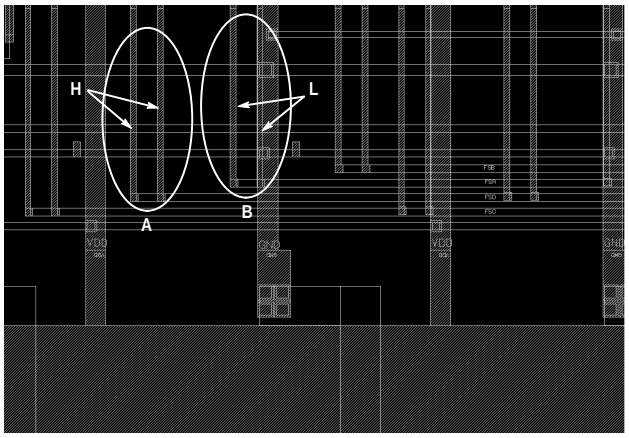
Fig. 7. (a) Metal 3 plot of Zero_topH buffer with failing input/output pair A. (b) Voltage contrast image of victimized buffer with failing input/output pair A.

Circle A in Fig. 7a identifies the buffer's input on the left and the output on the right. The expected value of each metal 3 line is indicated above the lines (L=Low, H=High).

Fig. 7b shows a voltage contrast image captured from the videotape showing the failure of the buffer. The image clearly shows a low level on the input (black) and a high level (white) on the output of the buffer in circle A. Note the difference between circle A and circle B which identifies the input and output of an identical buffer with no failures. It became clear from this picture that the electrical event that caused the buffer to output a high level was transitory in nature and not a static event. The read step of the image was taken with the IC powered down.

Metal 1 and even metal 2 lines can be difficult to image unless they are well-isolated from other metal structures. Fig. 8a shows the artwork and expected values where several metal 1 lines were imaged. The vertical metal 1 route in circle A should have a high or white level, and the route to the right of it in circle B should have a low or black level.

Fig. 8b is the voltage contrast image showing the logical misfiring (high/white) of the metal 1 route in circle B. This



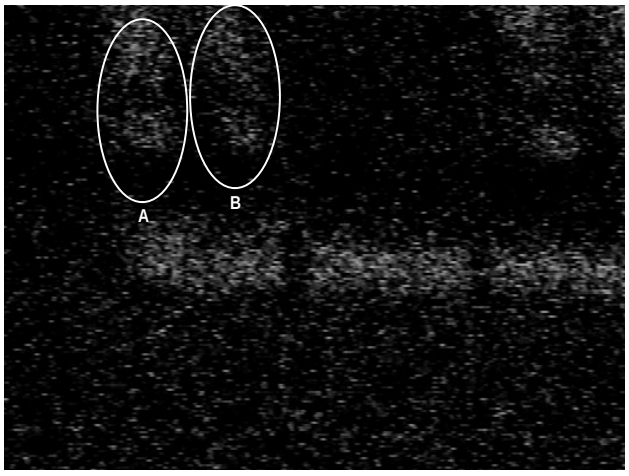
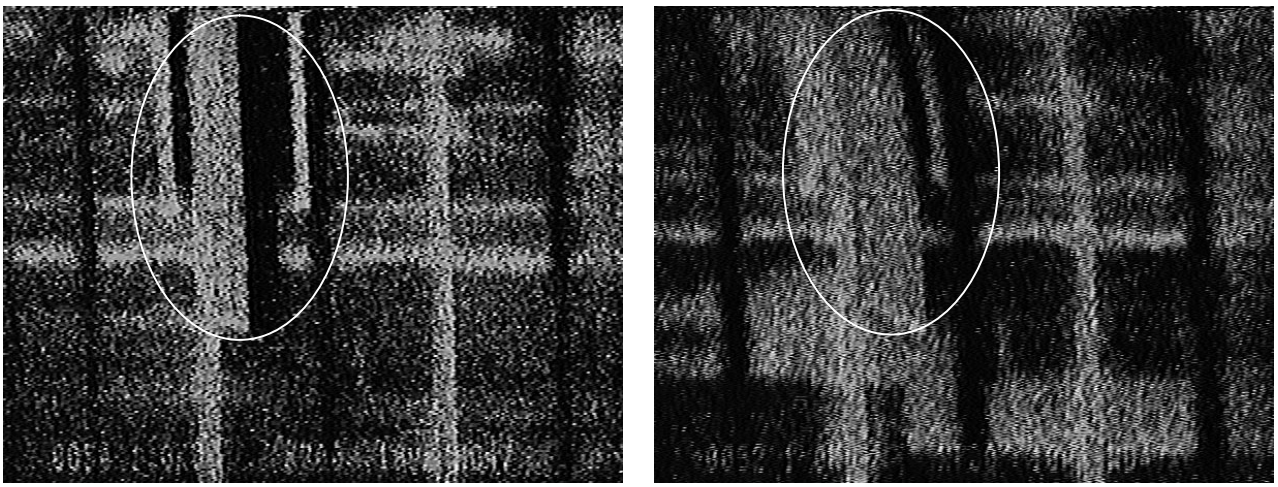


Fig. 8. (a) FS[ABCD] bus artwork. (b) Voltage contrast image of FS[ABCD] bus in metal 1 showing correct firing of the lines in circle A and the incorrect firing of lines in circle B.



(a) (b)

Fig. 9. (a) Metal 3 structure(vertical routing) in passing state at nominal voltage. Horizontal routes are metal 2. (b) Metal 3 structure in failing state at high voltage with wall failure.

failure was not seen until the root cause of the wall was identified and the proper FLOP for arming the failure was identified.

The logical states of individual lines of dense bus structures in lower metal levels can be difficult to discern, yet differences between two states can often be readily identified.

Figs. 9a and 9b illustrate the differencing technique with an example of a metal 3 structure in both a passing and failing state (note the differences in the vertically routed lines in the top-center of the figures). The bend or distortion in Fig. 9b is the result of poor synchronization between the SEM and the VCR that recorded the images. Note also the changes in the horizontally routed metal 2 lines.

One technique that greatly aided the interpretation of the captured images was to plot the artwork of the areas being imaged and annotate the plots with the expected logical levels as derived from a simulator.

Improvements and Future Use

It is difficult to determine if E-beam probing would have provided quicker, more pertinent information than voltage contrast. Each tool has its own benefits and drawbacks that the IC designer must weigh in light of the problem to be solved.

Additional IC physical structures and layouts could make new designs more amenable to voltage contrast imaging as well as E-beam probing and FIB experiments. These features could provide regular, systematic, top-level-metal access to control and data path signals throughout the design. Top-level-metal access could be provided through directed routing or through "via stacks" to top layers from lower-level metal routes. The efficiency of such features in terms of improved accessibility versus increased layout area is unknown.

The image quality obtained from the SEM for voltage contrast work could be improved by changing the electron gun filament from tungsten to a crystalline element. The crystalline filament would increase the beam current and thus effectively provide a brighter image without increasing the beam energy which reduces resolution.

Conclusions

The use of voltage contrast imaging proved to be a useful tool for analyzing and verifying the FPALU margin failure known as the wall. Although the information gleaned from the process did not lead directly to the discovery of the root cause of the failure, the voltage contrast process functioned well as a clue generator as suggested in reference 3 and provided important confirmation of the root cause hypothesis.

Acknowledgments

The author would like to thank Phil Fangman and Chris Anderson for their expertise and operation of the SEM, Rick Butler for his insight and probing questions which helped keep the project on track, and Craig Heikes and Bob Miller for their insight and musings on mousetrap and dynamic logic design and debug.

References

1. C. Heikes, et al, "The Design of 200 MFLOP Floating-Point Megacells for PA-RISC 7100," *Proceedings of the 1992 HP Design Technology Conference.*

2. B. Miller, "Mousetrap Logic—Its Uses and Abuses," *Proceedings of the 1992 HP Design Technology Conference.*

3. B. Miller, et al, "Turn-on and Debugging of MouseTrap Logic Designs," *Proceedings of the 1993 HP Design Technology Conference.*

4. S. Ferrier, et al, "Electron Beam Probing at HP: Contributions and Improvements," *Proceedings of the 1993 HP Design Technology Conference.*

Component and System Level Design-for-Testability Features Implemented in a Family of Workstation Products

Faced with testing over twenty new ASIC components going into four different workstation and multiuser computer models, designers formed a team that developed a common system-level design-for-testability (DFT) architecture so that subsystem parts could be shared without affecting the manufacturing test flow.

by Bulent I. Dervisoglu and Michael Ricchetti

Members of the latest-generation family of HP workstation and multiuser computer products use the same system architecture and differ mostly in their I/O subsystem architecture and configuration. From a system development point of view an important characteristic of these products is their use of a new high-speed system bus architecture and a large number (over 20) of new ASIC components that were developed to implement all of the various different configurations of the product line. Furthermore, all components that interface with each other via the system bus are required to operate with the same high-frequency system clock.

A further difficulty was that four different models, ranging from a single-user desktop workstation to a multiuser computer, were being developed by different design teams that were both organizationally and geographically separated from each other. This made it necessary to develop a common system-level design-for-testability (DFT) architecture to be used throughout the system and across the different computer models so that subsystem parts could be shared among the different computer models without affecting the manufacturing test flow.

To address these difficulties a DFT core team was formed at the very early stages of the project. Because of the large number of different ASIC teams involved, it was decided that all ASIC teams at the same site would be represented by a single representative on the DFT core team. This team has been instrumental in achieving goal congruence among the different design teams and manufacturing organizations. Furthermore, the presence of the DFT core team made it possible to develop and implement a DFT methodology that was used by all of the ASIC teams, although the level of adherence varied. The DFT core team also collected data and performed DFT design reviews for some of the ASICs.

ASIC DFT Design Rules and Guidelines

One of the first activities of the DFT core team was to develop a set of design rules and guidelines to be followed by the ASIC design teams to ensure that DFT features would be common among the various components. This made it possible to share efforts and results and to access the different DFT features in the ASICs during prototype system bring-up. The following is a summary of these rules.[1]

1. *All (functional) system clocks must be directly controllable from the chip pins and must not be used for any other function.* All systems use a common ASIC component (the system clock controller ASIC) to drive their clock terminals on the system board. This ASIC has control pins through which it can be programmed for different clock generation schemes as well as for starting and halting the system clocks. Thus, not only the individual ASICs but also the entire system board has directly controllable clocks.

2. *All scan and test clocks must be directly controllable from the component pins, which must not be used for any other purpose.* On the system board all test clocks are tied together and controlled from a single test point.

3. *For each ASIC there is a specific reset state which is entered when the component's* ARESET_L *signal is asserted.* On the system board, the power-on condition is detected and is used to reset the ASICs to a known starting state. Next, the memory controller ASIC generates an SRESET_L signal to all other components on the system bus. Additional reset signals are generated by other ASICs for use locally.

4. *All ASICs must implement a dedicated boundary scan register and its associated test access port (TAP) as specified in IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture.*[2] Serial scan-in and scan-out ports of all ASICs in the system (including the PA 7200 processor, which is on a separate module) are connected to form a single serial scan chain.

5. *Access to each ASIC's on-chip test functions must be provided using the IEEE 1149.1 test access port (TAP) protocol.* The same TAP controller design[3] is used or heavily leveraged in many ASICs. This way, test features implemented in this controller as an extension to the IEEE 1149.1 standard were easily leveraged across different ASICs. For example, the DRIVE_INHIBIT/DRIVE_ENABLE instructions and the OUT_OFF bit in the boundary scan register (see "TAP/SAP Controller," below) are duplicated in different ASICs in this way.

6. *All ASICs shall be designed to support I_{DDQ} testing, whenever this is not prevented by the technology used.* In most cases this requirement did not present any further design constraints or changes. In a few cases, an internal I_{DDQ} enable signal had to be used to disable active pull-up and pull-down circuits. However, because of schedule and cost considerations the PA 7200 processor chip does not support I_{DDQ} testing.

7. *All ASICs shall implement internal scan for testing. The percentage of internal nodes that are scannable shall be kept as high as possible without sacrificing major chip area or otherwise affecting the design methodology.* For most practical purposes all ASICs have implemented internal scan for 100% or nearly 100% of all internal flip-flops. However, because of design style and technology differences, some portions of the PA 7200 processor chip are not scannable.

8. *There shall be no asynchronous logic implemented in the ASICs.* Lack of asynchronous logic is an important requirement for many CAD tools for generating test vectors. Furthermore, this rule is intended to prevent side effects caused by changing the internal and external signals in arbitrary sequence. The only exception to this rule is granted for the reset signals, which are implemented to follow a carefully planned system reset strategy.

The following sections describe some of the DFT features that have been implemented in the ASICs. Not all features are implemented in all ASICs. Among the various ASICs, the memory controller stands out as the chip with the most extensive DFT features.

TAP/SAP Controller

Access to all on-chip DFT features is implemented through a test controller block called the *test access port/scan access port* (TAP/SAP). The test controller implements all of the required instructions for the IEEE 1149.1 TAP controller as well as an extensive set of public and private instructions which are targeted mostly for internal testing of the ASIC. Table I lists all of the TAP instructions that are implemented. Among the public instructions that have been implemented are the DRIVE_INHIBIT and DRIVE_ENABLE instructions which are used to set and clear a latch in the system logic domain (not considered part of the test logic).

System logic for all ASICs has been designed such that for normal system operation (i.e., when test logic is not controlling the I/O pins) the ASIC can drive out only if the DRIVE_INHIBIT latch is cleared. Each ASIC uses its ARESET_L input to clear the DRIVE_INHIBIT latch during power-up. Whereas ARESET_L controls the DRIVE_INHIBIT latch only if the TAP is in a reset state, explicit TAP instructions can be used at other times to set or clear this latch. This scheme allows in-circuit ATE programs to set the DRIVE_INHIBIT latch before they terminate and reset the TAP without creating possible board-level bus contention before removing electric power from the board. Whereas the DRIVE_INHIBIT latch is considered part of the on-chip system logic, it is implemented as part of the TAP controller design so that ASIC designers implementing normal system functions do not have to deal with any of the issues surrounding the DRIVE_INHIBIT and DRIVE_ENABLE operations.

Table I
TAP Instructions

Instruction	Drive I/O Pads	Scan Register
EXTEST	Boundary Register	Boundary
BYPASS	System Logic	Bypass
SAMPLE/PRELOAD	System Logic	Boundary
IDCODE	System Logic	ID Code
HI_Z	High-Impedance	Bypass
DRIVE_INHIBIT	Boundary Register	Bypass
DRIVE_ENABLE	System Logic	Bypass
SCAN_INTERNAL	System Logic	f(Mode)
CHIPTEST	High-Impedance	f(Mode)
INTEST	Boundary Register	Boundary
DR_SCAN	System Logic	f(Mode)
SELECT_MODE	Boundary Register	Mode
SET_MODE_BIT	Boundary Register	Mode
CLR_MODE_BIT	Boundary Register	Mode
ISAMPLE	System Logic	Bypass
ESAMPLE	System Logic	Bypass
DS_DRIVE	Boundary Register	Boundary
DS_RECEIVE	System Logic	Boundary

Other TAP instructions are used to set and clear bits of the mode register to provide access to additional test features such as I_{DDQ} testing, double-strobe, and so on. It is also possible to speed up internal scan operations by switching on the parallel scan bit in the mode register. This feature enables multiplexing of the chip's I/O pins to perform serial scan-in and scan-out of the internal scan register by breaking it into three independent sections which are scanned in parallel together with the boundary register, which is always scanned using the test data in and test data out pins of the TAP.

CHIPTEST Instruction

One of the major difficulties in implementing DFT in the ASICs used for this project has resulted from a common leveraged I/O pad design that contains nonscannable latches. Furthermore, the bidirectional I/O cell implements an internal bypass path to feed into the chip the same value that is being driven onto the I/O pad by that chip. In effect, I/O pads contain nonscannable pipeline stages that control both the direction and the value of data on the I/O pad. Following a recommendation from the DFT core team the basic I/O cell design was modified to allow data values received by the on-chip system logic to be set up using the dedicated boundary scan register. In addition, system logic output values can be captured into the boundary scan register using the system clock. These design changes were coupled with features provided by the CHIPTEST instruction in the TAP controller to streamline the internal testing of the ASICs. For example, all internal logic of the memory subsystem ASICs (memory controller, slave memory controller, and data multiplexer) is tested by the following sequence:

1. Load the CHIPTEST opcode into the ASIC.
2. Use test clocks to perform a parallel scan of the ASIC internal nodes and the boundary register. At the end of the scan-in process the newly scanned-in values are automatically moved from the boundary register to the nonscannable latches in the I/O-cells.

3. Apply a single system clock to capture test results in internal nodes and system logic output values in the boundary register.
4. Repeat steps 2 and 3 for each new vector, overlapping the scan-in and scan-out operations.

Since the CHIPTEST instruction drives the I/O pins to a high-impedance state it is possible (indeed it is intended) to execute these tests on a populated system board without fear of creating board-level bus clashes during such testing.

BIST Implementation

The memory controller ASIC incorporates several wide and shallow register files that are used for queueing operations within the data paths. The total number of storage elements in the register files is quite large, so it was not practical to make these storage elements scannable. Therefore, a built-in storage test (BIST) approach was chosen to test the memory controller data path register files.

The memory controller BIST implementation was developed with the following objectives:

- Provide high coverage and short test times.
- Provide at-speed testing of the register file structures to ensure that the memory controller ASIC works at the required system clock frequency.
- Provide flexibility and programmability in the BIST logic to allow alteration of the test sequence for debug and unforeseen failure modes. In particular, the system bring-up and debug plans provide a means for system-level scan access to the state within the ASICs. Providing these features allows read/write access to the nonscannable queue states for prototype system debug.
- Provide for testability of the logic surrounding the register files through added observation and control points at the inputs and outputs of the register file blocks. This is intended to support automatic test pattern generation (ATPG) tools used to generate test vectors for the memory controller and thus ensure high coverage of the standard cell control logic for the queues.

The design of the BIST logic in the memory controller data paths is based on previous work that was done for the PA 7100-based HP 9000 Model 710 workstation. For that product, a structure independent RAM BIST architecture that uses a pseudoexhaustive test algorithm and signature analysis was developed and was implemented in the I/O controller ASIC.[4] The structure independent, pseudoexhaustive test algorithm provides 99.9% fault coverage of typical RAM faults and can provide 80% to 99.9% coverage of neighborhood pattern-sensitive faults. It also allows the test time (number of read/write accesses per memory address) to be varied according to the desired fault coverage. BIST architectures for both the present memory controller ASIC and the previous I/O controller ASIC use a test algorithm similar to that described by Ritter and Schwair.[5] Using the system clock for BIST execution, the RAM structure can be tested at the normal system clock rate, thus providing at-speed testing of the RAM.

A dual-port write/single-port read register file from the present memory controller data path, with test structures that provide both BIST and ATPG support similar to the previous I/O controller BIST architecture, is shown in Fig. 1. The two write ports, A and B, can both be addressed and written independently. The single read port can also be addressed and read independently of the A and B write ports. Thus, two write operations and one read operation can all occur simultaneously for one to three register locations, depending on the A, B, and read port addresses.

Given the dual-ported design of the memory controller register files, it was necessary to extend the previous I/O controller BIST architecture to test a dual-ported RAM. This meant that the memory controller BIST implementation should be able to test not only the simultaneous dual-write operations but also the various combinations of A/B write and read operations to verify that the port interactions are working correctly. For the dual-port register files in the memory controller such interactions include an internal bypass when the read address is the same as either of the A or B write addresses and a B-port dominant write when the A and B write addresses are equal. This dual-write BIST algorithm is described in reference 6.

For the register file shown in Fig. 1 each of the BIST structures—LFSR (linear feedback shift register), SHIFT, COUNT, and MISR (multi-input signature register)—is dedicated to BIST. Each register file also has its own dedicated programmable BIST control queue for sequencing the BIST algorithm. The BIST_MODE signal enables the BIST functions and can be

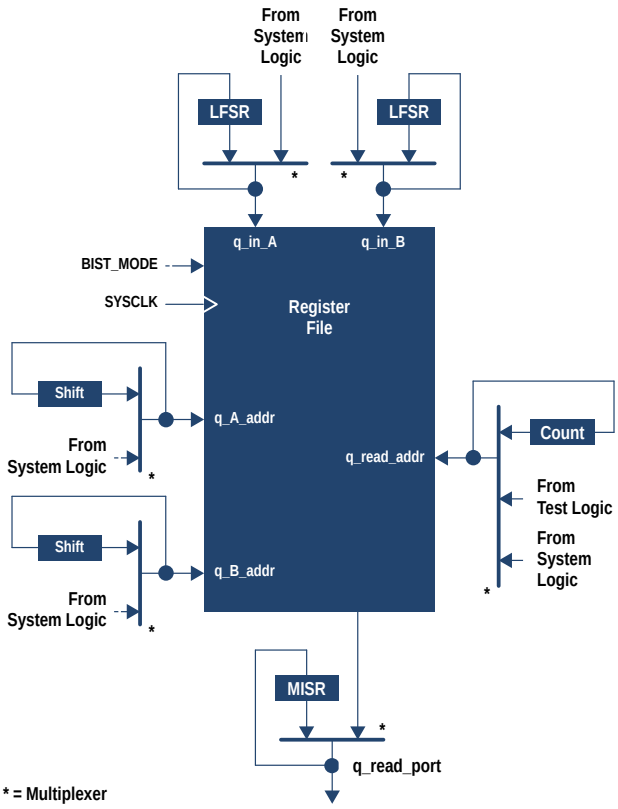


Fig. 1. Dual-port register file with built-in storage test (BIST) and automatic test pattern generation (ATPG) features. The inputs to the central, embedded RAM structure are provided by multiplexing between the normal system value and a BIST register, which is implemented as a linear feedback shift register (LFSR). The output multiplexer makes it possible to capture the outputs into a multi-input signature register (MISR) and to send either the RAM outputs or the MISR contents to the rest of the system.

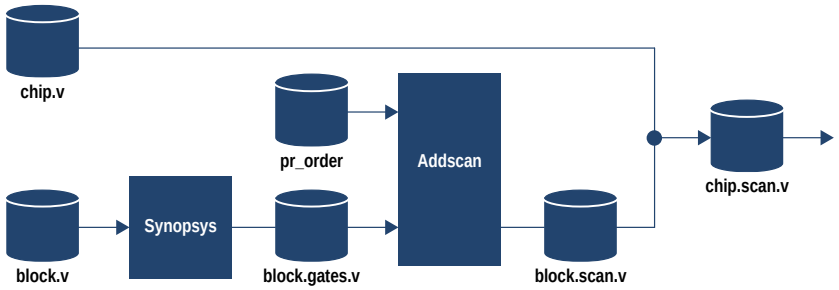


Fig. 2. Addscan tool flow. Addscan is an in-house software tool for scan insertion. Synopsys is an automatic design synthesis tool from Synopsys, Inc.

controlled either by a pin on the chip or through other test access logic such as an IEEE 1149.1 TAP controller or P1149.2 SAP controller.[2,7] All of the BIST registers are implemented in standard cell blocks separate from the data path register files. A detailed description of the memory controller BIST implementation and operation, along with hardware overhead and test coverage, can be found in reference 6.

Test Tools

The following sections describe the tools and tests that were used and developed to test the three memory subsystem ASICs: the memory controller, the slave memory controller, and the data multiplexer.

Addscan. Fig. 2 shows the flow for scan synthesis. All three memory subsystem ASICs were designed using a structured-custom design method.[8] Each standard cell block used the in-house Addscan tool[9] for scan insertion and the scan links between blocks were connected by hand in the top-level netlist of the chip. The internal scan order of each block is based on post place and route information.

Test Vector Generation. The test vector generation flow is shown in Fig. 3. The ATPG tools from Crosscheck, Inc. were used to generate most scan-based vectors. Some vectors were hand-generated. A gate-level netlist of the chip, prior to Addscan scan insertion, is used to create an ATPG database for vector generation. A similar data base is used by designers to do Timver static timing analysis. Timver, a timing analysis tool from Aida, Inc., is used as a part of the test methodology for two purposes. First, it allows the design to be checked for hold violations on all paths to guarantee that there will be no timing violations even if ATPG vectors exercise nonfunctional paths in the design. Secondly, Timver critical paths can be fed back into ATPG in the form of a vector.tcp file to generate double-strobe path delay vectors.

The following test vector sets were created for each of the memory subsystem ASICs:

- Continuity. Checks for opens and shorts among the ESD protection diodes. Prepared manually.
- Ringtest. Uses serial "flush" speed (total scan path delay) through the boundary scan register as a measure of the IC process and verifies that the part is within the six-sigma range. Generated manually in the form of a Cadence Verilog body file.
- Dc. These tests use the boundary scan ring to drive out all ones or zeros for dc parametric testing. Generated manually in the form of a Verilog body file.
- Leakage and tristate testing. Places the ASIC into a high-impedance state to allow testing the I/O pads for leakage. Generated manually in the form of a Verilog body file.
- I_{DDQ}. These vectors are generated by ATPG and are used to perform static I_{DDQ} test and measurement.
- TAP Tests. These are tests targeted at functional testing of the TAP controller. Generated manually in the form of a Verilog body file.
- Chiptest. These vectors are generated by ATPG to test the core chip logic in from and out to the boundary scan ring using the TAP CHIPTEST instruction. I/O pad logic is not fully tested by chiptest vectors.
- Pintest: These vectors are generated by ATPG and will test the remaining faults (primarily in the I/O pad logic) that are not covered by the chiptest.
- Bus Holder. Further testing of the electrical characteristics of the bidirectional I/O cells. Generated manually in the form of a Verilog body file.
- BIST. BIST vectors are only generated on the memory controller. These tests require only two scan vectors, one each to set up the initialization and test passes for BIST. After that, a burst of system clocks is applied to test the target

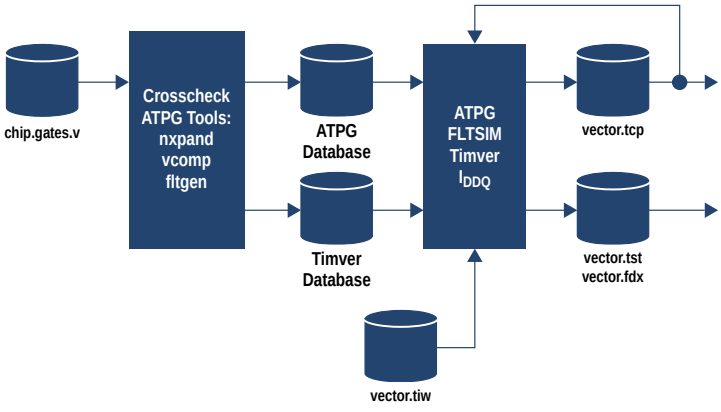


Fig. 3. ATPG (automatic test pattern generation) and timing tools flow. Timver is a timing analysis tool from Aida, Inc. FLTSIM is an in-house fault simulator for test verification. The I_{DDQ} vectors are generated by ATPG from Crosscheck Corp.

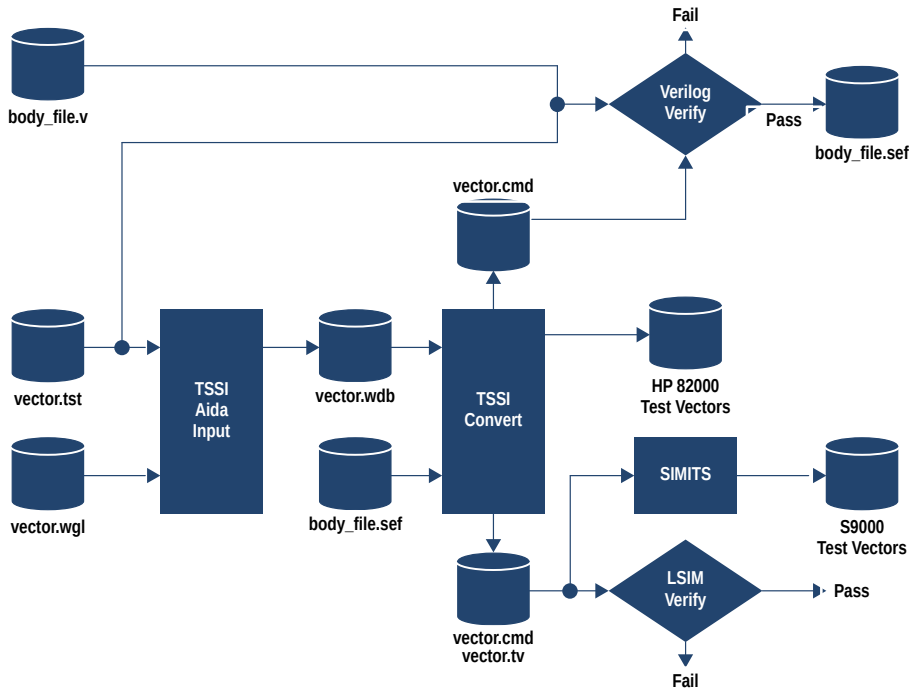


Fig. 4. Vector verification and tester translation tools flow. TSSI is a test program generation tool from TSSI, Inc. Aida represents a suite of test tools from Aida, Inc. LSIM is a special FET-level simulator. SIMITS is a format conversion tool from Schlumberger.

blocks at speed. These vectors are generated using a Perl script to produce a .tst vector file.

- Double-Strobe. These vectors are generated by ATPG based on Timver critical paths and are used to provide at-speed testing of the ASIC.[10,11]
- Ac testing of I/O Paths. These are functional tests that test the speed characteristics of critical I/O paths. Generated manually only if testing, design review, and chip characterization results indicate a concern.
- Process, Voltage, and Temperature (PVT) Block Test. Generated manually, this group of tests applies only to the slave memory controller chip which uses a unique PVT block to compensate for process, voltage, and temperature variations in a particular I/O cell.

Vector and Test Logic Verification. Fig. 4 shows the flow for test vector verification using Verilog or LSIM (a special FET-level simulator). Test vectors from ATPG can be directly converted using TSSI (a tool for test program generation from TSSI, Inc.) into a command file format and verified against a gate-level netlist in Verilog or a FET-level netlist using LSIM. Alternatively, test vectors can be simulated using a Verilog body file. A body file is a wrapper or test jig that can either be a test vector set itself (hand-generated functional tests) or can run scan-based ATPG vectors using a scan and clock sequence.

The AT&T Tapdance tool was used for further verification of the TAP logic before tape release of the ASICs. Tapdance generates a set of IEEE 1149.1 compliance tests to verify standard TAP functionality. The Tapdance vectors were converted using Perl scripts† into a Verilog force file and simulated on a gate-level netlist.

Tester Format Translation. Fig. 4 also shows the flow for translation of vectors into a tester format. Using TSSI, vectors were formatted directly to HP 82000 tester format. To get to the Schlumberger S9000 tester, vectors were first formatted to LSIM and then passed through SIMITS, a format converter from Schlumberger.

Using a Verilog programming language interface that outputs a TSSI simulation event format file dump, vectors can also be translated from body files to one of the testers.

System DFT Features

The new systems have been designed to provide a method to access ASIC scan paths, both boundary and internal, at the system level. This has two major purposes. First, it provides a means of accessing the internal state of complex VLSI components. This provides additional hardware state information to designers that would typically be inaccessible and can aid traditional prototype bring-up and debug methods. Second, it provides the ability to do scan-based testing of board and system interconnect and internal scan testing of ASICs.

The following test and debug features are provided by system scan access:

- Ability to halt the system clocks and interrogate the internal scan state of the ASICs.
- Single-cycle debug of the system core by halting the system clocks, interactive scanning of the internal state, and then starting or cycling the system clocks.
- Board-level and system-level interconnect testing and interactive debug using boundary scan. This includes testing connectors between two boards where boundary scannable buses cross the connector.
- Ability to test an ASIC while it is on the board using boundary and internal scan. This may include double-strobe tests and running on-chip BIST, if supported by the ASIC under test.

As part of the overall DFT requirements, all ASICs implement the IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture. This provides support for system-level scan access. In addition, key debug support features are

† Perl is a high-level programming language.

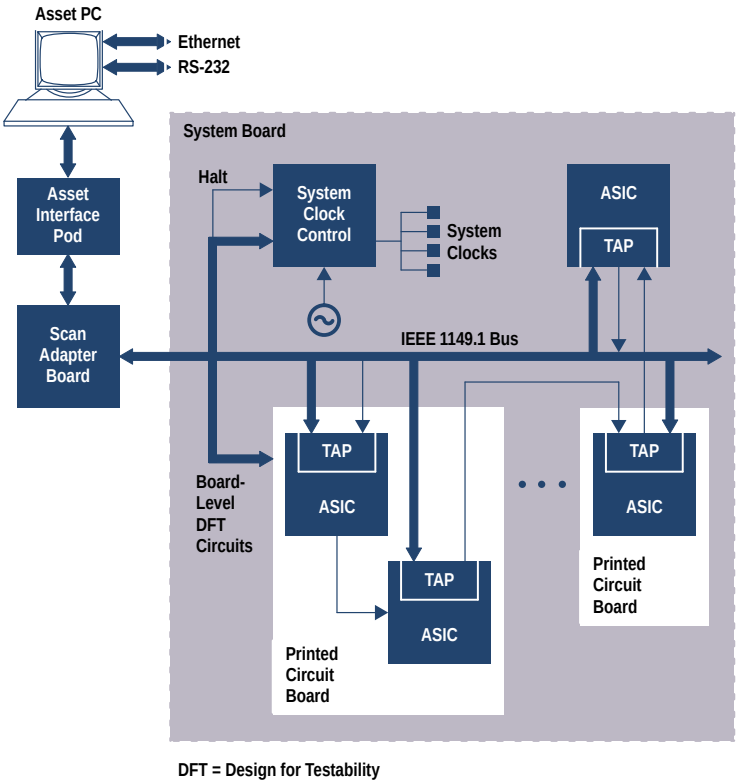


Fig. 5. System-level scan access. Asset is a set of scan tools from Texas Instruments, Inc.

incorporated into the system clock controller chip to allow for halting and controlling the system clocks. Further information on system clock controller features can be found in reference 12.

Fig. 5 shows a diagram of the system-level scan access hardware. The Texas Instruments PC-based Asset toolset is used as the interface to system scan. The Asset PC is connected to a scan adapter board via the Asset interface pod. The scan adapter board then plugs onto the system board and provides control of the system clock controller features, the TAP controller and system logic reset, the clock halt triggers, and the I/O device clock halt from the Asset software. The scan paths in the system are configured as a single serial scan chain with optional system boards implemented as dynamic scan paths that can be configured in Asset.

The Asset scan tools provide the following capabilities for system scan access:

- Interactive control of scan path data and TAP controller instructions with scan-bit name mapping and packing and unpacking of scan data.
- Macro scripting capabilities for combining several interactive operations into a single macro command. Asset also accepts serial vector format scan vectors for user-developed tests.
- Specification of system scan path configuration for dynamic scan paths and optional boards, such as CPUs, memory extender boards, and I/O.
- Scan path integrity testing and boundary scan interconnect testing of intraboard and interboard nets.
- Control of system clock halt, single-cycle stepping, and system and TAP reset.

Results and Conclusions

The DFT techniques described above, which were championed by the DFT core team, were implemented in several different ASICs with varying degrees of adherence to the DFT rules and methodology. In general, results obtained during prototype chip debug have shown a direct correlation between the level of DFT implementation and the rapidity of test development, chip characterization, and root-cause analysis. For example, while the three memory subsystem ASICs were the last to reach tape release, these chips were the first to reach the operational test release (OTR) and release to manufacturing test (RTPT) checkpoints. The availability of high-quality and comprehensive test sets for these chips enabled chip characterization efforts to be started right away. Furthermore, success in reaching the OTR checkpoint made it possible to transfer the task of testing prototype chips (which are used in the prototype systems) to the manufacturing engineers. This had a very positive effect on resources available to perform chip characterization. In turn, successful completion of this step coupled with efforts of the R&D engineers to improve test coverage enabled the team to reach the RTPT milestone well before any of the other ASICs had reached their OTR checkpoints.

The Asset tool and its customized extensions provided a low-cost system scan access solution with flexible functionality and ease of use. As a commercial tool solution it cut down on development and maintenance costs compared to developing a proprietary toolset and can be reused for future projects.

Acknowledgments

The authors gratefully acknowledge contributions from many members of the Chelmsford DFT group, the DFT core team, and several ASIC designers. In particular, contributions from Ruya Atac, Steven Zedeck, Tom Dickson, Shawn Clayton, Joy Han, Jim Williams, Matt Harline, Dave Malicoat, Ken Pomaranski, and Mick Tegethoff have been key to the successful implementation of DFT in this project.

References

1. B. Dervisoglu, *ASIC DFT Design Rules*, Hewlett-Packard ASD/CSL, November 1992.

2. IEEE Test Technology Technical Committee, *Standard Test Access Port and Boundary Scan Architecture*, IEEE Standard 1149.1/1990, Institute of Electrical and Electronic Engineers, Inc.

3. B. Dervisoglu, *PA 7200 Memory Subsystem TAP/SAP Design*, Hewlett-Packard ASD/CSL, January 1994.

4. M. Ricchetti, "A Structure Independent Built-In Test Architecture for Random Access Memories," *Proceedings of the 1992 Hewlett-Packard Design Technology Conference*, pp. 417-424.

5. H.C. Ritter and T.M. Schwair, "A Universal Test Algorithm for the Self-Test of Parametrizable Random Access Memories," *Proceedings of the Second European Test Conference*, April 1991, pp. 53-59.

6. M. Ricchetti, "Built-In Self-Test of the Memory Controller Data Paths," *Proceedings of the 1993 Hewlett-Packard Design Technology Conference*, pp. 303-312.

7. IEEE Test Technology Technical Committee, *Extended Digital Serial Subset*, IEEE Draft Standard P1149.2/D0.2, Institute of Electrical and Electronic Engineers, Inc.

8. S. Clayton, D. Chou, T. Dickson, D. Montrone, R. Ryan, and D. Schumacher, "The Automation of Standard Cell Block Design for High-Performance Structured Custom Integrated Circuits," *Proceedings of the 1993 Hewlett-Packard Design Technology Conference*, pp. 349-356.

9. S.A. Markinson, "The Application of Scan Synthesis in System ASIC Design," *Proceedings of the 1992 Hewlett-Packard Design Technology Conference*, pp. 449-458.

10. B.I. Dervisoglu and G.E. Stong, "Test and Design-for-Test Standards for the Kodiak Project," *Proceedings of the 1990 Hewlett-Packard Design Technology Conference*, pp. 564-571.

11. B.I. Dervisoglu and G.E. Stong, "Design for Testability: Using Scanpath Techniques for Path-Delay Test and Measurement," *Proceedings of the International Test Conference*, October 1991, pp. 364-374.

12. D. Malicoat, *System Clock Controller ASIC: External Reference Specification, Version 0.91*, Hewlett-Packard Systems Technology Division, November 30, 1992.