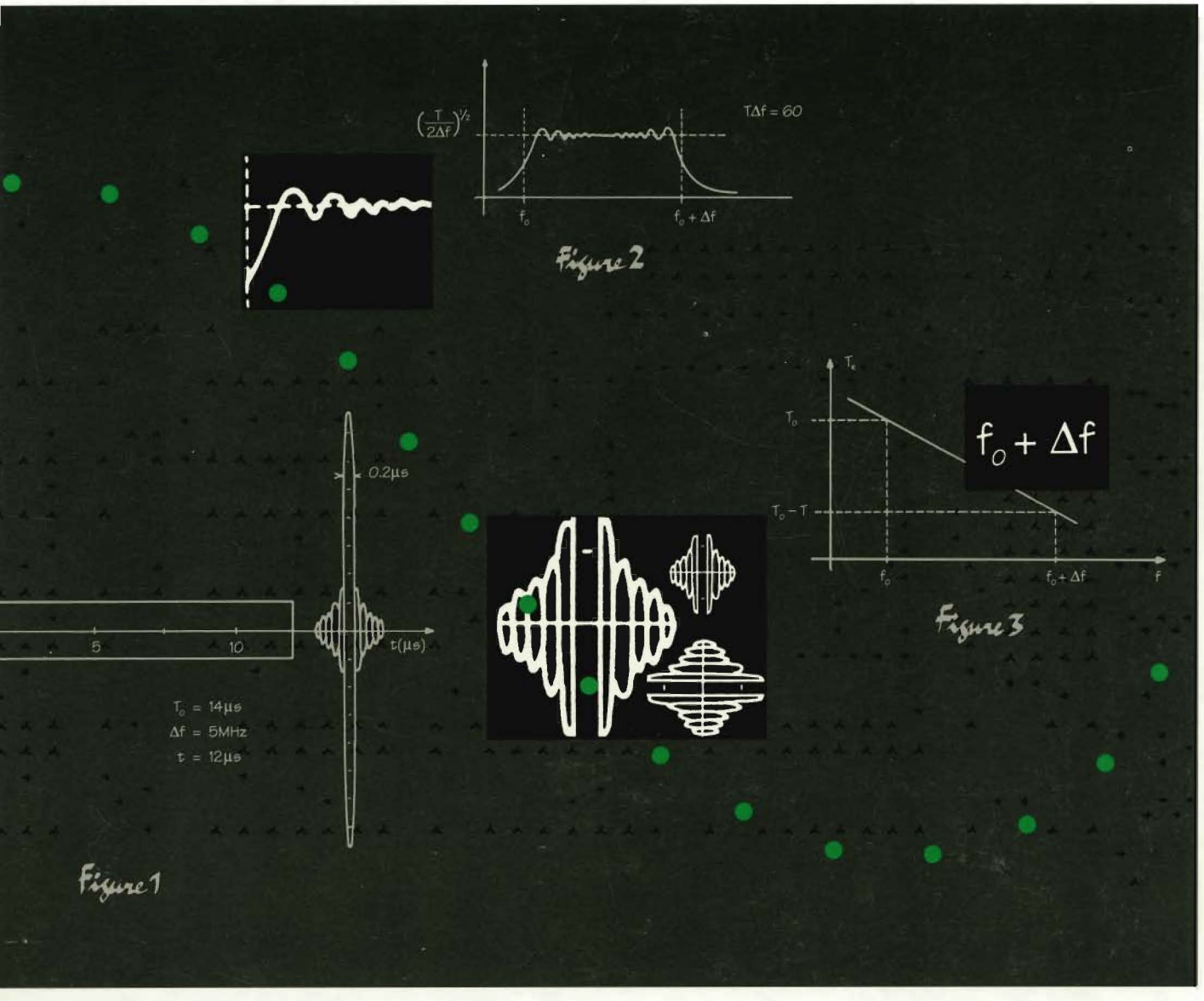
# Digital Technical Journal

Digital Equipment Corporation




Volume 3 Number 4
Fall 1991

The *Digital Technical Journal* is published quarterly by Digital Equipment Corporation, 146 Main Street MLO1-3/B68, Maynard, Massachusetts 01754-2571. Subscriptions to the *Journal* are $40.00 for four issues and must be prepaid in U.S. funds. University and college professors and Ph.D. students in the electrical engineering and computer science fields receive complimentary subscriptions upon request. Orders, inquiries, and address changes should be sent to the *Digital Technical Journal* at the published-by address. Inquiries can also be sent electronically to DTJ@CRL.DEC.COM. Single copies and back issues are available for $16.00 each from Digital Press of Digital Equipment Corporation, 1 Burlington Woods Drive, Burlington, MA 01803-4539.

Digital employees may send subscription orders on the ENET to RDVAX::JOURNAL or by interoffice mail to mailstop MLO1-3/B68. Orders should include badge number, site location code, and address. All employees must advise of changes of address.

Comments on the content of any paper are welcomed and may be sent to the editor at the published-by or network address.



ISSN 0898-901X

Documentation Number EY-H889E-DP



Book production was done by Digital's Database Publishing Group in Northboro, MA.

***Cover Design***

*High-performance screen display of bitonal images is one of the topics in this issue. The handwriting and manually produced technical drawings on our cover are types of images that can be scanned, stored electronically, and then displayed on an X terminal screen; portions of an image can be enlarged or rotated on screen.*

*The cover was designed by Sandra Calef of Calef Associates.*

# Contents

# Editor's Introduction

**Jane C. Blake**
*Editor*

Products designed for quality, high-performance presentation of data in both video and hard-copy form are the topics of papers in this issue of the *Digital Technical Journal*. The design challenges range from managing the huge storage requirements of images for display on X terminals to ensuring high-performance in a feature-rich printer environment.

Image processing is the subject of the opening paper by Chris Payson, Chris Cianciolo, Bob Crouse, and Cathy Winsor. The authors note that one advantage of scanning images for screen display is the input time saved; however, the scanned images and data can consume significant amounts of storage space. They then review the development of an image accelerator board that not only helps solve the problem of storage but also addresses the need for high-performance display—view and manipulation—of bitonal images. In addition to specifics of the board implementation, the authors offer an overview of imaging concepts, terms, and future directions for image accelerators.

The terminal on which the image accelerator board resides is DECimage 1200, an X terminal. X terminals development in general, including a discussion of the VT1200, is the subject of a paper by Björn Engberg and Tom Porcher. Björn and Tom focus their discussion on a comparison of the X terminal and X workstation environments, and explain why X terminals are a low-cost alternative. The authors present the design choices debated by the engineers during the development of Digital's X terminals, including the selection of a hardware platform, terminal and window management, X server, communications protocols, and font file systems.

Video terminal and workstation users need the assistance of a number of I/O devices, such as keyboards, mice, and tablets, all of which may not be made by the same company. A new open desktop bus, described by Peter Sichel, is a simple means to connect as many as 14 low-speed devices to a desktop system. In his paper, Peter presents the project background, reviews the $I^2C$ technology on which the bus is based, and describes the protocol and the configuration process.

Hard-copy presentation of data and recent developments in printer technologies are the topics of the next three papers. Rick Landau and Alan Guenther review the DECprint Printing Services, which is software that controls numerous printer features for a wide range of printers. Also called a common print symbiont, this component of the VMS printing system supports several page description languages, handles multiple media simultaneously, and uses different I/O interconnections and communication protocols.

Both DECprint Printing Services and the subject of the next paper, the common printer access protocol, are part of the DECprint architecture. The CPAP provides the fundamental services necessary for the presentation of data at the printer. Jim Jones, Ajay Kachrani, and Tom Powers describe the challenges of developing a protocol that operates in a heterogeneous, internetworking environment and that also ensures backward compatibility with older products. Their success in developing a high-performance protocol is evidenced by OSF acceptance of CPAP for inclusion in a future release of OSF/1.

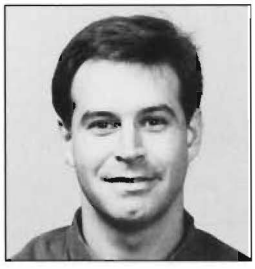
As was the case with the CPAP, performance was also key in the development of the turbo PrintServer 20 controller. Guido Simone, Jeff Metzger, and Gary Vaillette explain that the requirements of complex documents demanded turbo controller performance that was five to eight times that of the current controller. To aid them in making design decisions, a performance analysis tool, RETrACE, was created and is described here. Authors also relate how they used existing chips in order to keep development costs low and still deliver a high-performance controller.

The editors thank Liz Griego-Powell of the Video, Image and Print Systems Group for her help in preparing this issue.

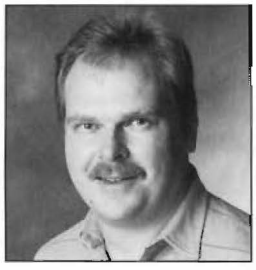
Jane Blake

## *Biographies*

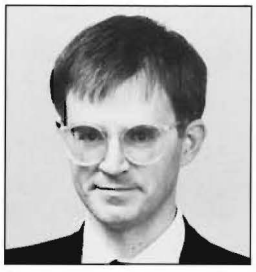
**Christopher J. Cianciolo** As a hardware design engineer in the Video, Image and Print Systems Group, Chris Cianciolo is currently working on the design for the group's latest imaging product. Chris joined Digital in 1985 after participating in a co-op session in the Power Supply Engineering Group. He also participated in co-op sessions for Charles Stark Draper Laboratory, Inc. on a fiber-optic missile guidance system project. He received his B.S.E.E. from Northeastern University in 1988 and is currently pursuing an M.S.E.E., also from Northeastern.



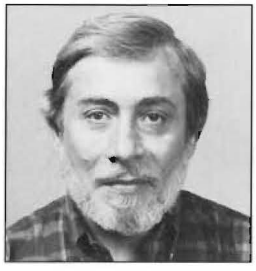
**Robert N. Crouse** Senior engineer Bob Crouse is a member of the Video, Image and Print Systems Group. He is currently working on the advanced development of new imaging technology for X window terminals. Bob was project engineer for the development of a bitonal imaging accelerator for a low-end VAXstation workstation. As a member of the Electronic Storage Development Group, he designed a double-bit error detection and correction circuit for a VAX mainframe. Bob received his B.S.E.E. from Northeastern University and holds one patent.



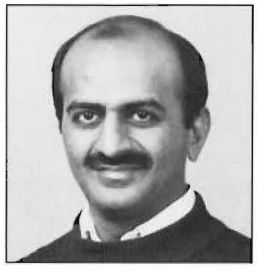
**Björn Engberg** As a principal software engineer in the Video, Image and Print Systems Group, Björn Engberg was the main architect and software project leader for the VT1000 and VT1200 X window terminals. He joined Digital in 1978 and worked as a development engineer at CSS in Sweden, where he modified Digital's terminals for the European market. He relocated to the United States in 1982 to work on the VT240, the VT320, the LJ250, and several advanced development projects. Björn received an M.S.E.E. (honors) from the Royal Institute of Technology in Stockholm.



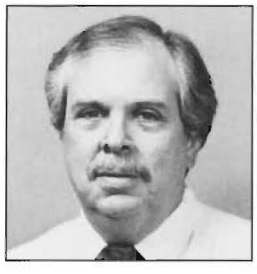
**A. Alan Guenther** As a member of the technical staff in the DECprint System Software Group, Alan Guenther is involved in the ongoing design and implementation of the DECprint common print symbiont. Prior to this, he was the primary designer and implementor of the distributed queuing services. Alan has worked at Digital since 1973, both as a full-time employee and as an independent consultant (from 1982 to 1990). After receiving a B.S. (honors, 1970) from the University of Montana, he worked at the university until he joined Digital.



**James D. Jones** Jim Jones is a principal engineer in the Hardcopy Systems Engineering Group. He joined Digital in 1974 and was part of a team developing diagnostic programs for the DECsystem-10 and DECSYSTEM-20 systems. After running his own software business for five years, Jim rejoined Digital to design printer controllers and software. Most recently, he provided software for the PrintServer products, authored the Common Printer Access Protocol specification, and is helping to define the next generation of network printers. Jim is a member of IEEE and ACM and participates in the IETF.



**Ajay P. Kachrani** Principal software engineer Ajay Kachrani currently works on the OSF/1 socket and XTI kernel interfaces and security project. Previously, he led the development of the overall PrintServer software version 4.0 with dual network protocol support (DECnet and TCP/IP), from inception through field test. Ajay presented the CPAP protocol as an Internet standard to the IETF and added PrintServer support in version 1.0 of the Palladium Print System at MIT/Project Athena. Ajay holds a B.S.E.E. (honors) from the University of Mysore, India, and an M.S.C.S. from the University of Lowell.

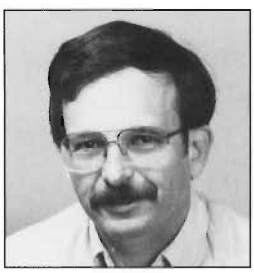


**Richard B. Landau** Richard Landau is the DECprint program manager for the Video, Image and Print Systems Group. Working to improve the interaction of printing software and hardware, he initiated the DECprint, Font, and PostScript programs. Prior to this, Rick was the program and development manager for the VAX DBMS, DATATRIEVE, CDD, and Rdb/VMS products and for the relational database architecture. Before joining Digital in 1974, Rick was an independent consultant and was also employed by Applied Data Research, Inc. He holds A.B. (cum laude, 1969) and M.A. (1973) degrees in statistics from Princeton University.

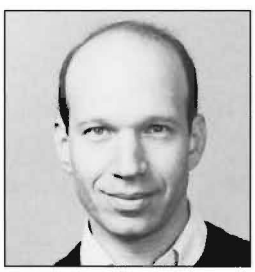
**Jeffrey A. Metzger** Presently a senior engineer, Jeff came to Digital as a co-op student in 1983, working first in the Semiconductor Engineering Group and then in Hardcopy Engineering. He became a full-time employee after graduating from Cornell University in 1985. He introduced Hardcopy to system-level logic simulation, contributed to the hardware, software, and firmware development of the PrintServer 20, and developed RETrACE, which is used to characterize the execution behavior of PrintServer systems. Jeff is currently working in the Entry Systems Business Group on a next-generation processor product.

**Christopher J. Payson** Chris Payson joined Digital as a hardware design engineer in 1989 after five co-op terms. He is currently working on XIE software and image hardware accelerators. Chris previously worked on performance testing, diagnostics, logic design, and demonstration software, all associated with imaging. He is coapplicant for a patent related to an image clipping algorithm and hardware logic. Chris received a B.S.C.E. from Rochester Institute of Technology with highest honors and is currently pursuing an M.S.C.E. from Northeastern University.

**Thomas C. Porcher** Principal engineer Tom Porcher is a member of the Video, Image and Print Systems Group. He provided technical leadership in the development of the VXT 2000 X terminal. Previously he was a technical leader for the VT240 terminal, VAX Session Support Utility, and the DECterm terminal emulator. Tom holds five patents for work on the VT240 terminal and on the multisession protocol used in the VT340 and VT400 series terminals. Tom received his B.S. in mathematics from Stevens Institute of Technology (1975). He is a member of the ACM.



**Thomas E. Powers** As a consultant engineer in the Hardcopy Engineering Firmware/Software Group, Tom Powers is a vendor liaison for desktop PostScript printer products. He chairs the DECprint PAP Architecture Team and was a contributor to the PrintServer 40 internal hardware/firmware architecture. Tom represented Digital on American and international standards committees on computer graphics from 1979 to 1989. He led several firmware teams and is coinventor of the ReGIS Graphics Protocol. Tom has a B.S.E.E. from Tufts University and an M.S.E.C.E. from the University of Massachusetts at Amherst.



**Peter A. Sichel** As a principal software engineer in the Video Terminals Architecture Group, Peter Sichel led the development of the ACCESS.bus architecture and device protocol specifications, in addition to writing the initial ACCESS.bus device firmware. He worked on the VT420 video terminal and the DECterm DECwindows terminal emulator, and helps maintain Digital standards for video terminals and keyboards. Peter joined Digital in 1981 after receiving B.S. and M.S. degrees in computer engineering from the University of Michigan.

**Guido R. Simone** Guido Simone is a principal engineer in the Print Systems Engineering Group and was the project leader and architect for the turbo PrintServer 20 controller. He is currently working on the development of a new print system architecture to be used with advanced printing technologies. In previous work, Guido was the project leader and architect for an rtVAX 78R32 CPU chip-based laser printer controller. Before joining Digital in 1980, he received a B.S. in electrical engineering from Rensselaer Polytechnic Institute.

**Gary P. Vaillette** Senior hardware engineer Gary Vaillette has been involved in the design and implementation of printing system hardware since joining Digital in 1983. His current work includes performance characterization of PostScript printers and PrintServer products, and hardware implementation of CCITT decompression in the turbo PrintServer 20 product. Previously, Gary worked at Data General Corporation and helped to develop their token bus network product. He holds an A.A.E.E (1974) from Quinsigamond Community College and expects to receive a B.S.C.S. (May 1992) from Boston University.

**Catherine F. Winsor** As a senior engineer in the Video, Image and Print Systems Group, Cathy Winsor has worked on image accelerators. As the project leader for the DECimage 1200 hardware and the image utility library software, Cathy was involved in the planning and development of an image-capable VT1200. She is currently leading the project to support imaging on the next generation of Digital's X terminals. The project includes an image accelerator board and XIE software. Cathy received an A.B. in engineering sciences from Dartmouth College and a B.S.E.E. from the Thayer School of Engineering.

# Foreword

**Larry Cabrinety**
*Vice President,*
*Video, Image and Print*
*Systems Group*

For the millions of people worldwide who use Digital's computer equipment, the computer is not the sophisticated system in the back room, or the complex network. It is the equipment they use each day—the terminal or monitor, keyboard and mouse, desktop printer or network printer system.

Today's users demand products with high levels of usability and superior ergonomic features. Digital's products set worldwide standards for the user interface to computer systems. In the 90s our focus is to offer products that operate in multivendor environments with the goal of delivering a complete computing solution. In this issue you will read about some of the Video, Image and Print Systems (VIPS) Group's products and technologies that support network computing and standards-based environments.

Digital entered the video terminal market in 1975 with the VT52 for its time-sharing users. Its replacement, the VT100, embodied two important principles—the use of standards in data communications interchange and the protection of customer investments through backward compatibility of new generations of products. The VT220, introduced in 1983, and the cost-effective VT320 terminals saw the addition of functionality and ergonomic features which established Digital as a leader in the commodities market.

In March 1990, Digital entered the X terminal market with the introduction of the VT1000, followed by the VT1200 and VT1300 terminals later that year. The emergence of MIT's X Window Systems as the accepted industry standard for windowing systems provided a standards-based environment for distributed applications display processing. The X terminal user can now benefit from the graphical user interface, sophisticated applications, and standards of performance previously available only on workstations. X terminals run X11 server code which is operating system independent and ideally suited for heterogeneous, network-based computing environments. In this issue you will read about the engineering decisions made as the X terminals were developed.

There is a growing need in the industry to have imaging applications run alongside conventional text and graphics applications. Technical documentation is an example of this. Imaging applications, however, have special requirements to achieve acceptable end-user performance. Although the X11 software can handle images as bit-map data, software and hardware assistance is required to achieve acceptable performance. Digital has designed DECimage hardware accelerators for rapid processing of image data. This technology is included in the DECimage 1200 and will be incorporated in following generations of X terminals. To make this possible, Digital developed extensions to the X server software that support the high-speed transport and display of image data. To assure open standards, the extensions have been proposed to MIT for incorporation into releases of the X11 server software.

In November 1990, Digital announced its next generation of X terminals. The VXT2000 terminal provides virtual memory and supports both a traditional host-based model with software downloaded to the terminals as well as the server style of X terminal computing.

The VXT2000 terminal was designed to support TCP/IP and LAT protocols, and further demonstrates our commitment to openness and support for customers' multivendor environments. This same philosophy is seen in our printer products and our open desktop bus.

Digital pioneered the distributed printing business with networked laser printers. This product area began when we combined two concepts which had not been combined before—mid-range laser printers and networks. In the mid-1980s most large-scale computing was done on mainframe computers with large printers attached directly to these systems. Typically these dedicated printers were only accessible to users on that particular system. Digital's distributed computing provided an alternative to the mainframe. By combining the power of multiple systems in clusters or on networks, a new distributed large system was created.

A printing solution was needed to effectively work in this new distributed computing environment. The PrintServer series addressed this need.

PrintServer products enabled printing resources to be directly connected to networks for the first time, and since they were on the network and not tied to any one system, they were accessible by all systems on those networks. They enabled the complex printer functionality previously found only in dedicated mainframe printers to be distributed throughout end-user environments.

As these mid-range printers migrated out of the computer room and into the office, new demands for functionality were created. Large groups of users brought many different requirements for printing, and our goal was to satisfy as many as possible in a single PrintServer. For example, some people need "A" size paper for office correspondence, while others may need "B" size paper for CAD/CAM or accounting work, and still others need transparencies for presentations. The PrintServer is flexible enough to have all of these different types of media available and offer both simplex and duplex printing.

In 1985 when Digital was first developing the PrintServer, there was no industry standard way of describing the contents of a page to a printer. Each major vendor had its proprietary language, and none offered the compatibility necessary to achieve our print system vision. Our goal was to create a family of products, from large to small, that offered compatibility for all applications. To achieve this goal we had to select a protocol that would enable us to print any file on any printer. At that time Adobe Systems, the developer of PostScript, was a small start-up company in Silicon Valley. PostScript was not a standard, and in fact, only a single PostScript laser printer model had been shipped, the original Apple LaserWriter. Our technical community felt PostScript was the best solution to our needs, and at that point Digital committed to adopting PostScript as our strategic page description language. PostScript printers and PostScript application support are now pervasive throughout the industry and standard printing protocols enable interactive communication with hosts on the network.

Significant advances have taken place in the PrintServer series over the past seven years. An entire MicroVAX II system was housed within the original PrintServer 40, along with custom hardware acceleration boards developed by the Hardcopy Group to enable printing at 40 pages per minute. In this issue you will read about the single-board controller that replaces the MicroVAX II and offers far more processing power. Using the latest system-on-a-chip technology, our new turbo board provides leadership performance for our printers. The CCITT image decompression chip enables us to provide full-speed image printing to our customers as the image market develops.

The first PrintServer systems supported printing from VMS hosts over DECnet networks. Since then the breadth of platform support has increased to include first ULTRIX systems and then UNIX operating systems. A software kit for Sun systems will be available soon. In expanding PrintServer connectivity to include UNIX systems and TCP/IP networks, we again faced the problem that no network printing protocol existed for TCP/IP. With the help of Digital's experts at the Western Research Laboratory, we were able to develop a solution. In this issue, we discuss the creation of a network printer access protocol for TCP/IP. Today this network protocol is a proposed standard at the Internet Engineering Task Force, the body controlling the TCP/IP protocol.

The development of the ACCESS.bus product has brought an easy, standard way to link a desktop computer to many interactive user interfaces. This open desktop bus is currently implemented on the Personal DECstation 5000 workstation, and implementations on future RISC workstations and video terminals is underway. Developers of Digital's products will continue to place a high priority on open standards. The papers included in this issue of the *Digital Technical Journal* will provide insight into the key areas of technology used in the design and development of VIPS products.

*Christopher J. Payson*
*Christopher J. Cianciolo*
*Robert N. Crouse*
*Catherine F. Winsor*

# *Hardware Accelerators for Bitonal Image Processing*

*Electronic imaging systems transfer views of real-world scenes or objects into digital bits for storage, manipulation, and viewing. In the area of bitonal images, a large market exists in document management, which consists of scanning volumes of papers for storage and retrieval. However, high scan densities produce huge volumes of data, requiring compression and decompression techniques to preserve system memory and improve system throughput. These techniques, as well as general image processing algorithms, are compute-intensive and require high memory bandwidth. To address the memory issues, and to achieve interactive image display performance, Digital has designed a series of bitonal image hardware accelerators. The intent was to create interactive media view stations, with imaging applications alongside other applications. In addition to achieving memory, performance, and versatility goals, the hardware accelerators have significantly improved final image legibility.*

Bitonal image technology, which can be viewed as the electronic version of today's microfilm method, is experiencing a high rate of growth. However, the electronic image data objects generated and manipulated in this technology are very large and require intensive processing. In a generic system, these requirements can result in poor image processing performance or reduced application performance. To address these needs, Digital has designed a series of imaging hardware accelerators for use in the document management market.

This paper provides a brief tutorial on electronic imaging. It begins with a general description of the imaging data type and compares this type to the standard text and graphics data types. It continues with a discussion of specific issues in bitonal imaging, such as image data size, network transport method, rendering speed, and end-user legibility. The paper then focuses on Digital's DECimage 1200 hardware accelerator for the VT1200 X window terminal developed by the Video, Image and Print Systems Group. It concludes with future image accelerator demands for the processing of multimedia applications and continuous-tone images.

## *Introduction to Imaging*

Just as graphics technology blossomed in the 1980s, electronic imaging and its associated technologies should come of age in the 1990s. Digital imaging is already in use in many areas and new applications are being created for both commercial and scientific markets. The emergence of digital images as standard data types supported by the majority of systems (like text and graphics of today) seems assured. For a greater understanding of specific imaging applications, this section presents general imaging concepts and terms used throughout the paper.

### *Concepts and Terms*

In its simplest form, imaging is the digital representation of real-world scenes or objects. Just as a camera transfers a view of the real world onto a chemical film, an electronic imaging system transfers the same view into digital bits for storage, manipulation, and viewing. In this paper, the term image refers to the digital bits and bytes that represent the real-world view.

The process of digitizing the view may be done through various methods, e.g., an image scanner or image camera. A scanner is the conceptual inverse of a normal printer. A printer accepts an electronic stream of bits that describe how to place the ink on the paper to create the desired picture. Conversely, optical sensors in the scanner transform light intensity values reflected from a

sheet of paper and create a stream of electronic bits to describe the picture. Similar sensors in the focal plane of a camera produce the other common digitization method, the electronic image camera.

The format of a digitized image has many parameters. A pixel is the common name for a group of digitized image bits that all correspond to the same location in the image. This pixel contains information about the intensity and color of the image at one location, in a format that can be interpreted and transformed into a visible dot on a display device such as a printer or screen. The amount of information in the pixel classifies the image into one of three basic types.

- A bitonal image has only one bit in each pixel; the bit is either a one or a zero, representing one of two possible colors (usually black and white).
- A gray-scale image has multiple bits in each pixel, where each pixel represents an intensity value between one color (all zeros) and another color (all ones). Since the two colors are usually black and white, they produce a range of gray-scale values to represent the image.
- A color image has multiple components per pixel, where each component is a group of bits representing a value within a given range. Each component of a color image corresponds to a part of the color space in which it is represented. Color spaces may be thought of as different ways of representing the analog, visible range of colors in a digitized, numeric form. The most popular color spaces are television's YUV format (one gray-scale and two color components) and the bit-mapped computer display's RGB format (red, green, and blue components).

The resolution of an image is simply the density of pixels per unit distance; the most common densities are measured in dots per inch (dpi), where a pixel is called a dot. For example, a facsimile machine (which is nothing more than a scanner, printer, and phone modem in the same unit) typically scans and prints at 100 dpi, although newer models are capable of up to 400 dpi. As another example, most workstation display monitors are capable of 75- to 100-dpi resolution, and some high-end monitors achieve up to 300-dpi resolution.

To display an image at a density different from its scanned density, without altering the image's original size, requires the image to be scaled, so that the new image density matches the output media density. Scaling an image may be as simple as replicating and dropping pixels, or it may involve interpolation and other algorithms that take neighboring pixels into account. Generally, the more complex scaling algorithms require more processing power but yield higher-quality images, where quality refers to how well the original scene is represented in the resulting image.

Before an image can be displayed, its pixel values often require conversion to account for the characteristics of the display device. As a simple example, a color image cannot retain its color when output to a black-and-white video monitor or printer. In general, when a device can display fewer colors than an image contains, the image pixel values must be quantized. Simple quantizing, or thresholding, can be used to reduce the number of image colors to the number of display colors, but can result in loss of image quality. Dithering is a more sophisticated method of quantizing, which produces the illusion of true gray scale or color. Although dithering need use no more colors than simple quantizing, it results in displayed images of much higher quality.

Image compression is a transformation process used to reduce the amount of memory required to store the information that represents the image. Different compression methods are used for bitonal images than those used for gray-scale and color images. These methods are standardized to specify exactly how to compress and decompress each type of image. For bitonal images, the most common standards are the ones used in facsimile machines, i.e., Recommendations T.4 and T.6 of the Comité Consultatif Internationale de Télégraphique et Téléphonique (CCITT).[1,2] Commonly known as the Group 3 and Group 4 standards, the designations are often shortened to G3-1D, G3-2D, and G4-2D, referring to the particular standard group and to the coding method, which may be either one- or two-dimensional. For gray-scale and color images, the Joint Photographic Experts Group (JPEG) standard is now emerging as a joint effort of the International Standards Organization (ISO) and CCITT.[3] Whichever format or process is used, compression is a compute-intensive task that involves mathematically removing redundancy from the pixel data.

A typical compression method creates an encoded bit stream which cannot be displayed directly; the compressed bits must be decompressed before anything recognizable may be

displayed. The term compression ratio represents the size of the original image divided by the size of the compressed form. For bitonal images using the CCITT standards, the ratio is commonly 20:1 on normal paper documents, but can vary widely with the actual content of the image. The CCITT standards are also "lossless" methods, which means that the decompressed image is guaranteed to be identical to the original image (not one bit different). In contrast, many "lossy" compression methods allow the user to vary the compression ratio such that a low ratio yields a nearly perfect image reproduction and a high ratio yields a visible degradation in image quality. This trade-off between compression and image quality is very useful because of the wide range of applications in imaging. An application need pay no more in memory space and bandwidth than necessary to meet image quality requirements.

### *A New Data Type and Its Features*

The image data type is fundamentally different from text and graphics. When a user views characters or pictures on a display device, the source of that view is usually not important. A sheet of text from a printer may have come from either a text file where the printer's own fonts were used, a graphics file where the characters were drawn with line primitives, or an image file where the original text document was scanned into the system. In any case, the same letters and words present the user with approximately the same information; the differences are mostly in character quality and format.

In spite of their large storage space requirements, images have several advantages over graphics or text. First, consider the process of getting the information into the computer. With the imaging process, documents may be scanned automatically in a few seconds or less, compared to the time required for someone to type the information correctly (absolutely no errors) into a text file. Also, even though the software exists to convert electronic raster images into graphic primitive files, the process loses detail from the original image and is relatively slow. Next, consider the variety of information possible on a sheet of paper: a user cannot easily reproduce a diagram or a signature on a document. A scanned image preserves not only the characters, but their font, size, boldness, relative position, any pictures on the page, and even smudges or tears depending on the quality of the image scan.

The major drawback in the imaging process is increased data size, which results in storage memory and network transport problems. High scan densities and color information components create large volumes of data for each image; a bitonal image scanned at 300 dpi from an 8.5-by-11-inch sheet of paper requires over 1 megabyte of memory in its original pixel form. Therefore, compression and decompression are integral parts of any imaging system. Even in compressed form, a bitonal image of a text page requires about 50 kilobytes of storage, whereas its American standard code for information interchange (ASCII) text equivalent requires only 4 to 5 kilobytes. Similarly, a graphics file to describe a simple block diagram is much smaller than its scanned image equivalent.

Based on these advantages and limitations, several applications have emerged as perfect matches for imaging technology. Bitonal images are used in the expanding market of document management, which consists of scanning volumes of papers into images. These images are stored and indexed for later searching and viewing. Basically an electronic file cabinet, this system results in large savings in physical cabinet space, extremely fast document access, and the ability for multiple users to access the same document simultaneously. Gray-scale imaging is often used in medical applications. Electronic versions of X rays can be sent instantly to any specialist in the world for diagnosis, and the ordering of sequential computer-aided testing (CAT)-scan images into a "volume" can provide valuable three-dimensional views. The applications for color imaging are relatively new and still emerging, but some are already in use commercially, e.g., license and conference registration photographs. A further extension to still imaging is digital video, which can be considered as a stream of still images. In conjunction with audio, digital video is commonly known as multimedia, applications for which range from promotional presentations to a manufacturing assembly process tutorial.

In this paper, we focus on the static bitonal imaging method of representing real-world data inside computers. Static imaging is a simpler method of representing a broader range of information than the text and graphics media types, but it carries a greater requirement for processing power and memory space. In addition, static imaging can be viewed as one part of true multimedia, as can text, graphics, audio, video, and any other media formats. Yet static imaging does not have the system

speed requirements of a motion video and audio system, which must present data at real-time rates. As long as the user can deal with static images at an interactive rate, i.e., being able to view the images in the format of choice as fast as the user can select them, then static imaging is a powerful media presentation tool. The next section presents the important issues concerning bitonal imaging in a document management environment.

## *Bitonal Imaging Issues*

As previously mentioned, bitonal electronic imaging as an alternative to paper documents offers many benefits, such as reduced physical storage space, instant and simultaneous access of scanned images, and in general a more accessible media. Serious issues need to be resolved before a productive imaging operation can be implemented. The chief issues are the image data size, transport method, perceived rendering speed, and final legibility. In the following sections, we examine each issue and present solutions.

### *Digitized Image Data Size*

The most important issue concerns image data size. Images are typically documents, drawings, or pictures that have been digitized into a computer-readable form for storage and retrieval. Depending on the dot density of the scanner, a single image can be 1 to 30 megabytes or more in size. However, storing a single image in its scanned form is not the typical usage model. Instead, a company may have tens of thousands of scanned documents. Clearly, with today's storage technologies, a company cannot afford to store such a large volume of images in that format.

A typical ASCII file representing the text on an 8.5-by-11-inch sheet of paper requires approximately 3 kilobytes of memory. If the same sheet of paper is digitized by scanning at various dot densities, the resulting data files are huge, as shown by the decompressed bitonal image sizes in Table 1. Note that Table 2 includes the size of the scanned image if scanned in gray-scale and color modes, although using these modes would not make sense on a black-and-white sheet of paper. The image sizes are included for comparison and are discussed in the section Future Image Accelerator Requirements. The data presented in Tables 1 and 2 illustrates that the size of the original ASCII file is much smaller than any of the scanned versions. The data also gives evidence that scanned images, in general, require considerable memory.

Since the typical use for bitonal images is for volume document archival, an imaging application must include a compression process to reduce memory usage. This process must transform the original scanned image file to a much smaller file without losing the content of the original scanned data.

Compression algorithms may take different paths to achieve the same result, but they share one basic process, the removal of redundant information to reduce the object size. A common compression routine searches the pixel data for groupings, or "run lengths," of black or white pixels. Each run length is assigned a code significantly shorter than the run length itself. The codes are assigned by statistics, where the most frequent run lengths are assigned the shortest codes; statistics have been amassed on a variety of document types for different scan densities and document sizes. A compression process parses through the original image file, generating another file that contains the codes representing the original image. Figure 1, a sample bitonal image compression, illustrates these compressed codes in a serial bit stream.

**Table 1 Sample Bitonal Image Sizes**

| Document Type (Paper Size) | Scan Density (dpi) | Kilobytes of Data | |
|---|---|---|---|
| | | Pixel Form (Decompressed) | Typical Compressed |
| A size | 100 | 114 | 46 |
| (8.5 × 11 inch) | 200 | 457 | 47 |
| | 300 | 1027 | 50 |
| E size | 100 | 1826 | 106 |
| (44 × 34 inch) | 200 | 7305 | 114 |
| | 300 | 16436 | 127 |

**Table 2 Sample Gray-scale and Color Image Sizes**

| Document Type and Size | Kilobytes of Data in Pixel Form (Decompressed) |
|---|---|
| 128 × 128 pixel, 12 bits per pixel gray-scale image | 24 |
| 512 × 512 pixel, 8 bits per pixel color image | 256 |
| 512 × 512 pixel, 24 bits per pixel color image | 768 |
| 8.5 × 11 inch, 100 dpi, 24 bits per pixel, color image | 2740 |

Several algorithms for bitonal compression are widely used today. As mentioned in the previous section, the most common for bitonal images are the CCITT standards G3-1D, G3-2D, and G4-2D, which all use the approach just described. For the one-dimensional method, the algorithm creates run lengths from all pixels on the same scan line. In the two-dimensional methods, the algorithm sometimes creates run lengths the same way, but the previous scan line is also examined. Some codes represent run lengths and even whole scan lines as "the same as the one in the previous scan line, except offset by $N$ pixels," where $N$ is a small integer. The two-dimensional method takes advantage of most of the redundancy in an image and returns the smallest compressed file. In addition to preserving system memory, these compression methods significantly improve network transport performance.

## *Network Transport Constraints*

The network transport performance for an image is important, because images are most often stored on a remote system and viewed on a widespread group of display stations. For example, one group in an insurance company receives and scans claim papers to create a centralized image database, while users in another group access the documents simultaneously to process claims. For the imaging system to be productive, this image data needs to be transported quickly from one group to the other: telephone attendants answering calls must have immediate access to the data.

Scanned image documents take a long time to transport between systems, simply because they are so large. When compression techniques are used, a typical uncompressed image stored in 1 megabyte can be reduced to approximately 50 kilobytes. Since transport time is proportional to the number of packets that must be sent across the network, reducing the data size to 5 percent of its original size also reduces the transport time to 5 percent of the original time. Therefore, you can now send twenty compressed images in the same time previously spent sending one uncompressed image.

Even with compression techniques, the image files are still larger than their text file equivalents. Moreover, most network protocols limit their packet size to a maximum number of bytes, i.e., an image file larger than the maximum packet size gets divided over multiple packets. If the protocol requires an acknowledgment between packets, then the transport of a large file over a busy network becomes a lengthy operation.

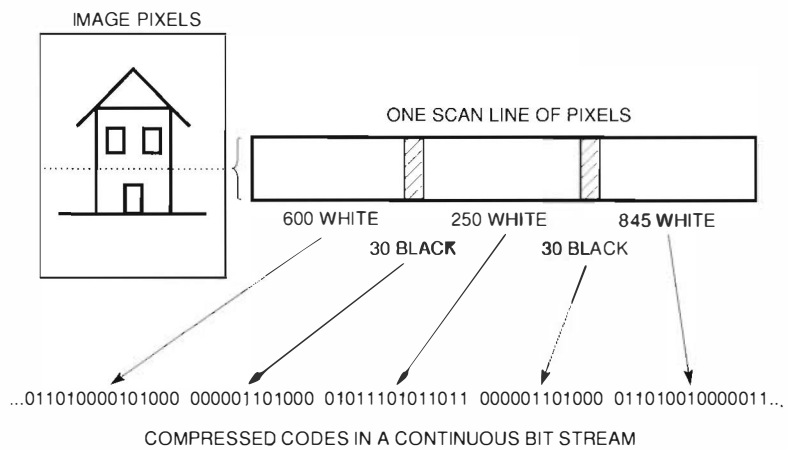



*Figure 1 Bitonal Image Compression*

The platform for our most recent accelerator is the VT1200 X window terminal, which uses the local area transport (LAT) network protocol. We soon realized that the X server packet size was limited to 16 kilobytes and the typical A-size compressed document was approximately 50 kilobytes. With this arrangement, each image transport would have required four large data packets and four acknowledgment packets. Working with the X Window Terminal Base System Software Group, we were able to raise the packet size limit to 64 kilobytes. The base system group also implemented a delayed acknowledgment scheme, which eliminates the need for the client to wait for an acknowledgment packet before sending the next data packet. Table 3 shows compressed image data taken during the DECimage 1200 development cycle. Notice that the network transport times for Digital document interchange format (DDIF) decrease sharply after the packet changes.

### *Perceived Rendering Speed*

Because the image scanning and compression operations occur only once, they are not as performance-critical as the decompression and rendering for display operations, which are done many times. Decompression and rendering are part of the system's display response time, which is a critical factor in a system designed for high-volume applications that access thousands of images daily. This time is measured from the instant the user presses the key to select an image to view, to the moment the image is displayed completely on the screen. The display response time is a function of the disk read time, network transport time, and display station render time.

Although network transport time and disk file read time have a direct effect on the response time, accelerator developers rarely have any control over them. The disk access time data from the DECimage project analysis shown in Table 3 demonstrates that the disk file read time is a significant portion of the overall response time. Thus, the display station render time is the only area of the display response time which can be clearly influenced and is, therefore, the main focus of our image accelerators. The local processing that must occur at the display station is not a trivial task; an image must be decompressed, scaled, and clipped to fit the user's current window size, and optionally rotated.

The decompression procedure inverts the compression process; both are computationally complex. Input to the procedure is compressed data, and output is the original scan line pixel data, which can be written to a display device. Scaling the data to fit the current window or fill a region of interest is not trivial either: a huge input data stream must be processed (the decompressed, original file), and a moderate output data stream must be created (the viewable image to be displayed). While simple pixel replicate and drop algorithms may be used to scale the data, a more sophisticated scaling algorithm has been shown to greatly enhance the output image quality.

In addition to scaling and clipping, the orthogonal rotation of images (in 90-degree increments) is a useful function on a display station. Some documents may have words running in one direction while pictures are oriented another way, or the user may wish to view a portrait-mode image in landscape mode. In either case, orthogonal rotation can help the user understand the information; i.e., the increased time to rotate the view is warranted.

When an image is scanned, particularly with a hand-held scanner, the paper is never perfectly aligned. Thus, the image often requires a rotation of 1 to 10 degrees to make the view appear straight in the image file. However, multiple users want the

**Table 3 DDIF Image File Read Time and File Transport Performance**

| Image Size (kilobytes) | Disk Read Time (milliseconds) | | | Network Transport Time (milliseconds) | |
|---|---|---|---|---|---|
| | MicroVAX II | VAX 8800 | VAX 6440 | After Packet Change | Before Packet Change |
| 19 | 1223 | 480 | 281 | 325 | 960 |
| 41 | 1534 | 655 | 332 | 614 | 1792 |
| 99 | 2351 | 1035 | 598 | 1351 | 3928 |
| 157 | 3288 | 1380 | 716 | 2283 | 6430 |

information from the document as quickly as possible, and should not have to rotate the image by a few degrees to make it perfectly straight on the screen. Therefore, this minimal rotation should be done after the initial scanning process; i.e., only once, prior to indexing the material into the database, and not by every user in a distributed environment. Because any form of rotation is compute-intensive, allowing the user to perform minimal rotations at a high-volume view station would reduce the application's perceived rendering speed and add little value to the station's function.

### *Final Legibility*

While the primary issue facing imaging applications is data size, image viewing issues must also be addressed. In short, an effective bitonal imaging display system must be responsive to overall image display performance and the resulting quality of the image displayed. To enhance our products, we optimized the display performance parameters as best we could, given that some parameters are not under our control. Improvements to monitor resolution and scanner densities continue to increase the legibility of images. An affordable image system should increase the image legibility by rendering a bitonal image into a gray-scale image using standard image processing techniques. We discuss the method used in our accelerators, i.e., an intelligent scale operation in the hardware pipeline, in the next section.

## *Hardware Accelerator Design*

As explained in the previous section, transforming documents into a stream of electronic bits is not the demanding part of a bitonal imaging process for document management. Also, scanners and dedicated image data-entry stations abound in the marketplace already. Instead, the challenge lies in: (1) managing the image data size to control memory costs and reduce network slowdown; (2) increasing the image rendering speed, i.e., decompress the image, scale it, and clip it to fit the window size with optional rotation; and (3) increasing the quality of the displayed images. This section describes the way our strategy influenced the design of DECimage products. We also discuss the chips used for decompression and scaling, and how Digital's existing client-server protocols support these imaging hardware accelerators.

### *General Design Strategy*

The number of applications using bitonal image data continues to increase. In general, these applications attempt to offer low cost while achieving an interactive level of performance, defined as no more than 1 second from point of request to complete image display. Ultimately, software may provide this functionality without hardware acceleration, but today's software cannot. Moreover, the parameters of image systems are not static; scan densities, overall image size, and the number of images per database will all increase. These increases will provide the most incentive for hardware assist at the low end of the X window terminals market, because software alone cannot perform the amount of processing that users will expect for their investment.

*The User Model* Although a single model cannot suit every application, imaging is centered on certain functions. Therefore, a user model built on these functions would be very useful in mapping individual steps to the hardware: hardware versus software performance, the function's frequency of use, and the cost of implementation.

The general user model for bitonal imaging systems is relatively simple. A small market exists for image entry stations, in which documents are scanned, edited, and indexed into a database. While a high throughput rate is important at these stations, a general-purpose image accelerator is not the solution—dedicated entry stations already exist in the market. Instead, we designed a general-purpose platform, or versatile media view station, to be used for imaging applications alongside other applications. The user model for this larger market is a set of operations for viewing and manipulating images already entered into a database. The most common operations in this model are decompression, scaling, clipping, orthogonal rotation, and region-of-interest zooming.

*Display Performance and Quality Optimization* The main thrust of the DECimage accelerator is to achieve interactive performance for the operations defined in the user model. A secondary goal is to bring added value to the system by increasing the quality of the displayed image compared to the quality of the scanned image. A side effect of maximizing performance in hardware is that the main system processor has work off-loaded from it, freeing it for other tasks.

The general design of the accelerator uses a pipelined approach. Since maximum performance is desired and a large amount of data must be processed by the accelerator board, multiple passes through the board are not feasible. Similarly, the targeted low cost does not allow a whole image buffer on the board. With one exception (rotation), all board processing should be done in one pipeline, with the system processor simply feeding the input end of the pipe and draining the output end. Because of the large amount of data to be read from the board and displayed on the screen, the processor should only have to move that data, not do any further operations on it. To this end, any logic required to format the pixels for the display bitmap should be included in the pipeline.

*Cost Reduction through Less Expensive System Components* The net cost of a bitonal imaging system is influenced by the capability of the assist hardware. The capability of the hardware implies flexibility in the choice of other system hardware. In this regard, the most significant impact on cost occurs in the memory and the display. A system that makes use of fast decompression and scaling hardware can quickly display compressed images from memory. This means either more images can be maintained in the same memory, or the system can operate with less memory than it would without the assist hardware; less memory means lower cost.

A more dramatic effect on system cost is in the display. Imaging systems generally need higher-density displays than nonimaging systems, but the cost of a 150-dpi display is approximately twice the cost of a 100-dpi display of the same dimensions. However, we found that we could increase legibility, i.e., expand a bitonal image to a gray-scale representation, by using an intelligent scale operation in the hardware pipeline. For example, a bitonal image rendered to a 100-dpi display using the intelligent scale process gives the perceived legibility of the same image rendered to a 150-dpi display with a simple scaling method. That is, by adding the intelligent scale, a 100-dpi display can be used where previously only a 150-dpi display would be adequate.

*Cost Reduction through Integration* Presently, as in the DECimage 1200, hardware-assisted image manipulation exists as a board-level option. Higher levels of integration with the base platform will provide lower overall cost for an imaging system. The most straightforward method of integration is to relocate the hardware from the present option to the main system processor board; successive steps of integration would consolidate mapped hardware to fewer total devices. The most cost-effective integration will be the inclusion of the mapped hardware in the processor in a way similar to a floating-point unit (FPU). Just as graphics acceleration is now being included in system processor design, images will eventually achieve the status of a required data type and thus be supported in the base system processor.

## *Product Definition—What Does the User Want?*

The previously described strategy was used in the design of the image accelerator board for the DECimage 1200 system. The product requirements called for a low-cost, high-performance document image view station. These requirements evolved from the belief that most users currently investigating imaging systems are interested in applications and hardware that will enable them to quickly and simultaneously view document images and run their existing nonimaging applications. These users are involved with commercial and business applications, rather than scientific applications. The DECimage 1200 system was planned for the management of insurance claims processing, hospital patient medical records, bank records, and manufacturing documents. As previously stated, the imaging functions required for these view-oriented applications are high-speed decompression, scaling, rotation, zooming, and clipping.

## *General Product Design*

In defining the image capable system, the key points in the product requirements list were

- High-performance image display
- Low cost
- Bitonal images only (not gray-scale or color)
- View-only functions

The need for high-performance display influenced the project team to design the hardware accelerator board to handle image decompression, scaling, and rotation. Previous performance testing on a 3-VUP (VAX-11/780 units of performance) CPU had yielded image software display times from 5 to 19 seconds. These images were compressed

according to the CCITT Group 4 standard (300 dpi, 8.5-by-11 inches), and ranged from 20 to 100 kilobytes in size. In addition, the software display times were highly dependent on the image data content. The more complex image files, which had lower compression ratios, took significantly longer to decompress, scale, and display than the simpler image files. For example, an A-size, 300-dpi, CCITT Group 4 compressed image with a compression ratio of 10:1 took approximately 18 seconds to display, while another with a ratio of 33:1 took approximately 7 seconds.

The other three requirements led to decisions about the specific design of the image accelerator board. The need for low cost meant designing an option for an existing low-cost platform, which led us to Digital's VT1200 X window terminal. This requirement also led to our support of the proposed X Image Extension (XIE) protocol.[4] The XIE protocol extends the X11 core protocol to enable the transfer of compressed images across the wire and to enable interactive image rendition and display at the server. In the X windowing client-server environment, image applications and compressed image files exist on the client host machine, as depicted in Figure 2. In addition, the XIE protocol standardizes the interface-to-image functions in the X windowing environment and enables the development of a common application that can be used on any XIE-capable station. The client application issues commands to the X server display subsystem and the XIE specialized image subsystem. When a user selects an image to view, the compressed image file is transported from the client-side storage device to the X server memory.

Because the proposed accelerator would handle only bitonal images, we could specialize our board to decompress only the standard CCITT Group 3 and Group 4 bitonal compression algorithms. This specialization allowed the use of a Digital application-specific integrated circuit (ASIC) decompression chip. Finally, the view-only requirement limited the scope and complexity of the design by eliminating the need for extra hardware to handle the compression of images after they have been scanned and edited.

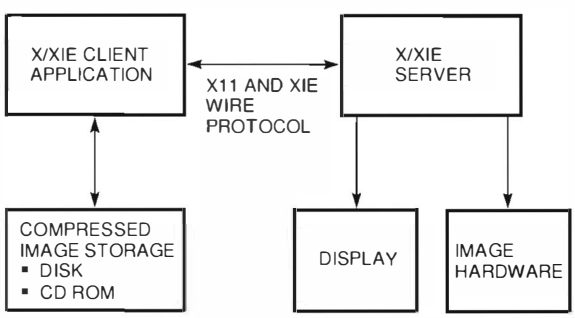



*Figure 2 X Client-server Architecture*

## *Specific Product Design*

The decisions described in the previous section led to our design of an image accelerator board that supports: CCITT Group 3 and Group 4 image decompression using an ASIC decompression chip; integer scaling using an ASIC scaling chip; orthogonal rotation; and image display. Figure 3 shows a general block diagram of the board and how it fits into Digital's VT1200 system architecture. The accelerator board is attached to the system address/data bus, and its registers, data input port, and data output port are mapped into the CPU's I/O space. The accelerator board is accessed by reading and writing specific addresses like any other system memory space. Note that the image accelerator logic is separate from the video terminal logic. Decompressed images are read from the image board and written to the base system video memory for display.

The main operation consists of the following steps: compressed image data is read from system memory and written to the ASIC decompression buffer by the processor; the data is then decompressed, scaled by the ASIC scaling chip, packed into words, and written to the output buffer. Figure 4 shows a detailed block diagram of the image accelerator board logic. The scaling chip outputs pixels of data (1 bit per pixel in this case) which are packed into words using shift registers. As soon as a word of data is available, the scaling chip output halts. Control signals generated in programmable array logic (PAL) write the packed word into the output buffer and tell the scaling chip to begin outputting pixels again. When the output buffer is full, the processor reads the rendered image data from the buffer. If rotation is required, the processor writes the data to the rotation matrix; otherwise, the data is clipped and written to the bit map. The image driver software, after setting up the board, alternates between checking whether the input buffer is empty and whether the output buffer is full.

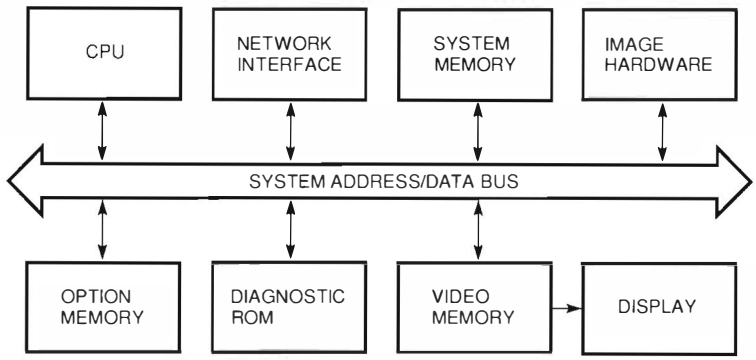



*Figure 3 VT1200 System Architecture*

The rotation circuit handles 90- and 270-degree rotation, whereas 180-degree rotation is handled in the data packing shift registers by changing the shift direction. The circuit rotates an 8-by-8-bit block of data at a time. The first byte of eight consecutive scan lines is written into eight individual byte-wide registers. The most significant bit (MSB) of each of these registers is connected to the byte-wide rotation output port latch. A processor read of this port triggers a simultaneous shift in all of the rotation data registers so that the next bit of each register is now latched at the rotation output port for the next read. Figure 5 diagrams the rotation circuitry just described.

To achieve the best performance, we pipelined the functional blocks in the hardware. The scaling engine does not need to wait for the entire image to be decompressed before it can begin scaling; instead, scaling begins as soon as the first byte of data is output from the decompressor. Thus different pieces of the image file are being decompressed, scaled, and rotated simultaneously. The hardware pipeline also eliminates the need to store the fully uncompressed image (approximately 1 megabyte of data for A-size 300-dpi images) in memory. The compressed image is written from system memory to the accelerator board and a decompressed, scaled, and clipped image is read from the board. Because of the speed of the hardware, the software can redisplay an image with different scaling, clipping, or rotation parameters; it merely changes the hardware setup for the different parameters and sends the compressed image file back through the accelerator board pipeline.

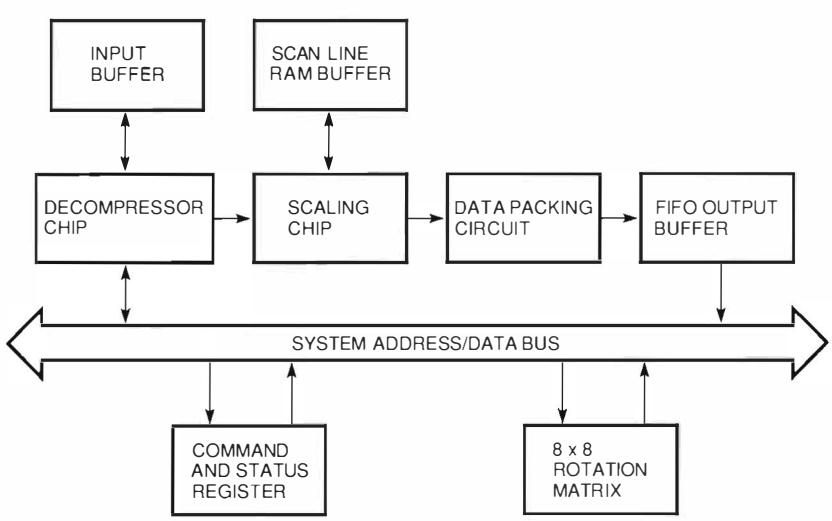



*Figure 4 Block Diagram of DECimage 1200 Accelerator Hardware*

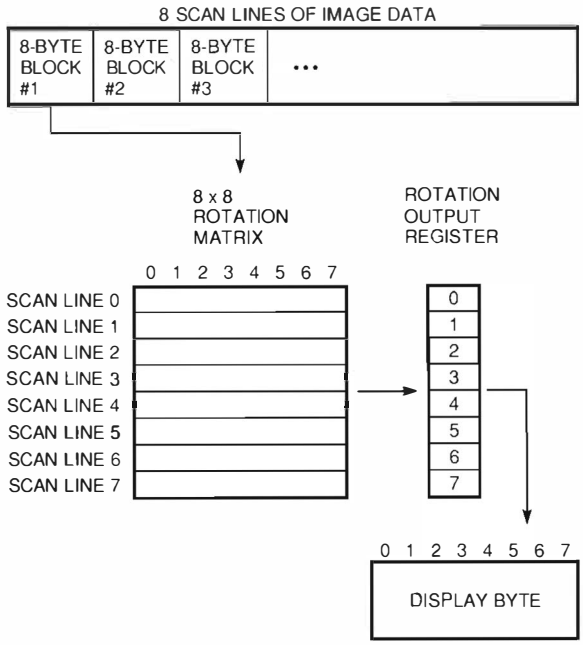



*Figure 5 Rotation Matrix*

## *ASIC Design Description*

The ASIC design consists of a decompressor chip, which decodes the compressed image data to pixel image data, and a scaling chip, which converts the image from the input size to the desired display size.

*Decompressor Chip* The decompressor chip acts as a CCITT binary image decoder. The chip contains three distinct stages, which are pipelined for the most efficient data processing. Double buffering of compressed input data is implemented to enable simultaneous input data loading and image decoding to occur. Compressed data is loaded into the input buffer by the processor through a 16- or 32-bit port. Handshaking controls the transfer of decompressed data from the decompressor's 8-bit-wide output bus to the scaling chip.

The first stage of the decompressor chip converts CCITT-standard Huffman codes, which are of variable-length, to 8-bit, fixed-length codes (FLCs).[5] A sequential tree follower circuit is implemented to handle this conversion. Every Huffman code corresponds to a unique path through the tree, which ends at a leaf indicating the FLC. The 8-bit FLC is sent to a first-in, first-out (FIFO) buffer, which holds the data for the second stage.

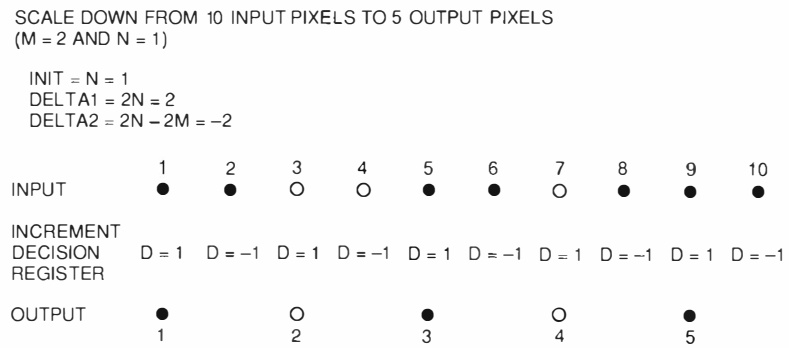
The second stage of the chip generates a 16-bit, run-length value from the FLC. The lower 15 bits of the word contain the number of consecutive white or black pixels (called the run length). The upper bit of the word contains the run-length color code (0 for a white run and 1 for a black run). An FLC is read from the FIFO buffer and decoded into one of eight routine types. Each routine is made up of several states that control the color code toggling, run-length adder, and accumulator circuits. At the end of each routine, a new word containing the run-length and color information is written into a FIFO buffer for the final stage.

The final stage of the decompressor chip converts the run-length and color information to black or white pixels. This stage outputs these pixels in 16-bit chunks when the scaling chip sends a signal indicating a readiness to accept more data.

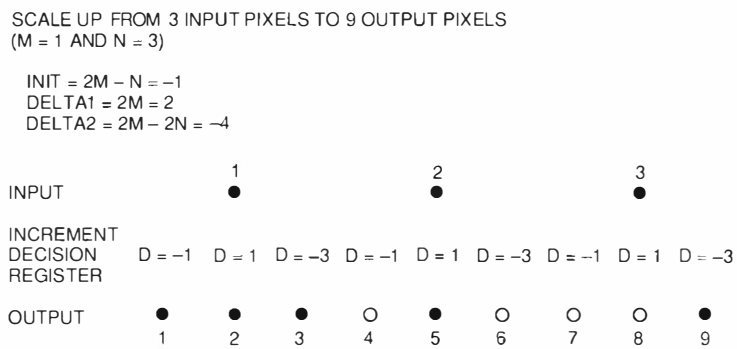
*Scaling Chip* The primary purpose of the scaling chip is to input high-resolution document images (300 dpi) and scale them for display on a medium-density monitor (100 dpi). The chip offers independent scaling in the horizontal and vertical directions. The scaling design implemented in the chip is a patented algorithm that maps the input image space to the output image space. General $M$-to-$N$ pixel scaling is provided where $M$ and $N$ are integers between 1 and 127, with the delta between them less than 65. $M$ represents the number of pixels in and $N$ represents the number of pixels out (in the approximated scale factor).

Given an image input size and a desired display size, we must find the $M$ and $N$ scale factors that best approximate the desired scale factor, within the range limits of $M$ and $N$ as previously stated. Thus an input width of 3300 and a desired output width of 550 are represented by an $M$ of 6 and an $N$ of 1. The approximated $M$ and $N$ values are loaded into the chip scale registers for downscaling or upscaling.

The chip scaling logic uses the scale register values to increment the input pointer position and generate output pixels. A latched increment decision term is updated every clock cycle, based on the previous term and the scale register values. When scaling down (where fewer pixels are output than are input), the logic increments the input pointer position every clock cycle, but only outputs a pixel when the increment decision term is greater than or equal to zero. Figure 6a illustrates how this algorithm maps input pixels to output pixels for a sample reduction. When scaling up (where every input pixel represents at least one output




*(a) Downscaling*




*(b) Upscaling*

*Figure 6 Chip Scaling Examples*

pixel), the logic outputs a pixel every clock cycle, but only increments the input pointer position when the increment decision term is greater than or equal to zero. Figure 6b illustrates how this algorithm maps input pixels to output pixels for a sample magnification. For both cases, the value of the pixel (black or white) being output is the value of the input pixel pointed at during that clock cycle. In this description, simply substitute rows for pixels to represent the vertical scaling process.

## *Software Support for the Hardware*

Software support is needed to enhance the functions of the hardware accelerator in our image view station. As mentioned in the section General Product Design, the XIE protocol extends the X11 core protocol to enable the transfer of compressed images across the wire and to enable image rendition and display at the server using the hardware accelerator board. Like the X11 protocol, the XIE protocol consists of a client-side library called XIElib, which provides client applications access to image routines, and a server-side piece, which executes the client requests. The XIE server implements support at two levels: device-independent and device-dependent. The device-dependent level supports the functions that benefit from optimization for a particular platform, or functions that are implemented in hardware accelerators. The device-independent level enables quick porting of functionality from platform to platform. Figure 7 illustrates the X/XIE client-server architecture.

The client-side XIElib offers the minimum functions necessary for image rendition and display. The toolkit level offers higher-level routines that assist with windows application development.

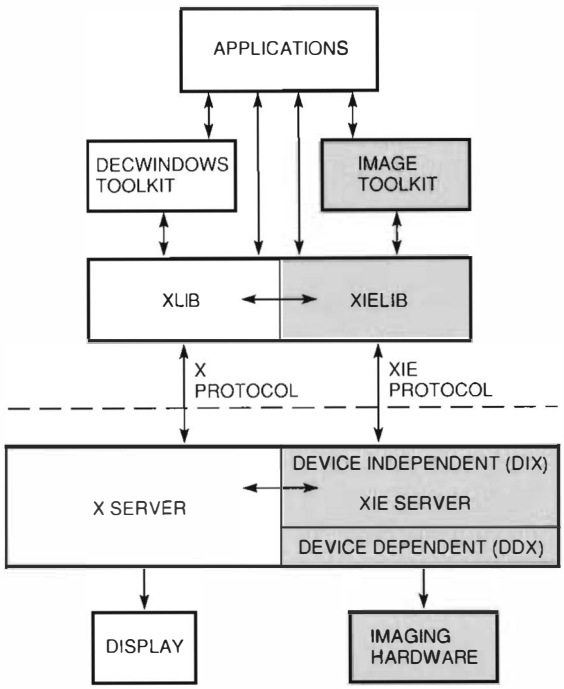



*Figure 7 X/XIE Architecture*

An example of a routine at this level might be ImageDisplay, which displays an image in a previously created window. ImageDisplay parameters might include $x$ and $y$ scaling values, the rotation angle, and region-of-interest coordinates. Whether programming with the XIE protocol at the library or toolkit level, applications developers benefit from the platform interoperability of the standard interface. Image accelerator hardware and optimized device-dependent XIE code changes the application's image display performance, but an application developed using the XIE protocol can run on any XIE-capable server.

### *Accelerator Performance Results*

With the DECimage 1200 X terminal, we have achieved interactive performance rates, reduced memory usage, and increased final image legibility. We achieved these rates by transporting compressed files instead of huge pixel files and by implementing specialized image processing hardware. The DECimage 1200 can read, transport, decompress, scale, and display an 8.5-by-11-inch bitonal document in 1 to 2 seconds. Successive displays, i.e., rotating, region-of-interest zooming, panning around the image, all occur in less than 1 second, which is essentially as fast as the user can ask for the displays. This speed is possible because the image already resides in compressed form in the server memory. Thus, the image does not have to be read from the disk or transported across the network.

## *Future Image Accelerator Requirements*

Hardware accelerators will continue to be required for bitonal imaging until software can provide the same functionality at the same performance level. This section discusses the more complex image schemes that are used for gray-scale imaging and multimedia applications. In contrast to bitonal imaging, these applications will require the use of hardware accelerators well into the future.

Other applications will require richer user interfaces utilizing continuous-tone images, video, and audio. All of these new data types are generally data-intensive, and compression or decompression of any one of them is a significant processing burden. Handling them in combination indicates that the need for specialized hardware assistance will persist for the foreseeable future.

### *Continuous-tone Images*

Bitonal images are either black or white at each point, but some applications require smoothly shaded or colored images. These images are typically referred to as continuous-tone images, a term that denotes either color or gray-scale, e.g., photographs, X rays, and still video. The representation and required processing of this image format is significantly different from that of bitonal images.

Continuous-tone images are represented by multiple bits per pixel. This format allows a greater range of values for each pixel, which yields greater accuracy in the representation of the original object. Additionally, each pixel can consist of multiple components, as in the case of color. The number of bits used to represent a continuous-tone image is chosen according to the nature of the image.

For example, medical X rays require a high degree of accuracy. Consequently, 12 bits are generally regarded as the minimum acceptable for the rendering of this class of image. Color images typically require 8 bits per pixel for each component (YUV or RGB format) for a total of 24 bits per pixel. Table 2 shows the relative size of samples of each image. The need to express these images in a

compressed format is obvious from the storage space requirements and the current storage media limits.

The compression of continuous-tone images can be accomplished in several ways. However, most imaging applications are not closed systems; inevitably, each system needs to manipulate images that are not of its own making. For this reason we adopted the JPEG standard, which specifies an algorithm for the compression of gray-scale and color images. Specifically, the JPEG compression method is based on the two-dimensional (2D) discrete cosine transform (DCT). The DCT decomposes an 8-by-8 rectangle of pixels into its 64 2D spatial-frequency components. The sum of these 64 2D sinusoids exactly reconstructs the 8-by-8 rectangle. However, the rectangle is approximated—and compression is achieved—by discarding most of the 64 components. Typically adjacent pixel values vary slowly, thus there is little energy in most of the discarded high-frequency components.

The edges of objects generally contribute to the high-frequency components of an image, whereas the low-frequency components are made up of intensities that vary more gradually. The more frequency components included in the approximation, the more accurate the approximation becomes. Table 4 shows some sample JPEG image compression ratios.[6]

The most popular part of the JPEG standard, the "baseline" method, was defined to be easily mapped into software, firmware, or hardware. Straightforward DCT algorithms can be efficiently implemented in firmware for programmable DSP chips, due to their pipelined architecture. The first systems to embody the standard did so using DSPs, because any change to either the evolving standard or a standard extension could be easily introduced to the firmware. The fastest implementations are achieved by special-purpose hardware accelerators.

**Table 4 Typical Compression Parameters for JPEG**

| Compression Ratio | Compression Method | Rendered Image Integrity |
|---|---|---|
| 2:1 | Lossless | Highest quality—no data loss |
| 12:1 | Lossy | Excellent quality—indistinguishable from the original |
| 32:1 | Lossy | Good quality—satisfactory for most applications |
| 100:1 | Lossy | Low quality—recognizable |

The JPEG implementation does not require hardware, i.e., the algorithm can be performed completely in software. The case for hardware assist is made in performance. Table 5 describes the reduced instruction set computer (RISC) processor performance, in millions of operations per second (mops), needed to provide the specified operation at a motion video rate of 30 frames per second.[7] However, generic RISC processors of those speeds are not available today. Therefore, dedicated, custom very large-scale integration (VLSI) devices (such as the CL550-10 from C-Cube Microsystems) must be used to perform the operations.[8] Even if the motion video rate is not required, the ASIC devices offer the simplest hardware solution.

## *Live Video and Video Compression*

Video captures the natural progression of events in an environment, and is therefore a natural and efficient way to communicate. Consider, for example, the assembly of a set of components. One way to express the assembly process is to show a series of photographs of the assembly at successive steps of completion. As an alternative, video can show the actual assembly process from start to finish. Subtle details of the process such as part rotations and movements can be clearly conveyed, with the added dimension of time.

Obviously, information expressed in video form can be valuable; however, significant problems arise in adapting video for use in computer systems. First, the huge data size of video applications can strain the system's storage capability. Video can be characterized as a stream of continuous-tone images. Each of these images consists of pixel values with individual components making up each pixel. For video to have full effectiveness, the still images must be presented at video rates. In many cases the rate to faithfully reproduce motion is 30 frames per second, which means that one minute of uncompressed video (512-by-480 pixels at 24 bits per pixel) would consume over 1 gigabyte of storage. In addition to storage demands, large volumes of data cause bandwidth problems. Presenting 30 frames per second to the video output with the above parameters would require a transfer rate of more than 22 megabytes per second from the storage device to the video output.

**Table 5 Processing Requirements for Imaging Functions**

| Imaging Functions | Processor Operations per Pixel* | | | | | Processor Operations at 30 fps (mops) |
|---|---|---|---|---|---|---|
| | Read | Write | ALU† | Multiply | Total | |
| Pixel move | .25 | .25 | 0 | 0 | .5 | 15 |
| Point operation | 2 | 1 | 1 | 0 | 4 | 120 |
| 3 × 3 convolve | 9 | 1 | 8 | 9 | 27 | 810 |
| 8 × 8 DCT | 24 | 1 | 14 | 16 | 65 | 1950 |
| 8 × 8 block matching | 128 | 1 | 191 | 0 | 320 | 9600 |

*RISC processor, 1M pixels, 30 frames per second (fps), 8 bits.
†ALU = arithmetic logic unit

Thus, reducing the amount of data used to represent the video stream would alleviate both storage and bandwidth concerns.

The starting point for the compression of video is with still images and, as previously mentioned, the JPEG algorithm can be used to compress still continuous-tone images. Because video can be represented as a sequence of still images, the algorithm could be applied to each still. This procedure would produce a sequence of compressed video frames, each frame independent of the other frames in the sequence.

The evolving Motion Picture Experts Group (MPEG) standard takes advantage of frame-to-frame similarities in a video sequence, thereby enabling more efficient compression than the application of the JPEG algorithm alone.[9] In most situations, video sequences contain high degrees of similarity between adjacent frames. The compression of video can be increased by encoding a frame using only the differences from the previous frame. The majority of scenes can be greatly compressed; however, scene transitions, lighting changes, or conditions of extreme motion need to be compressed as independent frames.

The need for hardware assist in this area is compelling. Table 5 shows that to sustain a JPEG decompression at 30 frames per second would require a 1950-mops processor. The same result can be obtained using the CL550-10 JPEG Image Compression Processor.[9] Although this device does not make use of interframe similarities to increase compression efficiency, a device implementing the MPEG standard would exploit these similarities. Table 5 shows that motion compensation, to be supported at 30 frames per second, requires a 9600-mops processor.

### *Audio and Audio Compression*

Video is usually accompanied by audio. The audio can be reproduced as it was recorded (with the video), or it can be mixed with the video from a separate source (such as a compact disc (CD) player). The audio data is defined by application requirements. If the application allows lower quality, the audio can be sampled at lower rates with fewer bits per sample, such as telephony rates, which are sampled at 8 kilohertz and 8 bits per sample. For applications requiring high-quality (CD) audio, samples are usually taken at 44 kilohertz and 16 bits per sample.

Integrating audio data into an application creates special problems. The major characteristic that differentiates audio from the other data formats presented here is its continuous nature. Audio must flow uninterrupted for it to convey any meaning. In video systems, the flow of frames may slow down under heavy system loading. The user may never notice it, or may not be annoyed by it. Audio, however, cannot slow or stop. For this reason, large buffers are used to allow for load variations that may affect audio reproduction.

A more subtle problem in creating applications using audio is in synchronization. Audio data is usually included to add another dimension of information to the application (such as speech). Without a method of synchronizing the video and audio, one data stream will drift out of phase with the other. One way to include synchronization is to use time stamps on the audio and video. This is particularly useful because standard time codes are used in most production machines.

The compression of audio data is not as efficient as that of the other data formats. Since a statistical approach to coding audio is highly dependent on

the type of input (i.e., voice, musical instrument), another method is required for generalized inputs. Differential pulse code modulation (DPCM) is often used to encode audio data. DPCM codes only the difference between adjacent sample values. Since the difference in value between samples is usually less than the magnitude of the sample, modest compression can be achieved (4:1). The limitation using this technique is in the coding of high-frequency data.

Hardware assist for the audio data format will probably come in the form of hardware to perform functions other than compression. For instance, DSP algorithms can perform equalization, noise reduction, and special effects.

### *Multimedia*

As the term implies, multimedia may integrate all of the previously mentioned image formats. The word "may" is important in this context. This area has been mainly technology-driven, due to such factors as lack of standards, developing I/O devices, insufficient system bandwidth, differing data formats, and a vast amount of software integration.

It is currently a topic of debate whether typical users will require the ability to create, as opposed to only access, multimedia source material. However, for discussion purposes, multimedia platforms can be classified into two categories: authoring and user. Authoring refers to creation of multimedia source material and requires different capabilities than user platforms. In the creation of a multimedia application, data from many different devices may need to be digitized and cross-referenced. As the data is incorporated, it is compressed and stored. Authors require the capability to edit and mix video and audio passages to get the desired result. Moreover, the video and audio may originate from different devices and may even be in different formats.

As defined above, "user systems" do not require all of the functions that authoring systems need: only decompression is required in a typical user system. Most existing user systems require an analog video source (videodisk), which is purchased as part of the application. The device control is performed by the application, i.e., when a user selects a passage to be replayed, the application sends commands to the videodisk. Figure 8 depicts an authoring system and a user system, along with suggested I/O capability.

Next-generation multimedia platforms will make full use of digital video and audio. This implies that systems will be able to receive and transmit multimedia applications and data over networks. This interactive capability will improve the efficiency of many mundane applications and devices. For example, electronic mail can be extended with video and audio annotations, or meetings can be transformed into video teleconferencing. The adoption of completely digital data for multimedia also implies that the platform I/O will change. Some user systems will not require analog device interfaces or control: the user will load the application over the network or from an optical disk.

Each of the image formats described in this section has different characteristics, and each will

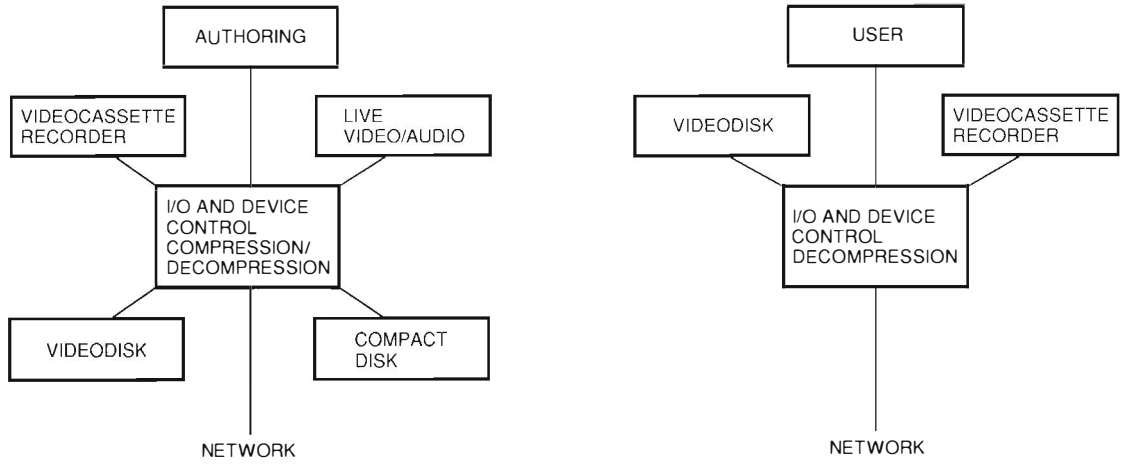



*Figure 8 Sample Multimedia Platforms*

be presented in the embodiment of multimedia. Given the size, processing requirements (compression and decompression), and real-time demands of applications, hardware assist will be a necessity.

## Summary

Imaging is a unique data type with special system requirements. To achieve interactive rates of bitonal image display performance today, hardware accelerators are needed; that has been the primary focus of this paper. In the future, a general-purpose processor should be able to handle the imaging process at the necessary speed, and beyond that, the processor should be affordable in a low-cost bitonal imaging system. However, the bitonal document processing market will not wait; it is in a high state of growth and requires that products like accelerators be developed for at least a few years.

Continuous-tone documents and multimedia applications will place an even heavier processing load on an imaging system. These areas will require accelerators for several years. As imaging applications, including bitonal, expand to cover more markets, the quality enhancements and performance benchmarks met by accelerators today will set customer expectations. Consequently, our future imaging products must be designed to meet these expectations.

## Acknowledgments

The authors wish to express thanks to the X Window Terminal Hardware and Software Design Groups for their support in developing the DECimage 1200 option. The two major ASICs used in the design were developed for previous projects, and those two design teams are also offered our thanks. Special thanks to Frank Glazer and Tim Hellman for their insightful research on the image rendering process.

## References and Note

1. *Standardization of Group 3 Facsimile Apparatus for Document Transmission,* CCITT Recommendations, Volume VI—Fascicle VII.3, Recommendation T.4 (1980).

2. *Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus,* CCITT Recommendations, Volume VII—Fascicle VII.3, Recommendation T.6 (1984).

3. *Digital Compression and Coding of Continuous-Tone Still Images, Part 1, Requirements and Guidelines,* ISO/IEC JTC1 Draft International Standard 10918-1 (November 1991).

4. J. Mauro, *X Image Extension Concepts, Version 2.4* (Cambridge: MIT X Consortium, June 1988).

5. D. A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," *Proceedings IRE,* vol. 40 (1962): 1098–1101.

6. G. K. Wallace, "The JPEG Still Picture Compression Algorithm," *Communications of the ACM,* vol. 34, no. 4 (April 1991): 30–44.

7. Table 5 is adapted from Y. Kim, "Image Computing Requirements for the 1990's: From Multimedia to Medicine," *Proceedings of Electronic Imaging West* (April 1991).

8. *CL550 JPEG Image Compression Processor, Preliminary Data Book* (San Jose, CA: C-Cube Microsystems Inc., November 1990).

9. *Coding of Moving Pictures and Associated Audio,* Committee Draft of Standard ISO 11172: ISO/MPEG 90/176 (December 1990).

*Björn Engberg*
*Thomas Porcher*

# X Window Terminals

*X window terminals occupy a niche between X window workstations and graphics terminals. The purpose of terminals in general is to provide low-cost user access to host computers or smaller dedicated systems. X window terminals further the advance in graphics terminals and provide new and interesting ways to utilize host systems. Ethernet cable provides for graphics performance previously not seen in terminals. The X Window System developed by MIT allows multiple applications to be displayed and controlled from the user's workstation. Now, with X window terminals, the same powerful user interface is available on host and other non-workstation computers.*

In mid 1987, the Video, Image and Print Systems (VIPS) Group began the design of Digital's first X window terminal, the VT1000 terminal and its code upgrade, the VT1200 terminal. Our goal was to design and implement an X window terminal that would allow the use of windowing capabilities on large computer systems. In 1989, Digital developed the VT1300 X terminal and in 1991 the VXT 2000 X terminal. The designs of these X window terminals are all quite different. Our design approach changed as the underlying technology changed.

This paper first compares host-system computing with applications that run on workstations. It summarizes the significance of the X Window System developed by MIT and discusses the client-server model. The paper then presents the need for X window terminals and follows their development stages. It compares and contrasts Digital's different design strategies for the VT1000, VT1200, and VT1300 X terminals. The paper concludes with a summary of the recently announced VXT 2000 X terminal.

## Background

Before the development of the X Window System, there was very little overlap in functionality between workstations and other kinds of computers. Workstations had stunning and fast graphics, and many powerful applications were available on them. Those applications were not available to users of basic 80-by-24 character-cell text display terminals connected to a host system located in a clean room. Graphics terminals, of course, allowed the use of ReGIS or another protocol for math and business graphics, but their performance was far below the expectations of a workstation user. Few people have the patience to run, for example, a computer-aided design application on a VT240 terminal, assuming such a version of the application is available.

Although a workstation offers fast graphics capabilities, its applications sometimes need more CPU power or more disk space to do calculations in a timely fashion. Graphics applications written for workstations could not run on faster host computers, which did not provide a display. Nor was there a standard way to get data from the host to display on a workstation. Each application required a unique solution to this problem.

Since the introduction of the new client-server model of computing and modern networks, many tasks can be divided into subtasks that can run on the most suitable processor. The X Window System uses the client-server approach, as shown in Figure 1. The application is viewed as an X client, and a workstation or a terminal can run an X server that controls the display. The X server also controls input from the keyboard and mouse or other pointing devices.

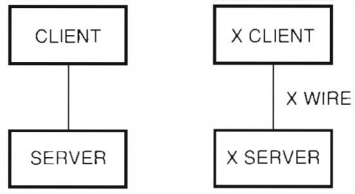



*Figure 1 Client-server Model*

An X client and an X server use an X wire to communicate, as shown in Figure 2. The X wire is simply a two-way error-free byte stream, which can be implemented in many different ways. The X Window System architecture does not stipulate how the X wire should be implemented, but several de facto standards have emerged. Manufacturers have designed X wires usually based on the data transport mechanisms that were available and convenient when the X Window System was implemented. The X wires use transmission control protocol/internet protocol (TCP/IP), DECnet, Local Area Transport (LAT), and other protocols, and even shared memory buffers as a transport to avoid protocol overhead. A single implementation often supports several transport mechanisms.

The X server typically executes on a processor with display hardware. The X client can execute on almost any processor. It may execute on the same CPU as the X server, or it may execute on a host, another workstation, or a compute server. The X server can be connected to several X clients simultaneously, with any combination of local (running on the same CPU) or remote (running on another CPU) X clients. The X server treats local and remote clients equally.

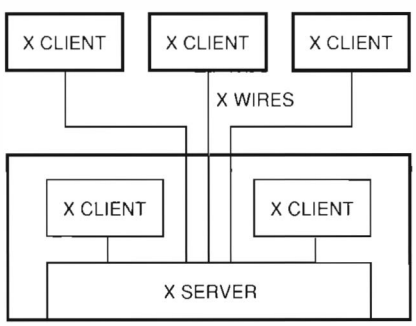



*Figure 2 X Wires*

## Workstation Environment

Figure 3 compares a traditional non-X windowing workstation with an X windowing workstation. In both workstations the application must use a graphics library to communicate with the display hardware and software.

In an X windowing client environment, the library of routines is called Xlib. An application designer can choose from a wide variety of toolkits, which are essentially a level of additional library routines between the application and Xlib. The use of a toolkit can significantly reduce the amount of work an application programmer has to do. The application software, Xlib, optional toolkit, and other libraries compose the X client, as shown in Figure 4.

With few exceptions, the X server comes with the display hardware and input devices (keyboard and pointer) indicated in Figure 5.

The X Window System with its flexibility neatly solves the problems of CPU power and disk space versus display availability. Applications written for X can execute on a wide variety of computers, and the results can be displayed on any of a multitude of devices, even on a workstation that would not

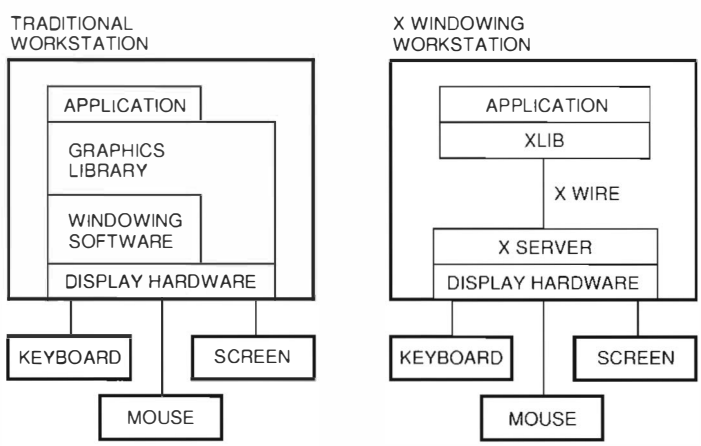



*Figure 3 Inside the Workstation*

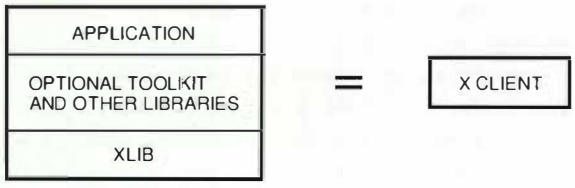



*Figure 4 The X Client*

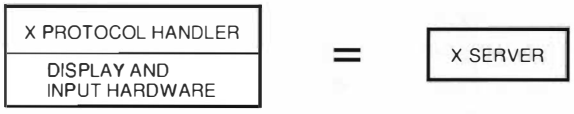



*Figure 5 The X Server*

have the capacity to run the application locally. Figure 6 shows how the X Window System fits into a network environment.

The X Window System has already generated many useful applications, and its widespread popularity ensures that many more applications will be made available in the future.

## Need for X Terminals

In a study to determine how workstations are used, the VIPS Group found that many users did not take advantage of the full potential of their workstations. In a software development or document editing environment, the users often set up their workstations as terminals. They usually created a few terminal emulation windows and used SET HOST or RLOGIN commands to connect to a host system on which they stored their working environment and files. Only two features of a workstation were frequently used. Users kept several terminal emulators on their screens at the same time, and set the terminal emulator windows to be larger than 80 by 24 characters. Only rarely did the average workstation user take advantage of the full power of graphics applications.

The results of our study indicated a need for a cost-effective alternative to a workstation that would provide the features desired by a large number of users. We envisioned a new kind of terminal, one that would allow people to have multiple windows of arbitrary size, to connect with multiple hosts, and, since the X architecture allowed it, to be able to use the same kind of graphics as a workstation.

From an X architecture standpoint, X terminals and X workstations are quite similar. They can in fact use the same hardware. For example, Digital's VT1300 terminal runs on the same hardware as the VAXstation 3100 workstation. X terminal software can also be made to run well on hardware platforms that are not suitable for workstations.

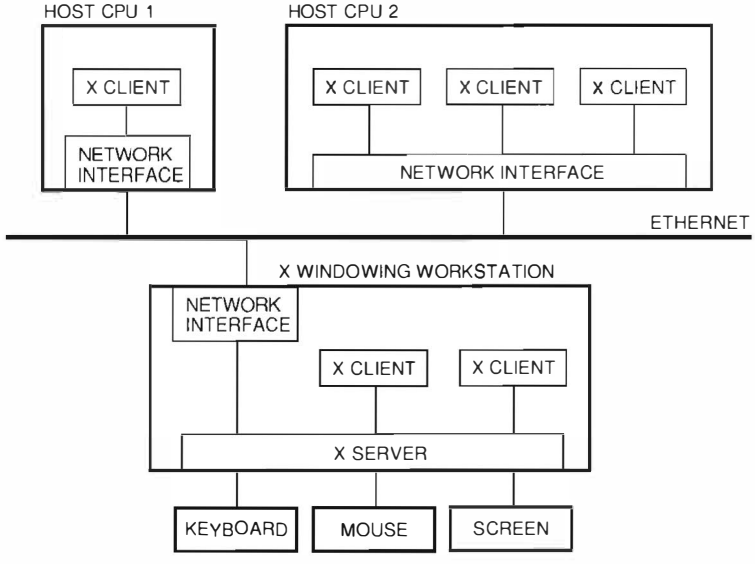



*Figure 6 X Window Network Environment*

The main architectural difference between the X terminal and X workstation software is that X terminals are closed systems that do not support local user applications. Although this may seem to be an unnecessary restriction, it does allow X terminals to be made for less money. An open system that allows any user application to run locally must have an established CPU architecture, a supported operating system, such as the VMS, UNIX, or ULTRIX system, and, subsequently, sufficient memory and/or disk space to support such an environment. A closed system, on the other hand, can be designed with simpler hardware, a smaller operating system, less memory, and thus lower cost. The absence of the ability to run user applications locally does not impact usability significantly since the user can run any desired application on another CPU. Digital's VT1000 and VT1200 X terminals were designed based on this approach.

## X Terminal Environment

X terminals often have local applications, but they must be built into the terminal by the designers. The VT1200 terminal has a video terminal emulator (VTE), a window manager, and a terminal manager as the local applications. The VTE allows the VT1200 terminal to make American National Standards Institute (ANSI) character-cell connections to a host, via the Ethernet or the serial lines as shown in Figure 7. This capability makes the VT1200 terminal useful in an environment that does not have X window support.

Although any X server can run windows software, it does not provide a user interface. To manipulate the windows, the user needs a window manager. The window manager creates window frames that allow the user to invoke functions to move windows, resize windows, change stacking order, and use icons. This capability also makes the VT1200 terminal useful when no host is available to run a remote window manager. A terminal with a local window manager generates less network traffic, and window management is not slowed by host congestion or network round-trip delays. The VT1200 X terminal allows use of a remote window manager, if the user prefers a different style of window management.

The local terminal manager provides the user interface to initiate connections to host systems. It is also responsible for the terminal customization interface.

All clients communicate with the X server using standard X wire commands only. Any window manager, remote or local, can manage all the windows on the screen, regardless of whether the clients are remote or local.

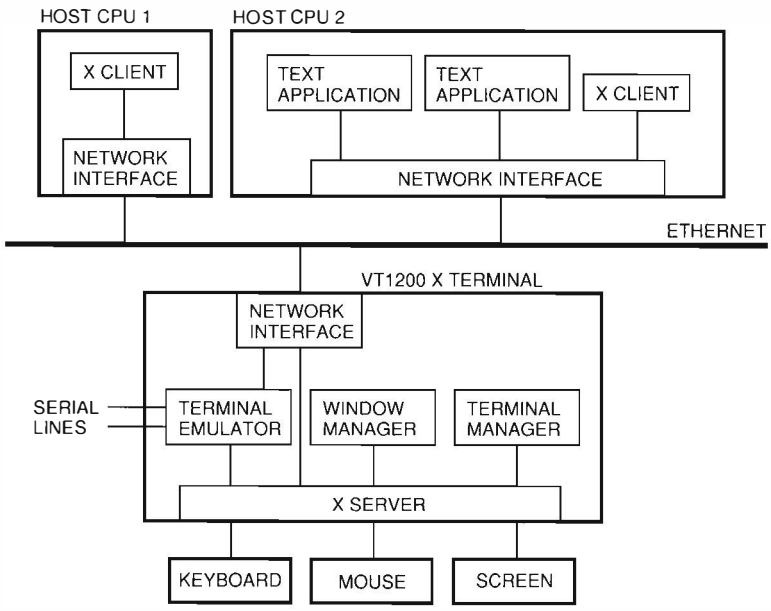



*Figure 7 The X Terminal Environment*

## Development of X Window Terminals

The development process of the VT1000 and VT1200 X terminals has important lessons to teach us. The knowledge we gained in 1987 has helped us develop future generations of X terminals.

When we designed the VT1000 X terminal and its code upgrade, the VT1200, we held many discussions within the group and with people from other groups. We planned many iterations before we arrived at the final architecture. It was by no means the only way to design an X terminal, and in 1989 we tried a different approach with the design of the VT1300 terminal. We knew that the best decision at a particular time might be very different from the best decision one year later, since the technical and marketing environment is constantly changing. New tools, standards, and practices enter the field while others become obsolete. Newer products must always have new features to meet changing technology requirements.

### Hardware Platform

Our first step was to discuss the hardware platform and select the kind of CPU to use, memory size, I/O considerations, type of display, etc. We studied many different CPUs to determine which one would provide the most capabilities for the lowest cost. A VAX chip was rejected because, at the time, it was far too expensive for the required price range of the VT1000 terminal. The Motorola 68000 series CPUs are quite powerful, but we had to consider other factors such as availability of software and hardware tools, cross compilers and linkers that could run on the VMS system, and hardware debugging facilities of sufficient power. We finally selected Texas Instruments' TMS34010 microprocessor with video support and several built-in graphics instructions that made it a cost-effective solution. It also came with VMS development tools, a C compiler, an assembler and linker, a single-step, hardware trace buffer with disassembler, and a powerful in-circuit emulator that made it possible to control execution in detail, inspect registers and memory, and set break points and hardware watch points (for example, break when writing value $x$ into location $y$).

We further discussed the kind of I/O to use. A sample implementation of the MIT X server on a VAXstation 2000 workstation and a primitive serial line protocol showed, as expected, that serial lines were clearly insufficient to carry the X wire protocol without some compression of the wire protocol itself. We had to build Digital's first X terminal with an Ethernet interface.

We needed to determine if this hardware platform could give us sufficient performance. We made several performance estimates, based on what we knew then about the X server and other software components. We went through each step in as much detail as we could (before anything was built). We calculated how many instructions were necessary to perform each task in the chain of receiving a command and displaying it on the screen. By knowing the speed of the CPU, we could estimate performance in characters or vectors per second. Our estimates showed that the VT1000 X terminal would not be exceedingly fast, but the performance would most probably be sufficient, definitely faster than a VAXstation 2000 in most cases.

In retrospect, actual performance of the VT1000 terminal and the later software upgrade, the VT1200, was close to our estimates, but it took several passes of code optimization to achieve such performance.

We also discussed alternate hardware designs for performance improvements. One solution proposed two CPUs, the TMS34010 microprocessor to handle the display and a 68000 microprocessor to handle I/O and other tasks. Unfortunately, we found no easy way to balance the workload between the two CPUs. We estimated that the different software components would have the following relative CPU demands:

- Interrupts, 5 percent
- Communications, 10 percent
- Operating system, 5 percent
- X server (minus display routines), 60 percent
- Display routines, 20 percent

To equalize the load between the CPUs, we would have had to split the X server in two, a solution that was not feasible. Any other split of tasks would cause one CPU to spend most of its time waiting for the other, and the overall performance gain would be minimal. Communication between multiple CPUs is complex and is very difficult to debug. Therefore, we decided that two CPUs were not worth the trouble or the cost. The best way to double performance is to install a single CPU that is twice as fast. At that time, the TMS34020 was

already being mentioned as a follow-up microprocessor. Since its software would be compatible with the TMS34010, we decided to keep it in mind for possible use in a future terminal.

### Code Selection

The use of read-only memory (ROM)-based code versus downloaded code has been debated for some time. ROM-based code starts up faster and incurs less network traffic at startup time (especially on a site with many X terminals), but is not flexible when software is upgraded. On the other hand, downloaded code can be easily distributed. An entire site can be upgraded with one or a few installations by a system manager as opposed to changing ROMs in a large number of terminals. (With the VT1200 X terminal, customers can change ROM boards.) From the point of view of terminal business, it made sense to use ROM-based code in 1987. We reasoned that not all sites would have Ethernet, but with ROMs the X terminal would still be useful as a multiwindow terminal emulator. We realized that such concerns would change with time, and on the whole, downloaded code would become the better approach. The only exceptions would be in the home or small office markets where a boot host or an Ethernet might not be available. Subsequent X terminals are being made in both downloaded (for example, in the VT1300 terminal) and ROM versions.

### Operating System Selection

Next we considered which operating system to use. We looked at other vendors' operating systems, but found they were either too complex and big or inadequate. One of our coworkers had written a very compact operating system for a VAX system used on another project. We used it in our prototype and then adapted it for the TMS34010 processor. We implemented additional functions to run the rest of the software with minimum changes.

There are many advantages to working with "your own" operating system. It is easy to make changes, to work around tricky problems, and to make special enhancements. But operating system code is difficult to debug. Timing is very critical, and throughout the project, we found strange bugs in code that had initially appeared to be all right to everyone involved. We found bugs under heavy load conditions after a rare sequence of events uncovered little timing windows and race conditions that had not been handled properly. Even with in-circuit emulators, such bugs could take weeks to track down.

In the VT1300 we decided to use the VAXELN operating system. We wanted to avoid the possibility of time wasted on finding and patching holes in the design of a new operating system.

### Local Terminal Manager

The VT1000 X terminal is self-starting at power-up, but without a host system, it needs a local user interface. We decided that this interface should resemble a workstation session manager and thus called it the local terminal manager. Although it covers a different set of functions, we wanted the local terminal manager to implement a similar set of objects and operations (the "look and feel" or style) of a workstation session manager. The style of the DECwindows session manager was chosen to make it easier for a user to switch between an X terminal and a DECwindows workstation. We wrote a subset toolkit for all the "customize" screens and ensured that the VTE could use the same subset toolkit for its "customize" screens. As DECwindows has progressed, subsequent X terminals have adapted the new user interface preferences, in this case Motif.

### Local Terminal Emulator

We considered a local terminal emulator to be an important component. We knew that X-based terminal emulators could run on the host, but in 1987 hosts with X windowing support were rare. Since we were in the terminal group, a terminal that could not manipulate ordinary text by itself was considered unsellable. We wanted the ability to access both X and non-X hosts and we wanted to support multiple text windows. Therefore we defined the terminal emulator as an X client so that text windows could coexist with X client windows. This feature has proved to be exceptionally popular. A large number of users use nothing but video terminal emulator windows. They are not interested in X windowing graphics, but do want multiple and/or larger text windows on a large screen.

### Local Window Manager

We debated whether or not to implement a local window manager. The DECwindows window manager was under development and was constantly changing. The DECwindows window manager

contained far too many VMS dependencies to be ported easily. Also the X terminal did not have enough memory to run the DECwindows toolkit locally. We could have ported other window managers, but they lacked the essential characteristics of the DECwindows window manager. For a while we considered letting the local clients have a primitive way to manage their own windows, until a full-featured window manager could be started on a host. Again, this alternative lacked the DECwindows system's qualities. We eventually decided to write a window manager based only on Xlib and our subset toolkit calls. It has the essential characteristics of the DECwindows product. Also, since the DECwindows window manager of necessity would keep changing, we wrote the local window manager in such a way that it could relinquish control to a remote window manager. This solution gave us the most flexibility for this hardware platform. The recently announced VXT 2000 X terminal has been designed with virtual memory to accommodate a well-established unmodified window manager, the Motif Window Manager.

### *X Server*

We also needed to choose an X server. We could have based our code on the distribution tape from MIT, but at the time the X Window System was not yet a mature product. Every implementor had to spend considerable time stabilizing the implementation enough to yield a product and improve performance. Since the VMS DECwindows Group had been writing code for the server, we decided to use DECwindows code. Once the porting effort started, we found that most of the performance had been improved by VAX MACRO code. Consequently, we had to re-engineer all the modules or adapt new ones from the MIT tape. As we kept porting and enhancing performance, our code changed more and more until it became extremely difficult to track bug fixes made by the DECwindows Group. The MIT patches were also nearly impossible to use because of code changes and because our starting code was one step removed from the tape.

Today the MIT X server is a mature product; patches and bug fixes are readily available from MIT and from the X community. In our current X terminals, the high degree of portability of the MIT X server allows us to keep most of the MIT X server source code almost unchanged so patches are easily applied.

### *Communications Protocol*

Many communications protocols were available, but our choice was dictated by market pressures rather than technical reasons. The market demanded TCP/IP. DECnet would have been acceptable, but it was running out of available addresses, at least within Digital. DECnet address space supports only 64,000 nodes and requires manual address and name assignments. After waiting weeks to get addresses for a few workstations, we realized that adding thousands of X terminals into Digital's internal network would not be possible. DECnet Phase V software has solved this problem.

Next we looked at the LAT protocol used by Digital terminal servers and found that it had several advantages. First, the VMS operating system supports the LAT protocol. LAT uses unique 48-bit Ethernet addresses to identify each node, which allows a large node address space. LAT also does not require any system management to add another terminal. A user can connect a terminal to a power source, and the terminal automatically becomes part of the network. Our performance evaluations found that the LAT interface on the host could be written to incur less host overhead than DECnet, which is important when many X terminals are connected to hosts.

Changes were needed in the VMS LAT driver to accommodate X wire and font service connections. The VMS Software Engineering Group worked with us to ensure that we would have those changes on schedule and in the appropriate VMS releases. As a result, we chose the LAT protocol for the VMS community and TCP/IP for users of ULTRIX and UNIX systems.

### *Font File System*

Storing fonts and changing font file formats were major problems. Since the VT1000 X terminal did not have a local file system, some fonts had to be stored in ROM to allow the VT1000 terminal to function in standalone mode. A quick review of the available DECwindows fonts showed that not all of them fit in the ROM space allowed for the terminal. Furthermore, customer-designed fonts or new font releases could not be accommodated. The solution was to be able to read fonts from a host system. This approach provided a font service on the VMS system, and enabled font files to be read over the Internet. We designed a process called the font daemon to run on the VMS operating system. This pro-

cess could deliver font data on request to one or several VT1000 terminals. The VMS system's font daemon uses the LAT protocol to deliver the fonts and protects somewhat against font file format changes. In many ways, the design of the font daemon makes it a precursor to a general font server, and it is very similar to the X Font Server being delivered by MIT in the latest release of the X Window System.

To use the font service, the terminal user must specify a font path in the VT1200 local terminal manager. Specifying a host name is sufficient to access the default font path, although users with their own font files can optionally search other directories. At startup, the VT1200 terminal makes a font connection to the host's font service and delivers the font path specification to the font service. The font service sends font names and other basic font information about all the fonts in the selected path. When the VT1200 X server needs a font, the VT1200 first searches the ROM-based fonts; if it is not there, a request to read the font is sent to the font daemon. The daemon sends the required information to the VT1200, and the X server can display characters from that font. Since memory is limited, the VT1200 has font caching, a mechanism to discard fonts no longer used or to discard the least used fonts. Our current X terminals increase the robustness of the font mechanism; for example, they provide recovery should the font service or its host become unavailable.

The special LAT code that we used on VMS systems for the font service was not available on UNIX and ULTRIX operating systems. Since internet protocol (IP) was available, we could use the trivial file transfer protocol (TFTP) to read a file from a host system, if the system manager set the proper protections. We chose TFTP for its ease of implementation and its wide availability on UNIX and ULTRIX systems. The TFTP font path in a VT1200 terminal specifies a host IP address and a complete path to a file (usually named font.paths) that contains the complete path to all the font files that the VT1200 can use. The terminal can then access all those font files, again through TFTP, to obtain font names and other basic information about each font. When a client wishes to use a font, the proper font file can be read again, this time to load the complete font. Since this process is time-consuming, the font path pointing to the file has an alternate format in which the font name follows the complete path to each file. Using this alternate format, the VT1200 terminal does not have to open and read the font file until a client actually intends to use it.

## Comparison of X Terminals

The VT1200 and VT1300 X window terminals were built using different approaches to solve the problems encountered during development. The X terminal is a new and flexible concept; there is no single "best" design. Table 1 compares the most important differences between the two terminals. We also include the specifics for the VXT 2000 X terminal.

The VT1200 is ROM-based; all its software is permanently resident in the terminal. The VT1300 software is downloaded, so a host or bootserver on the same network must supply the terminal with a load image at power-up.

Since downloaded terminals are dependent on the existence of at least one working host system, the user interface can be designed differently. While the VT1200 X terminal has a built-in user interface, the VT1300 does not need it. The VT1300 terminal automatically makes an X connection to a host at power-up, and the user is presented with the same DECwindows login box as on a workstation. The VT1300 has no local clients; all clients run on the host system.

The VT1200 terminal uses the LAT protocol for its ease of use and minimal network management demands. The VT1300 terminal uses the DECnet software already implemented in the VAXELN operating system used internally. Both terminals support TCP/IP.

## VXT 2000 X Terminal

One problem that has plagued all X terminals is limited memory space. Workstations usually have a virtual memory system, which provides large paging and swap areas on a disk, and applications and X servers can use more memory space than the hardware has. Until now X terminals have not had virtual memory systems. If too many applications made excessive demands, or if a client created large off-screen images (called "pixmaps" in the X Window System) the terminals quickly used all memory space. If the X server implementation was correct, an error was reported and a client might try a less demanding approach. In other cases, the terminal or client might simply crash. One alternative was to install more memory in the

**Table 1 Comparison of X Window Terminals**

| VT1200 Terminal | VT1300 Terminal | VXT 2000 Terminal |
|---|---|---|
| Monochrome only | Color only | Monochrome and color |
| 1 bit plane | 4 or 8 bit planes | 1 or 8 bit planes |
| Code in ROM | Code downloaded | Code downloaded |
| No virtual memory | No virtual memory | Virtual memory |
| 2-4MB RAM | 8-32MB RAM | 4-16MB RAM |
| TMS34010 CPU | VAX CPU | VAX CPU |
| Special operating system | VAXELN operating system | Special operating system |
| Local clients:<br>Terminal manager<br>Window manager<br>Video terminal emulator | No local clients | Local clients:<br>Terminal manager<br>Motif window manager<br>DECterm terminal emulator |
| Local customization | Customized on host just as a workstation | Local customization<br>Centralized customization |
| Choice of host (LAT only) | Automatic X window login to boot host | Choice of host (LAT and TCP/IP using XDMCP) |
| LAT protocol | DECnet protocol | LAT protocol |
| TCP/IP protocol | TCP/IP protocol | TCP/IP protocol |
| Special hardware | Available on several workstation platforms | Uses standard hardware |

X terminal, although this can be costly and offers no guarantees.

In the next generation of Digital's X terminals, the VXT 2000, this problem has found a cost-effective solution. Based on the VAX architecture, the VXT 2000 terminal uses virtual memory and downloaded code. The Digital InfoServer, an Ethernet storage server, provides the load image, virtual memory paging space, fonts, and customization storage. The same InfoServer also solves another problem: now the X terminal has access to a file system. This allows more extensive customization, as well as centralized management of the customization of all X terminals on the network. Figure 8 shows the configuration for the VXT 2000 X terminal.

## Conclusion

X terminals are not intended to replace workstations. Nor will workstations replace host systems or completely displace X terminals in the foreseeable future. It is likely that host computers will always be faster and have more memory and disk space than reasonably priced workstations of the same era. It is also likely that terminals can be built cheaper than workstations of reasonable

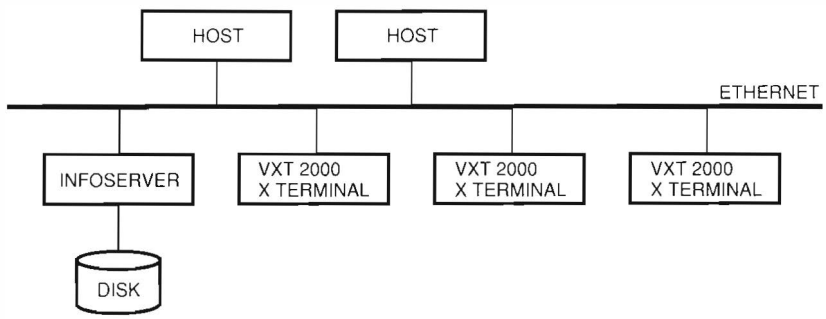



*Figure 8 The VXT 2000 Network Environment*

performance for some time to come. As long as that is the case, there will be a market for X terminals and host systems. Future X terminals will be faster, and have more built-in functionality, more local applications, X extensions, and most likely, additional hardware features. X terminals will be the networked terminals of the 1990s.

## Acknowledgments

We wish to thank the members of the VT1200 development team who worked many long hours on this project. Thanks to everyone inside and outside the Video, Image and Print Systems Group who contributed helpful suggestions, constructive criticism, and important hours using and testing the products. Thanks to the LAT and VMS Software Engineering Groups for incorporating the changes needed for the VT1200 X terminal to be useful. Thanks to the VIPS Quality Group for ensuring that as few bugs as possible remained in the product when shipped.

# ACCESS.bus, an Open Desktop Bus

*With the recent introduction of the ACCESS.bus product, Digital has affirmed its commitment to open systems and thus to facilitating better solutions for interactive computing. This open desktop bus provides a simple, uniform way to link a desktop computer to as many as 14 low-speed I/O devices such as a keyboard, mouse, tablet, or three-dimensional tracker. ACCESS.bus features a 100-kilobit-per-second maximum data rate, hardware arbitration, dynamic reconfiguration, a mature capabilities grammar to support generic device drivers, and off-the-shelf, low-cost $I^2C$ microcontroller technology.*

As the cost of personal interactive computing decreases, the range of applications and the need for specialized I/O devices is growing dramatically. Traditional personal computers were designed to accept only a small number of standard devices; adding devices beyond those originally envisioned usually requires specialized hardware or software. Custom interfacing is expensive for vendors and users and thus limits the availability of new devices.

ACCESS.bus provides a simple, uniform way to link a desktop computer to a number of low-speed I/O devices such as a keyboard, a mouse, a tablet, or a three-dimensional (3-D) tracker. Designed from the beginning as an open desktop bus, ACCESS.bus facilitates cooperative solutions using equipment from different vendors. This paper describes the ACCESS.bus design and gives some insight into how the idea was adopted at Digital.

## Design Goal, Process, and Advantages

The design goal for the desktop bus follows from our experience within the Video, Image and Print Systems (VIPS) Input Device Group with trying to support new devices on Digital terminals and workstations. While various new devices have been successfully prototyped over the years, the need for nonstandard hardware and custom software drivers was always an expensive, time-consuming obstacle. Even after successful prototyping, these devices could not be readily adapted to our standard systems, limiting their use to custom applications. In designing the desktop bus, our goal was to make it as easy as possible to interface previously unavailable I/O devices to our systems in a way that was both practical and marketable. This section explains the benefits of using a desktop bus, describes the process we went through to convert to a new bus architecture, and summarizes the key advantages of the chosen design.

The basic desktop bus concept is illustrated in Figure 1. The bus allows multiple, low-speed I/O devices to be interconnected and thus interfaced through a single host port. Desktop bus devices such as a keyboard or a tablet, which are not hand-held, provide two connectors and allow another device to be daisychained. A hand-held device such as a mouse can be placed at the end of the daisychain, or a connector expansion box can be attached to accommodate additional devices that do not provide two connectors.

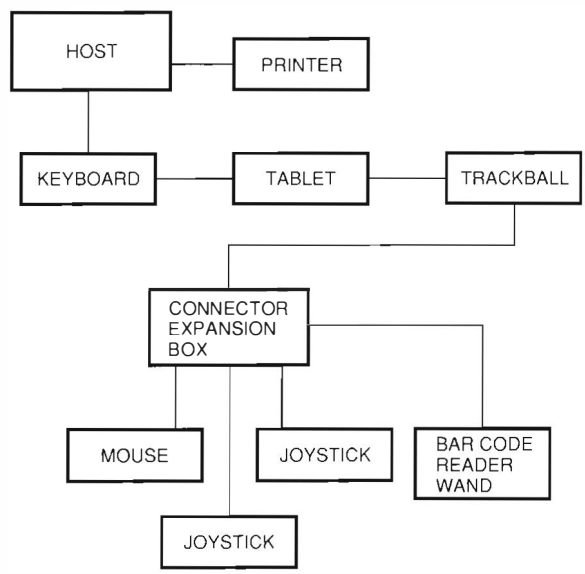



*Figure 1 Basic Desktop Bus*

The desktop bus has the following benefits:

- Enables greater flexibility and variety of use
- Reduces the cost of connecting multiple devices
- Expedites bringing new technology to market
- Helps leverage third-party devices

The first benefit, greater flexibility, can be simply achieved by allowing additional devices and more modular solutions. We further extended this benefit by designing a way for devices to be added at run time without disrupting system operation. Configuration should be automatic; connecting standard devices should not require powering down or rebooting the system before a new device can be used. The desktop bus supports multiple like devices without switches or jumpers.

The second benefit, reduced cost, was crucial to having the bus accepted as a solution across a wide range of products from low-end video terminals to high-end workstations. We recognized that contemporary electrical techniques could eliminate the need for level translation circuits, -12 volt (V) power supplies, and perhaps some of the protective components used with RS-232 interfacing. Although many devices would now require two connectors, system cost would decrease because we would need to supply only as many connectors as the number of devices to be attached, or possibly one more.

The third benefit, expediting the time to market for new technology, allows us to better satisfy changing requirements. Key to this benefit is having the means to connect new devices without changing the system hardware or software. Based on our experience with input devices, we developed the concept of device capability reporting and generic device protocols. Standard devices like keyboards and locators, e.g., mice, tablets, and trackballs, all work in similar ways. For this class of device, we define a simple device protocol and a way to parameterize and report device unique characteristics. A single generic driver can adapt itself to work with a class of similar devices so that no custom software is required for basic operation of standard devices.

Leveraging third-party devices, the fourth benefit, is aimed at satisfying diverse customer requirements. Because the use of computers continues to proliferate, the range of applications far exceeds that which any one vendor can master. By making the bus truly open, we encourage third parties to add value to our systems.

The benefits of a desktop bus are significant. But converting to a new architecture, especially one that is not backward compatible, is expensive in terms of the time and effort required. How does a large corporation build agreement to make such an investment decision? The desktop bus project started as a grass roots engineering effort and gradually built momentum. The process was one of dialogue to attract partners. Initially, three groups with slightly different objectives worked together to develop the bus. The visibility of separate groups jointly supporting the bus concept was essential to transform the idea into action. People are more willing to accept an idea that others around them have already adopted.

The three groups that initiated the desktop bus project were our VIPS Input Device Group in Westford, MA, mentioned previously; the Workstation Systems Engineering (WSE) Group, located in Palo Alto, CA; and the Video Advanced Development (A/D) Group in Albuquerque, NM. Our Input Device Group was looking for ways to simplify the process of prototyping specialized input devices and of getting related software support for our video terminals and workstations. WSE was developing a low-cost, personal workstation and needed a flexible way to support multiple input devices without greatly increasing the cost of the base workstation. The Albuquerque A/D Group had been experimenting with next generation I/O devices, i.e., force-feedback joystick, 3-D tracker, and real-time audio and video, and was interested in having these technologies adopted by other Digital groups. This A/D Group had used I²C technology successfully in one of its previous video projects.

In January of 1990, engineers from each group realized they were working on similar problems and began to collaborate. The WSE Group was to build the desktop bus host interface and software drivers into their workstation; the VIPS Group was to help define the device protocols and supply desktop bus keyboards and mice; and the Albuquerque A/D Group was to support bus development and prototype additional devices. Within four months, VIPS had defined the basic protocols and could demonstrate a working I²C keyboard and mouse. These early prototypes helped persuade WSE to support the project and, in turn, helped reinforce the importance of the project to the VIPS Group.

We began presenting the desktop bus idea to interested groups within Digital and received many useful suggestions including

- Use the same keycodes as on the LK201 keyboard to eliminate the need to rewrite keyboard lookup tables.
- Store the country keyboard variation inside the keyboard so users will not need to enter it manually.
- Keep the devices simple, without modes.

In addition, third-party input device vendors made the following suggestions.

- Use a modular connector that is easy to plug and unplug correctly.
- Provide enough power for several additional devices.
- Allow vendors to supply their own device drivers; tuning their own device drivers is part of the value added by the vendor.

The bus idea was elegant and generally well received. Most of the reservations centered around the likely impact on existing system components, the current problems, and whether conversion to the bus was feasible. Because we recognized that other groups were facing tight development schedules, we did not pressure these groups to support our desktop bus work. We presented the desktop bus as a possible solution to interface problems, made our design information available, and worked to incorporate suggestions. But as the development work progressed, more partners supported our effort.

Once we decided to use a desktop bus, we looked at available designs, including the Apple DeskTop Bus, the Musical Instrument Digital Interface (MIDI), and serial buses offered by other semiconductor vendors, and evaluated these alternatives with respect to our design goal. Key advantages of the design chosen, i.e., the ACCESS.bus, are

- Off-the-shelf interintegrated circuit ($I^2C$) microcontroller technology with maximum data rate of 100 kilobits per second (kb/s). This technology is low-cost, yet fast enough for sophisticated input devices like a 3-D tracker.
- Built-in hardware arbitration, which simplifies the software and allows reliable communication without inventing a new protocol.
- Dynamic reconfiguration. The hardware and software allow bus devices to be "hot-plugged" and used immediately, without restarting the system. The devices are recognized automatically and assigned unique addresses. This advantage results in a plug-and-play user interface.
- A mature capabilities grammar to support generic device drivers. An extensible free-form grammar allows devices to describe their characteristics to a generic driver. Most common devices can work with standard drivers.

Bus or network interconnection has become widely accepted as a means of providing flexible open solutions. To appreciate ACCESS.bus, it is helpful to position its performance capabilities with respect to those of other network interconnect technologies, as shown in Table 1.

**Table 1 Network Interconnects**

| Bus Type | Order of Magnitude Performance (kilobits per second) |
|---|---|
| Apple DeskTop Bus, ACCESS.bus | 10–100 |
| LocalTalk | 100–1,000 |
| Ethernet | 1,000–10,000 |
| FDDI | 10,000–100,000 |

At first glance, the 100-kb/s speed of the ACCESS.bus may seem adequate for large desktop devices like printers and modems. But these devices can transmit long data streams independent of any user activity and, if not restricted, could compromise the interactive performance of the bus. Thus, ACCESS.bus is intended for low-speed activities that people perform with their hands and is fast enough to handle multiple interactive devices like a keyboard, mouse, or 3-D tracker.

## *Hardware Description*

Before discussing the ACCESS.bus design, we present a description of the Philips $I^2C$ technology upon which the design is based. Details of the specific ACCESS.bus implementation follow.

### *Interintegrated Circuit Fundamentals*

ACCESS.bus extends the Philips $I^2C$ bus to operate off-board and, thus, connect desktop devices. The $I^2C$ is a two-wire serial clock and serial data

open-collector bus. An open-collector design means that the clock and data lines are normally in a high-impedance floating state and are pulled up to a logical high state.

A device that wants to send a message waits for any message frame in progress to complete, then asserts a START signal to become bus master and begins to generate data and clock signals. The bus clock is synchronized among all devices by its wired AND connection. Each device, whether transmitting or receiving, stretches the low period of the clock until ready for the next bit to be transferred. When the last device is ready, the bus clock is allowed to go high, generating a rising edge on the serial clock. At this time, all active devices sense the state of the bus data line. For a receiving device, the state represents the received data bit. For a transmitting device, the state determines whether the device has successfully asserted its data on the bus. A transmitter that is sending a logical high state and detects that the data line is being held low by another sender, recognizes that it has lost arbitration and must try again later. When a "collision" or arbitration occurs, no data is lost, one message is transmitted and received, and the remaining messages must be sent again.

I²C data messages are transmitted as 8-bit bytes, with each byte being acknowledged by a ninth ACKNOWLEDGE bit from the receiver. I²C technology also defines unique START and STOP signals to delimit message frames. The first byte of any message frame is always the destination address.

### ACCESS.bus Physical Implementation

Details of the physical implementation of ACCESS.bus are as follows:

- Basic electrical configuration. ACCESS.bus uses four-pin, shielded, modular-type connectors that feature positive orientation and locking tabs. Data and power for the bus are transmitted over low-capacitance, four-wire, shielded cable. The four conductors are used for ground, serial data, serial clock, and +12 V.
- Available power. The maximum available power for all devices is 12 V at 500 milliamperes (mA). ACCESS.bus devices may supply their own power from a separate source, if needed. A power-up reset circuit must still be provided to reset the device when bus power is applied.
- Cable length. The maximum cable length for the entire bus is 8 meters. The limiting factor is a maximum capacitance not to exceed 700 picofarads (pF).
- Number of devices. The maximum number of ACCESS.bus devices allowed on the bus is 14. Limiting factors are the device addressing range and the power distribution (a total of 500 mA for all devices).
- Hardware interfaces. ACCESS.bus hardware interfaces are implemented using standard I²C microcontrollers developed by the Signetics Company or under license from Philips Corporation. (Signetics Company is a division of North American Philips Corporation.)

## ACCESS.bus Protocol

Every device on the bus is a microcontroller with an I²C interface and behaves as either a master transmitter or a slave receiver, exclusively, as defined by the I²C Bus Specification.

### Message Format

A message transmits information between a device and the computer or between the computer and one or more devices. There is one exception: a device may attempt to reset other devices assigned to the same address by sending a Reset message to itself.

ACCESS.bus messages have the following format:

| Byte Number | Bit Number [ 1 2 3 4 5 6 7 8 ] | |
|---|---|---|
| 1 | [ destaddr \|0 ] | Destination address |
| 2 | [ srcaddr \|0 ] | Source address |
| 3 | [ P \| length ] | Protocol flag, length (the number of data bytes from 0 to 127) |
| 4 through (length + 3) | [ body ] | Consists of 0 to 127 data bytes |
| length + 4 | [ checksum ] | |

Initially, devices respond to a default power-up address. During the configuration process, the computer assigns a unique address to every device on the bus. Messages are either device data stream (P=0) or control/status (P=1), as indicated by the

protocol flag. The minimum length of a message is 4 bytes; the maximum length is 131 bytes (127 data bytes and 4 bytes for overhead). The message checksum is computed as the logical XOR of all previous bytes, including the message address.

### *Standard Messages*

The ACCESS.bus protocol defines the seven standard interface messages summarized in Table 2. Parameters defined within the body of the message are listed in parentheses.

### *Identification*

Since the ACCESS.bus is a bus-topology network, unique identification strings are used to distinguish devices. These strings are structured as follows:

| | | |
|---|---|---|
| protocol revision: | 1 byte | (e.g., "A") |
| module revision: | 7 bytes | (e.g., "X1.3 ") |
| vendor name: | 8 bytes | (e.g., "DEC ") |
| module name: | 8 bytes | (e.g., "LK501 ") |
| device number: | 32-bit signed integer | |

The module revision, vendor name, and module name strings are left-justified ASCII character strings padded with spaces. The device number string is a 32-bit two's complement signed integer and may be either a random number (if negative) or a unique serial number (if positive).

## *Configuration Process*

The configuration process is used to detect what devices are present on the bus, assign each device a unique address, and connect devices to the appropriate software driver. Configuration normally occurs at system start-up, or at any time when the computer detects the addition or removal of a device.

### *Power-up/Reset Phase*

When reset or powered up, a device always reverts to the default address and sends an Attention message to alert the computer to its presence. At system start-up or reinitialization, the computer sends a Reset message to all I²C addresses in the ACCESS.bus device address range (14 messages) to ensure that all devices on the bus respond at the power-up default address.

### *Identification Phase*

To begin address assignment, the computer sends an Identification message at the device default address. Every device at this address must then respond with an Identification Reply message. As each device sends its message, the I²C arbitration mechanism automatically separates the messages based on the identification strings. The computer can then assign an address to each device by including the matching identification string in the Assign Address message. When a device receives this message and finds a complete match with the identification string, it moves its device address to the new assigned value. As soon as a device has a unique address, it is allowed to send data to the computer.

The I²C physical bus protocol allows multiple devices on the bus at the same time if those devices

**Table 2 Standard ACCESS.bus Protocol Messages**

| Computer-to-device Messages | Purpose |
|---|---|
| Reset () | Force device to power-up state and default I²C address. |
| Identification Request () | Ask device for its "identification string." |
| Assign Address (identification string, new address) | Tell device with matching "identification string" to change its address to "new address." |
| Capabilities Request (offset) | Ask device to send the fragment of its capabilities information that starts at "offset." |
| **Device-to-computer Messages** | |
| Attention (status) | Inform computer that a device has finished its power-up/reset test and needs to be configured; "status" is the test result. |
| Identification Reply (identification string) | Reply to Identification Request with device's unique "identification string." |
| Capabilities Reply (offset, data fragment) | Reply to Capabilities Request with "data fragment," a fragment of the device's capabilities string; the computer uses "offset" to reassemble fragments. |

are transmitting exactly the same message. In the rare event that two like devices report the same random number or are mistakenly assigned to the same address, each interactive device transmits a Reset message to its assigned address prior to sending its first data message after being assigned a new address. The self-addressed Reset message forces other devices at the same address back to the power-up default address, as if they had just been hot-plugged. The message guarantees that each device has a unique address, but not until the device is actually used. The pseudo-random number (or serial number, if available) distinguishes devices at identification time before they are used, allowing the host to inventory which devices are present.

### *Capabilities Phase*

Device capabilities is the set of information that describes the functional characteristics of an ACCESS.bus peripheral. The purpose of capabilities information is to allow software to recognize and use the features of bus devices without prior knowledge of their particular implementation. By having locator devices report their resolution, for example, generic software can be written to support a range of device resolutions. Capabilities information provides a level of device independence and modularity.

The structure of capabilities information is designed to be simple and compact for efficiency, but also extensible to support new devices without requiring changes to existing software or peripherals. These objectives are supported by making the structure hierarchical and representing capabilities information in a form that applications (and humans) can use directly. The capabilities information is an ASCII string constructed from a simple, readable grammar. The grammar allows text strings to be formed into lists, nested lists, and lists with tagged elements. The capabilities string for a locator might read as follows:

```
(prot(locator)
 type(mouse)
 buttons( 1(L) 2(R) 3(M) )
 dim(2) rel res(200 inch) range(-127 127)
 d0(dname(X))
 d1(dname(Y))
)
```

After assigning a unique address to a device, the computer retrieves the device's capabilities string as a series of fragments using the Capabilities Request and Capabilities Reply messages. The computer then parses the capabilities string to choose the appropriate application driver for the device. The parsed string is also made available to application programs using the device.

### *Normal Operation*

During normal operation, the computer periodically requests inactive devices to identify themselves. If a device is found to be missing, or a new device appears by sending an Attention message at the default address, the computer sends an Identification Request message to each device address previously recorded as in use (up to 14 messages) to confirm which devices are still present. The computer also sends a Reset message to each device address previously recorded as not in use. The computer then begins the address assignment process by sending an Identification message to the default address and assigning each device that responds to an unused device address.

## *Generic Device Concepts*

ACCESS.bus uses the concept of generic device drivers to support familiar I/O devices using only a few drivers. Generic specifications for keyboards, locators, and text devices have been developed.

### *Keyboards*

The keyboard device protocol attempts to define the simplest set of functions from which a Digital LK201 or a common personal computer keyboard user interface can be built. A generic keyboard consists of an array of key stations assigned numbers between 8 and 255. When any key station transitions between open and closed, the entire list of key stations currently closed or depressed is transmitted to the host. This reporting scheme is functionally complete; the host can detect every key transition, and the scheme provides information about the full state of the keyboard on each report. No special resynchronization reports are required.

In addition to reporting key stations, the generic keyboard device can support simple feedback mechanisms such as keyclicks, bells, and light-emitting diodes. These mechanisms are controlled explicitly from the host so that minimal keyboard state modeling is required. The capabilities information is used to identify the keyboard mapping table and the feedback mechanisms available. The keyboard mapping table can also be stored in the keyboard itself as part of the capabilities string.

### *Locators*

The locator device protocol is designed to accommodate a range of basic locator devices such as a mouse or tablet. More complex devices can be modeled as a combination of basic devices or can provide their own device driver, thus minimizing the burden on the protocol.

A generic locator consists of one or more dimensions described by numeric values and, optionally, a small number of key switches. The standard driver requires the locator device to identify the type of data it will report from a small list of options and adjusts to handle this data type. These options are

- Number of dimensions, e.g., two, for a mouse or a tablet
- Dimension type: absolute, i.e., referenced to some fixed origin, like a tablet; or relative, i.e., changed since last report, like a mouse
- Resolution in divisions per unit, e.g., counts per inch or counts per revolution
- Dynamic range of values that can be reported, i.e., the minimum and maximum values
- Number of key switches, from 0 to 15

The assignment of scalar-value dimensions returned from one or more devices to the user interface functions is left to the application. However, to accommodate most conventions, the scalar dimensions and the key switches can be labeled in the capabilities string.

### *Text Devices*

The text device protocol is intended to provide a simple way to transmit character data to and from character devices such as a bar code reader or a small character display. A generic text device transmits a stream of 8-bit bytes from a character set. Simple control messages are defined to support flow control and to select communication parameters that might be used to interface with a modem. The capabilities string contains information that identifies the specific character set and communication parameters used.

## *Summary*

The ACCESS.bus network interconnect offers the possibility of a standardized, low-speed, plug-and-play serial communications channel that can untangle peripheral interfacing and open the way to new applications. As the advantages of this open desktop bus design become well known, we expect wider adoption of this product. The ACCESS.bus is currently implemented on Digital's Personal DECstation 5000 workstation, with implementations underway for the next generation of RISC workstations and video terminals.

## *Acknowledgments*

Many people contributed to the design and development of ACCESS.bus. I would especially like to acknowledge Tom Stockebrand and Tom Furlong for their vision and early support; Chris Cued, Mark Shepard, and Ernie Souliere for their contributions to the ACCESS.bus electrical design and protocols; and Robert Clemens for the excellent demonstration hardware and firmware development support.

## *General References*

D. Lieberman, "Desktop Bus Is Born Free," *Electronic Engineering Times* (September 2, 1991): 16.

*ACCESS.bus Developer's Kit* (Palo Alto, CA: Digital Equipment Corporation, Workstation Systems Engineering TRI/ADD Program, 1991).

*Signetics $I^2C$ Bus Specification* (Sunnyvale, CA: Signetics Company, a Division of North American Philips Corporation, February 1987).

*Richard Landau*
*Alan Guenther*

# *Design of the DECprint Common Printer Supervisor for VMS Systems*

*DECprint Printing Services software controls a variety of printer features for a wide range of printers. It supports several different page description languages, handles multiple media simultaneously, and uses different I/O interconnections and communication protocols. Operating within the VMS printing environment, it implements a large number of user-specified options to the PRINT command. DECprint Printing Services functions as the supervisor in the VMS printing system for all PostScript printers supplied by Digital. The common printer supervisor has an especially flexible internal structure and processing method to serve complex printing environments.*

The increasing variety and complexity of printing devices in the last decade have strained the abilities of operating systems to support them. Users demand access to, and control over, the increasingly sophisticated features of their printers. At the same time, application programming resources are stretched by the requirement to support various devices and features. Modern operating systems include printing systems that support printers and insulate applications from many details of printing.

DECprint Printing Services software was designed to handle a wide variety of printers, with a range of I/O connections, media handling capabilities, finishing equipment, data syntaxes, and so forth. It provides the controlling software that supports the full range of Digital printers capable of printing PostScript documents.

DECprint Printing Services functions as a component of the VMS printing system at the level of printer supervisor, called symbiont in VMS terminology. The supervisor is known within Digital as the DECprint common printer supervisor or common print symbiont (CPS). It is called common because it replaces a number of different symbionts and is common to a range of printers. CPS is a completely new program developed by the Video, Image and Print Systems Group.

This paper explores the environment in which printing systems now reside. It describes the structure and functions of DECprint Printing Services and the design of CPS, focusing on its capabilities within the VMS system. The paper then discusses the operation of the VMS printing system and the enhanced printing environment made possible by CPS.

## *Printing System Dimensions*

A printing system is the set of software and hardware components through which print requests pass from the time the user decides to print a document until the appropriate hard copy arrives.

The variety of printing devices in use is a challenge for the printing system and for application programmers. We use the word "printer" in this article to imply the full range of output devices that are attached to systems and networks. A system today must support a wide number of dimensions: marking technologies, media, medium sizes, speeds, transmission rates, and interconnects.

## *The DECprint Model of Printing*

The DECprint model of printing is composed of several layers. Each layer has defined functions and I/O interfaces. The layers of the DECprint model and their relationships to VMS and CPS are shown in Figure 1. This model of printing describes a useful structure with consistent functions and responsibilities.

- Application. An application program creates information that the user may want to print. All types of applications fit into the model at this

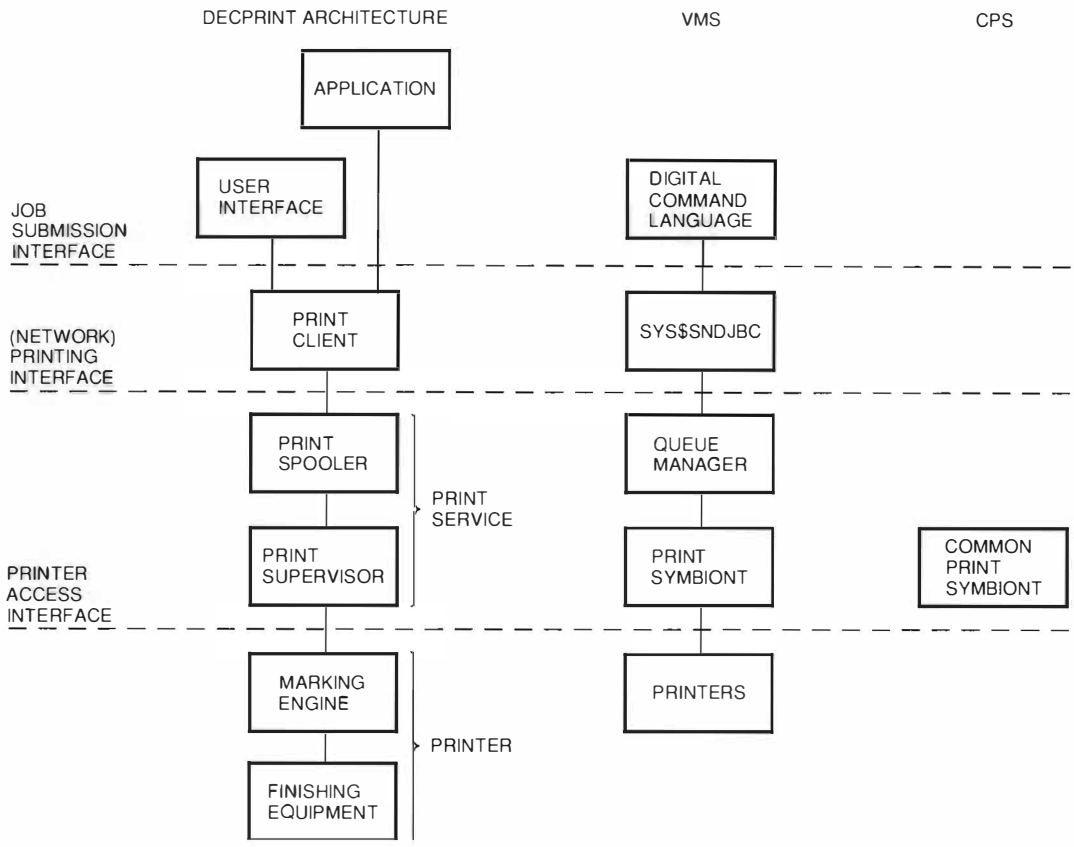



*Figure 1 Relationships of the VMS Printing System Components to the DECprint Model*

level, from data processing programs and simple text editors to high-quality document formatting and publishing applications. The application may present a printing interface directly to the user, or may create a final form document from which the user can access other printing interfaces.

- User printing interface. A user expresses the desire to print through a user interface to the printing system. The interface may be oriented to written commands, to user selection of menu choices, or to a point-and-select graphical interface.
- Job submission interface. User interface programs communicate with the lower levels of the printing system through an application programming interface (API) to the print client. The API contains full capabilities for creating, destroying, and managing print jobs of all types. The job submission interface may be operating system-specific or may be based on emerging standards for network printing.
- Print client. The client accepts requests through its API, performs defaulting for the user, assists in selecting the correct print service, gathers the print instructions and document files, and submits the job to the print service. The protocol used to submit the job may be operating system-specific or may be based on emerging standards for network printing. The print service may be local to the print client (and the user), or it may be located elsewhere in the network.
- Print service. The print service is a convenient abstraction that includes the print spooler and all subsequent layers in the execution of the print job, for some set of physical printers. Printers are often grouped together based on their static characteristics, such as type of printer, printer data syntax, and default media.
- Print spooler. The print spooler accepts the print job from the client, spools the files and queues the job for later execution if necessary, and then schedules the job for execution. If the job

requires resources that are not immediately available, human intervention may be necessary. For example, if a job requires a special print medium, then an operator or other printer attendant must provide the medium for the printer. If the job requires a special font, the spooler may be able to obtain the font from a library without human intervention.

- Printer supervisor. The supervisor directly controls the printer. It interprets the print instructions for the job, manages the printer and its finishing equipment, and writes the document data to the page description language (PDL) interpreter. It also monitors the status of the printer, supplies some resources on demand, and responds to error conditions. On the VMS operating system, the printer supervisor is called a symbiont; on ULTRIX and UNIX systems, a daemon.

- PDL interpreter. Generally, final form document data is written in a data syntax intended for printing, but it is not in the native form required by the marking engine. A PDL interpreter transforms the printer language into the lower-level form for the marking engine. For example, in a typical laser printer, a PostScript interpreter transforms the PostScript language into a device-level bit map and media control instructions for the print engine. In a simpler impact printer, the controller turns characters and control sequences into pin timing and paper movement instructions.

- Marking engine. The marking engine consists of the media transport and printing mechanisms, generally controlled at a low level. Marking may be done by a wide spectrum of technologies, and the media used may also vary widely. For the most part, descriptions in this paper use raster devices such as laser printers as examples.

- Finishing equipment. The overall printing system includes finishing options that are not often considered part of the (largely electronic) printing system. Currently affordable components of the printing system are typically automated. For example, several years ago duplex (two-sided) printing was not economical for most office applications; today it is, and many office printers include this finishing feature. Stapling, on the other hand, is still not economical for most office applications, though it is implemented in many high-end production printers.

Implementations of the model in various operating systems and printers may express the layers differently, sometimes skipping certain layers. The VMS printing system contains components at most levels of the DECprint model. The DECprint common printer supervisor (CPS) operates within the VMS system, as indicated in Figure 1. We designed CPS to satisfy the requirements and projected needs of users, system managers, and programmers. In the next section we discuss the design of CPS.

### *Sharing Devices*

Printers are often shared, especially high-end or specialized, expensive devices. Since shared printers are not always immediately available to the user or application program, the printing system is required to hold jobs for printing later. The system must be able to store the user's instructions for printing, along with the contents of the document, until they are needed.

### *Insulating the Application from Details*

A printing system insulates applications from the details of printing devices. For example, DECprint Printing Services provides communications mechanisms and protocols, determines whether a shared device is currently busy, and sometimes translates printer data syntax.

Application programmers generally prefer to deal with as few external interfaces as needed to perform the task. Thus it is desirable to minimize the number of different *classes* of printing devices while maximizing the variety and flexibility of printing devices. The DECprint architecture specifies that the printing system take responsibility for matching the needs of the application to the capabilities of the output device, whenever possible. For example, a printing system might need the ability to transform the printer data stream from a data syntax used by the application to a data syntax used by the printer. Hidden transformation makes the system easier for applications to use. DECprint Printing Services provides a certain number of printer data syntax transformations of this type, from languages such as DEC PPL3 (which is commonly referred to as "ANSI" within Digital) and ReGIS to PostScript, and from PostScript to printer bit maps.

## *Internal Structure of CPS*

In designing CPS, our primary goal was to create a flexible system that would handle all the printer

features we could foresee and many that we could not foresee, a system that could be modified as needed to handle not just new printers but new *classes* of printers. CPS is capable of managing a wide variety of character, line, page, and document printers.

To create a flexible printing system, we needed to design a highly modular internal structure. This internal structure combines modules into sequences at several levels to provide a general framework for controlling and manipulating I/O devices.

At the bottom level of the structure are filter modules, which are lightweight, independently schedulable subprocesses within a VMS process. Filter modules communicate with each other by means of I/O routines and a shared data structure containing job information. Pointers to the I/O routines and shared data are supplied in the invocation of the filter module. The effect of the stream I/O routines is much like that of pipes in the UNIX operating systems.

At the next higher level is a set of communicating filter modules; each stream of filter modules is called a job step. Finally, a module called the print job analyzer combines a sequence of job steps to handle a complete print job.

### *Filter Modules and Job Steps*

Filter modules can read input from a preceding filter module and write data to a succeeding filter module. Filter modules may perform functions such as reading a file, converting carriage control, translating data syntax, or writing data to the printer. A filter module receives as arguments an input stream and an output stream, like a UNIX process, and a shared data structure, unlike a UNIX process. A simple filter module reads data from the input stream, processes data, and writes data to the output stream.

A filter module may condition its operation based on information from the shared data structure or the contents of the data stream. For example, a translator filter module might format data based on the page size, margins, and aspect ratio specified in the shared data structure, or based on control sequences in the data stream, or both.

Not all filter modules use the input or output streams. The file reader filter module reads from the file instead of the input stream. Similarly, the device output module writes to the printer instead of the output stream.

A job step is a set of filter modules piped together to perform one complete subtask. A subtask may be as simple as "create a separator page" or as complex as the sequence "read a file, perform carriage control conversion, add /HEADER, translate from ANSI data syntax to PostScript, and write the result to the printer." A print job is a set of job steps that performs all functions the user requests explicitly or implicitly. The CPS facility that translates selected printer data syntaxes into the PostScript language is discussed in the section Data Syntax Translation.

### *Print Job Analyzers*

To simplify the addition of new printers and new classes of printers, CPS contains a software structure that corresponds to the hardware mechanisms of a printer.

A print job analyzer (PJA) determines which job steps are required to process a job. CPS includes a separate print job analyzer for each major class of printer that it supports: serial PostScript, PrintServer, and LN03 Image printer devices. When the symbiont begins execution, a PJA is chosen based on the type of device associated with the queue. This PJA is used until the symbiont is stopped. If a terminal device, such as a TT or TX or LT device, is associated with the queue, then the PJA for a serial device is invoked. If an LD device is used, then the PJA for an LN03Q printer is chosen. Otherwise, the PJA associated with PrintServer devices is used.

Each PJA contains a list of all job steps required to execute a job on the class of printers it supports. The PJA selects the job steps it needs from this list, depending upon the instructions received from the queue manager.

Job steps are linked together. The first job step chosen by the PJA is linked to the termination of the PJA itself; when the PJA finishes compiling the job, it terminates, thus starting the execution of the job. At the beginning of each job step, each filter module is assigned stack space and a stack frame. Its initial program counter address and arguments are stored in its saved registers for process activation.

CPS uses a piped stream I/O mechanism similar in function to a UNIX stream; a filter module's input comes from the output of the previous module, and its output becomes input to the following module. By convention, the first filter module of the job step is activated first in the job step; when a filter blocks for output, the next filter module is activated. That filter module then runs until it blocks for input or output, at which point the previous or following filter module is activated.

**Table 1 Simplified Job-step Sequence**

| Job Step | Function |
|---|---|
| init_ps_device | Ensure the device is "fixed up." |
| check_prologues | Ensure that persistent prologues are loaded. |
| sheet_count | Get the beginning page count. |
| job_burst | Print job burst page. |
| sheet_size | Get the current sheet_size. |
| wait_sheet_size | Wait for the sheet_size before continuing. |
| file_setup | Send any file /SETUP modules. |
| get_vmbytes | Get the amount of local printer memory available on the printer. |
| wait_vmbytes | Wait for the local printer memory message from the printer. |
| file_out | Read the file to print and send it to the DECansi translator. |
| sync | Wait for the printer to finish all pages. |
| init_ps_device | Ensure the device is "fixed up." |
| sheet_count | Get the ending page count. |
| wait_sheet_count | Wait for the page count to come back. |
| job_trailer | Print the job trailer page. |
| sync | Wait for the printer to finish the job-trailer page. |
| disconnect | Release the printer. |

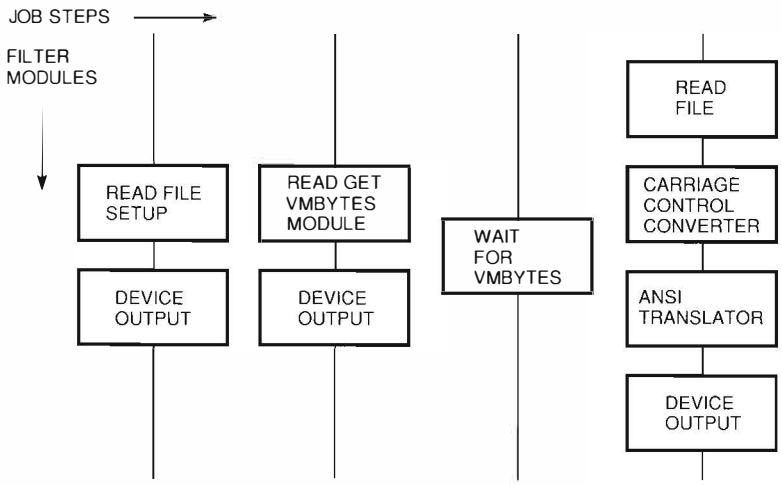
Table 1 shows a simplified listing of the job steps compiled by the serial PJA to process a simple job: one file to be printed in ANSI mode. Each of the job steps shown contains one or more filter modules piped together. For example the job-burst job step has two modules piped together: the job-burst module and the write-to-printer module. Figure 2 shows several job steps with several filter modules each.

If an error occurs at any point in the processing of a job, CPS skips job steps until it reaches the identified error job step set by the PJA. In Table 1, the error job step points to the sync job step that precedes the job-trailer job step. In this case, CPS resynchronizes with the printer and prints the job-trailer page, including the error message.

## *Event Handling*

In addition to the output side of processing a job, there is a corresponding input side. The input side reads messages from the printer, parses them, and notifies the appropriate handler of the event. The handler is chosen based on the type of message sent.

- CPS internal messages are dispatched to the appropriate symbiont routines. For instance, printer resource messages contain information that affect CPS internal operations: paper size is stored for later use by layup (the general mapping of page images to sheets) and translators; virtual memory size is stored for translators; and page count is stored for later use in accounting.




Note that data flows from top to bottom and job steps progress from left to right.

*Figure 2 Job Steps and Filter Modules*

- Printer status messages are dispatched to the operator and, in some cases, to the current user. CPS uses the normal VMS OPCOM notification mechanism to send messages to the system operator. If the user specified /NOTIFY in the print instructions, then CPS uses the VMS $BRKTHRU system service to send the message to the user also.

  In some cases, printer status messages require additional processing. For example, paper jams require special handling on some printers: since CPS cannot determine how many pages were lost in the jam, it invokes human intervention by placing the job on hold. The operator or user can determine what parts of the job, if any, to reprint.

- Program status messages and user data messages are dispatched to the job log. If the user specified /NOTIFY, then they are also displayed with the $BRKTHRU system service. These messages may be printed or logged.

The input and output sides of the symbiont run asynchronously most of the time, but occasionally it is necessary for the output side to wait for a message from the printer. This synchronization between the input side and output side of the symbiont is accomplished by an internal event-signaling facility. When synchronization is required, the output side waits for a specific named event and the input side signals that event when it is detected. For example, at the end of a job, CPS needs the final printer sheet_count in order to calculate the sheet_count for the job; this count is printed on the trailer page and stored in the VMS accounting records. When CPS needs the sheet_count, the output side waits for an event named sheet_count. The input side parses the incoming sheet_count message, stores the returned value in the shared data structure, and signals the sheet_count event. The processing of this event is asynchronous: at the time the message comes in, the output side may or may not have stalled while waiting for the sheet_count event. If the output side was waiting for that event, it is scheduled for further processing; if the output side was not waiting, the event is remembered, in case the output side attempts to wait for this condition in the near future.

In the next section we describe the ways CPS is controlled and managed in the VMS printing system and how it expands printing capabilities in the VMS environment.

## The VMS Printing System Environment

CPS functions as a component of the VMS printing system at the level of printer supervisor. As such, it interacts with, and is shaped by, the other components of the VMS system. The term printer supervisor is used in this paper to be consistent with the terminology of the emerging International Standards Organization (ISO) Document Printing Application draft standard, ISO/IEC DIS 10175.

### Components

The VMS Batch/Print system is a general queue management service, capable of queuing, scheduling, and executing jobs in response to a variety of user-specified instructions.[1] On the VMS system, the printing instructions are stored in a print job object, which is placed in a queue of jobs for a printer. Modern print jobs often resemble batch jobs, due to complex stored processing instructions and the heavy computing load placed on graphics printer controllers.

The VMS printing system contains components at most levels of the DECprint architectural model.

- User printing interface. The VMS system includes interactive Digital Command Language (DCL) interfaces for printing and managing print jobs, printers, and the printing system itself.[2] For DECwindows applications, the DECwindows Print Widget provides a graphical interface that permits users to specify all the options for printing, and the ALL-IN-1 application provides character-cell menus for choosing print options, including the enhanced options offered by CPS.

- Job submission interface. The VMS system includes program call interfaces that give the program all the capabilities of the DCL user interface.[3]

- Print client and service for remote printing. The distributed queuing services product currently provides transparent remote printing in networks using a proprietary network protocol.

- Print spooler. The VMS Job Controller, recently replaced by the VMS Queue Manager, functions as queue manager and scheduler. (The function of spooling printer data to temporary files is performed by the VMS file system and is transparent to most components of the printing system.)

- Printer supervisors. The VMS system provides two standard symbionts to support most line

printers and serial printers. PRTSMB supports printers attached directly to communication ports on the CPU, e.g., the printer port on a VAX workstation. LATSYM provides support for printers attached to the serial or parallel ports of DECserver network communications servers. For PostScript printers, CPS is used instead of these standard symbionts.

The VMS printing system also contains components that affect CPS processing.

- Device control libraries are collections of small text sequences that can be inserted into the data stream from the symbiont to the printer. The sequences are ideally organized into text libraries containing named modules, with a separate library for each type of output device. Device control modules can be associated with a printer queue by the system manager as part of a FORM definition or a job reset function, or accessed directly by the user with the /SETUP qualifier.

  Device control libraries frequently contain device-specific control sequences that alter the format of the text and pages, for example, setting printer paper margins, setting character pitch, or enabling landscape printing. They may also contain downloadable font data or preprinted data for each page.

- VMS form definitions contain page size and margin specifications that guide the print formatting process for a print job. The user can also specify page setup strings and can prohibit the symbiont from wrapping lines during processing.

### *VMS Print Queues*

VMS has several distinctly different types of queues. Execution queues process jobs through a symbiont, and generic queues transfer jobs to other queues. Often generic queues are used for load balancing: one generic queue may feed several printers of similar capability and location.

CPS also uses generic queues in an unusual way. Default attributes can be specified for generic queues that cause all jobs submitted through the queues to inherit certain default print instructions. For example, a queue can be established that, by default, assumes that jobs are PostScript documents, or assumes that jobs should be printed in landscape orientation. This ability to set default queue attributes is essential for supporting applications that can specify the queue name for a print job, but cannot specify certain other qualifiers such as DATA_TYPE. It can also permit users of old applications to access new features of the printing system.

### *VMS Print Commands and Interfaces*

The VMS printing system is manipulated through DCL commands and qualifiers. Many of the qualifiers are handled by the queue manager and have no impact on the operation of print symbionts; others directly affect the operation of CPS.[2] The VMS system also supplies a call interface to these functions.[3]

### *VMS Interfaces to Symbionts*

The VMS Job Controller/Queue Manager provides two interfaces for customizing print symbionts: the PSM module-replacement interface, and the SMB server symbiont interface. CPS is currently implemented as a single-stream symbiont through the SMB interface.

The SMB interface permits a user to replace the flow of control of the symbiont with a separate process. The process may be written in any style and structure suitable to the task at hand, and need follow only certain minor guidelines with respect to the operating system environment. To use the SMB interface, we replaced the entire symbiont process. The result was much greater flexibility, but we were required to write more program code.

The SMB interface provides services to the symbiont process through subroutine entry points and callbacks that pass messages between the symbiont and the VMS queue manager. Messages from the system to the symbiont specify functions such as start up, shut down, begin job, pause, resume, and interrupt. Messages from the symbiont to the system return information such as job status, job completed, device status and error information, and checkpoint and accounting data.

## *Range of Printers Supported*

CPS currently supports the full range of PostScript printers supplied by Digital, from a low-speed color printer up to a 40-page-per-minute laser printer that can handle 11 different paper sizes.

### *Special I/O Processing*

CPS supports several different means of communication with the printer: serial, Ethernet, and a special high-speed video connection.

The serial connection may be either a direct connection between the computer and the printer or a local area transport (LAT) connection by which printer is attached to a serial port of a DECserver terminal server. The two methods differ only in the way jobs are started and terminated. For LAT-connected printers, CPS must establish and dismiss the LAT connection at the start and end of each job.

Once the connection is established with the serial printer (via LAT or direct connect), CPS begins a dialogue with the printer using an asynchronous serial line protocol and PostScript programs. The asynchronous serial line protocol, defined by Adobe Systems Inc., consists of five control characters that alter or query the state of the printer.

The symbiont forces the printer into an idle state by a series of control/T, control/C, and control/D characters. When a control/T results in an IDLE message from the printer, the symbiont and printer are ready to process a job.

PrintServer printers on Ethernet networks are DECnet nodes. To write to a PrintServer printer, CPS establishes a DECnet task-to-task session at the beginning of the job. The dialogue required for synchronizing serial printers is not necessary for the Ethernet printers; the PrintServer protocols provide synchronization and device control operations through separate control channels.

Printers connected through Ethernet use several protocols, which are layered on DECnet task-to-task communications. The protocol used depends upon the version of the PrintServer code.

The local area print service (LAPS) protocol was developed for the PrintServer family and is still in use. The Common Printer Access Protocol (CPAP) will replace LAPS in all PrintServer printers.[4] PAP is based on the earlier Reid-Kent protocol, Internet Socket 170, and is being discussed as a possible new Internet standard.[5]

## *Special Processing for "Dumb" Printers*

In some printer configurations, it is economical to use the workstation or CPU as the printer controller. In this case, the printer includes only the print engine and media handling and finishing equipment, and none of the electronics, computers, and interpreter programs that render the graphics language into the elements required by the print engine (usually an array of pixels). Such a "dumb" printer is physically connected to the computer by a very high-speed link such as a direct video connection or data bus. For such a controllerless printer to be generally useful, the printing system must emulate an existing class of printer.

The LN03 Image printer (LN03Q) is a bit-map printer of this type. It uses a special high-speed DMA bit-map interface that plugs into a Q-bus and provides the speed required for printing scanned images. The protocol between this interface and the printer consists of bit maps and a small amount of status and synchronization information.

The engine itself includes only the laser imaging and paper handling equipment. CPS handles the rest of the controller functions in the host computer. Because of the level of support and emulation provided, the LN03Q printer appears to be an ordinary PostScript job printer with some special image capabilities.

For a given print job, CPS performs the normal processing up to the point at which the PostScript language data stream would normally be sent to the printer. At this point, CPS directs the data stream to a special PostScript interpreter subroutine that produces a bit-map image of the printed page in memory. The bit-map image is then sent to the printer through a special LNV21 direct memory access I/O interface on the Q-bus.

The software for the LN03Q printer also has one special processing path. The LN03Q printer is intended as an image printer for bit-map images. CPS supports image files containing page images that are scanned or precomputed at device resolution (300 dots per inch) and optionally compressed with Comité Consultatif Internationale de Télégraphique et Téléphonique (CCITT) Group 3 (1D) or Group 4 (2D) compression methods. Image files can be transmitted directly to the printer without converting to PostScript. Image files can only be sent directly to the printer if they are printed one page per sheet; if the user requests printing multiple pages per sheet, or other layup functions, then the image is processed through the PostScript interpreter.

Image files are structured in Digital document interchange format (DDIF), which expresses text, graphics, and images together. Files intended for the LN03Q printer must contain only image bit maps.

If the print job specifies DATA_TYPE=DDIF or the file is a DDIF file, then CPS examines the file in a special mode. If the file correctly contains only image bit maps, CPS decompresses the images in memory if necessary, using the DECimage Image Support Library routines, and then sends the uncompressed bit map directly to the LN03Q print engine. Thus

the image goes directly to the printer without passing through the PostScript interpreter.

## Special Processing in CPS

CPS includes a number of special features and functions to satisfy the requirements of the DECprint architecture and the VMS printing system. In this section, we discuss the features that extend the process of standard print symbionts or are completely new.

### Reading Print Instructions

CPS reads the print instructions for a job from the VMS queue manager through the SMB$READ_MESSAGE and SMB$READ_MESSAGE_ITEM functions of the SMB interface. Print instructions are expressed as attributes with values. Each attribute has an associated numeric code and symbol, called an item code, and a value of a specific data type. The symbiont reads each item code and value, and stores the information in a static data structure. The information is used later to determine the processing sequence for the job, special information to be displayed on separator pages, and so forth.

### Bidirectional Communication with PostScript Printers

CPS requires a full duplex communications path to PostScript printers since they report many conditions by sending messages to the host computer. These messages include device status messages, program status and error messages, user data messages, and replies to CPS inquiries.

CPS also requests information from the printer for synchronization, formatting, and accounting purposes. For instance, to determine how to format ANSI text, the symbiont needs to know what paper is loaded in the printer.

CPS receives the messages from the printer and parses them to determine what it should do with the message. If the message is device status, then CPS routes the message to the operator and/or the user whose job is being printed. If the message is an internal CPS communication, then CPS processes it. Otherwise, the message is either a program status message or a user data message. In either case it is logged for the user.

All messages are parsed except user data messages. Messages from the printer's interpreter are converted to a standard format that would, if desired, permit the message to be translated into the user's native language.

### Data Syntax Translation

CPS provides a facility that translates selected printer data syntaxes into the PostScript language. The translating programs are subroutines, some quite large and complex, that accept a data stream in one format and produce a data stream in another format. The translators are responsible for all formatting, including sheet size, page orientation, aspect ratio, and type sizes; CPS is responsible for all I/O and coordination with the printer. The translation facility currently supports the following printer data syntaxes: DEC PPL3, ReGIS, Tektronix 4010/4014, and PCL Level 4.

The translation facility has several restrictions. A file may consist of only one data syntax, and all files in a job must be of the same data syntax.

In general, CPS performs the translation from one data syntax to another on the host computer. In this way, simple printers that support only the PostScript language internally can be extended to support a number of printer languages. This reduces the requirement for a complex printer controller that supports multiple data syntaxes internally. Host translation can guarantee consistent use across jobs of the printer's internal fonts, page orientation, finishing equipment, and page layup The general mapping of page images to sheets supplied as part of CPS requires that the printer operate in PostScript mode. To ensure consistent use of fonts and consistent positioning of pages with respect to finishing such as duplexing and stapling, all language translation must be done by the symbiont.

### Page Layup Multiple Pages per Sheet

Page layup is the process of printing more than one page image on a sheet of paper. When more than one page image is placed on a sheet of paper, the images are rotated and scaled to fit on the page, but are altered in no other way. The layup facility works with all data types, including PostScript and PCL data syntaxes. Layup also permits formatting for larger paper sizes and then printing on smaller sheets.

Layup is invoked explicitly with one or both of the extended qualifiers NUMBER_UP and LAYUP_DEFINITION. NUMBER_UP specifies the maximum number of page images that will be printed on a single side of a sheet; for example, two-up printing is specified by the "NUMBER_UP=2" option. Two or four page images per side may save significant quantities of paper for draft printing, handouts, and the like. Up to 100 page images may be placed on a

single sheet of paper for thumbnail draft printing to review the overall layup of a document.

Layup may also be invoked through a combination of PAGE_SIZE and SHEET_SIZE with NUMBER_UP. For example, the combination of PAGE_SIZE=E, SHEET_SIZE=A,NUMBER_UP=1 permits printing draft copies of large-format documents on small paper. Conversely, the combination of PAGE_SIZE=A,SHEET_SIZE=B,NUMBER_UP=1 magnifies the smaller page to fit the larger sheet.

### *Duplex Printing*

Printing on both sides of the paper introduces a number of new options and interactions that require special processing in CPS. CPS begins each document on the first side of a new sheet, so that recto and verso (right-hand and left-hand) pages and alternating margins are aligned with the correct sides of sheets as they are stacked by the printer. This function also interacts with the direction in which the medium is physically loaded into the printer if the medium is not symmetric left-to-right, top-to-bottom, or front-to-back, such as predrilled paper.

The interactions of PDL coordinate systems, page layup, media selection, asymmetric media, duplex printing, and binding are the most elusive engineering problems in the printing application space. No general model of these interactions has been developed, despite considerable effort in standards committees. It appears that it is necessary to implement every possible option.

### *Separator Pages*

CPS prints all the separator pages defined by the VMS queuing system as well as some generated by CPS. Flag, burst, and trailer pages for job and file levels are available as defined by VMS, and contain the same information presented in a highly legible format. In addition to the standard VMS information, the job trailer page also contains the first two PostScript language errors returned from the printer. This often makes it unnecessary to use MESSAGES=PRINT to see simple errors.

To ensure that the job separator pages can always be printed correctly, CPS resets the PDL interpreter in the printer before printing these pages. The CPS-generated separator pages do not alter the coordinate system of the interpreter; the user's document starts printing with the default PostScript state. File separator pages, in contrast, print in the current PostScript environment, including the altered page geometry, e.g., layup established by the print job.

CPS defines two new separator pages. The file error page is printed when a file cannot be opened or an error occurs while reading the file. The file error page informs the user of the error condition which caused it to be printed. The job log page contains up to 40 lines of the job log file. The job log file contains job events such as job start and job completion as well as program status messages and user data returned from the printer.

### *Managing Printer Resources*

Once communication is established with the serial printer, the symbiont must establish what resources are available on the printer. These resources include prologues, which are commonly used PostScript routines, the amount of available virtual memory, and the medium in the default paper tray. For example, CPS persistently loads the PostScript prologue for the output of the ANSI text translator into the PostScript interpreter. This resource might be lost to the printer because of a power failure or might become obsolete due to a software upgrade. CPS interrogates the printer at the beginning of any job requiring the translator prologue and loads a new prologue, if necessary. CPS also performs similar processing for the PostScript prologue that is used to generate the separator pages.

For traditional resources such as paper, CPS relies on status messages from the printer to indicate that the printer is stopped because paper supply is empty or jammed. These conditions are relayed to the operator and to the current user by standard VMS mechanisms.

### *Library Search Lists*

In the standard VMS print symbiont, only one device control library may be associated with a queue. This is not a problem since the standard VMS print symbiont deals with only one data syntax. (Recall that device control libraries are often written in device-dependent data syntax.) CPS, on the other hand, uses more than one data syntax when printing a non-PostScript job: the data stream to the printer is PostScript, but the data stream to the translator is in another data syntax.

Early versions of symbionts that supported PostScript suffered from the same restriction: only one device control library was available, and its

modules were expressed in PostScript. This made it impossible for users to share device control libraries with their standard VMS print symbiont and their non-PostScript printers.

To solve the problem of multiple data syntaxes in a job, CPS introduced device control library search lists. The system manager, rather than specifying a single file specification in the INITIALIZE/QUEUE/LIBRARY command, creates a logical name instead. CPS translates that specific logical name and uses each element of the result as a device control library. Each library in the search list can have a data syntax associated with it by adding the qualifier, /DATA_TYPE=.

CPS supplies a device control library, CPS$DEVCTL, which must be included in the search list, usually as the first, or only, element in the search list.

## Summary

The DECprint model of printing describes a useful structure with consistent functions and responsibilities. CPS is an advanced print symbiont that runs in the VMS printing system. It includes many specialized functions to support the features of a wide range of modern printing devices. It provides, we feel, an extraordinary level of support. It was designed with a highly modular and flexible internal structure to permit enhancements to be engineered with minimal interactions with current operations.

CPS is currently shipping its fourth version. This version fully supports the ten different PostScript printers supplied by Digital, which range from a low-speed color printer to a high-speed laser printer. It also supports five different data syntaxes in which applications can write documents. We expect that more printers and more capabilities will be added in future versions, and that CPS will require a minimum of additional engineering effort due to its very general internal structure.

## Acknowledgments

We would like to thank Peter Conklin for actively initiating CPS and Gary L. Brown for even more actively expanding it. We would also like to thank past and current CPS developers: Ned Batchelder, Cathy Callahan, Mark DeVries, Rich Emmel, Dave Gabbé, David Larrick, Klara Levin, Mary Marotta, Doug Stefanelli, and Charlotte Timlege. We would like to thank Bill Fisher for his extensive comments on this article. Finally, we thank the many sites and people who have tested the DECprint Printing Services software.

## References

1. *VMS Utility Routines Manual* (Maynard: Digital Equipment Corporation, Order No. AA-LA67B-TE, 1990).
2. *VMS DCL Dictionary,* 2 vols. (Maynard: Digital Equipment Corporation, Order Nos. AA-PBK5A-TE and AA-PBK6A-TE, 1991).
3. *VMS System Services Reference* (Maynard: Digital Equipment Corporation, Order No. AA-LA69A-TE, 1991).
4. J. Jones, A. Kachrani, and T. Powers, "The Common Printer Access Protocol," *Digital Technical Journal,* vol. 3, no. 4 (Fall 1991, this issue): 55–60.
5. B. Reid and C. Kent, "TCP/IP PrintServer Print Server Protocol," *Western Research Lab Technical Note TN-4* (Maynard: Digital Equipment Corporation, 1988).

## General References

*Guide to Maintaining a VMS System* (Maynard: Digital Equipment Corporation, Order No. AA-LA34A-TE, 1990).

*DECprint Printing Services User's Guide* (Maynard: Digital Equipment Corporation, Order No. AA-PBZGA-TE, 1991).

*DECprint Printing Services System Manager's Guide* (Maynard: Digital Equipment Corporation, Order No. AA-PBZFA-TE, 1990).

*Digital ANSI-Compliant Printing Protocol Level 3 Programming Reference Manual* (Maynard: Digital Equipment Corporation, Order No. EK-PPLV3-PM, 1991).

*Digital ANSI-Compliant Printing Protocol Level 3 Programming Supplement* (Maynard: Digital Equipment Corporation, Order No. EK-PPLV3-PS, 1991).

*PostScript Translator's Reference Manual for ReGIS and Tektronix 4010/4014* (Maynard: Digital Equipment Corporation, Order No. AA-PBWFA-TE, 1991).

*PostScript Printers Programmer's Supplement* (Maynard: Digital Equipment Corporation, Order No. EK-POSTP-PS, 1991).

*PostScript Language Reference Manual,* 2nd ed., Adobe Systems Incorporated, ISBN 0-201-18127-4 (Reading, MA: Addison-Wesley, 1990).

*Information Technology—Text and Office Systems—Document Printing Application (DPA),* ISO/IEC JTC1/SC18 N, Draft International Standards 10175-1 and 10175-2 (September 1991).

*CDA Base Services Technical Overview* (Maynard: Digital Equipment Corporation, Order No. AA-PHJYA-TE, 1991).

*Creating Compound Documents Using CDA Base Services* (Maynard: Digital Equipment Corporation, Order No. AA-PHK2A-TE, 1989).

*Writing Converters Using CDA Base Services* (Maynard: Digital Equipment Corporation, Order No. AA-PHK1A-TE, 1991).

*CDA Base Services Reference Manual,* 2 vols. (Maynard: Digital Equipment Corporation, Order Nos. AA-PHJZA-TE and AA-PHK0A-TE, 1991).

*CDA: DDIF Technical Specification* (Maynard: Digital Equipment Corporation, Order No. AA-PHK3A-TE, 1991).

*CDA: DTIF Technical Specification* (Maynard: Digital Equipment Corporation, Order No. AA-PHK4A-TE, 1991).

*DDIS Syntax Specification* (Maynard: Digital Equipment Corporation, Order No. EL-00081-00-1, 1987).

*James D. Jones*
*Ajay P. Kachrani*
*Thomas E. Powers*

# *The Common Printer Access Protocol*

*The DEC PrintServer Supporting Host Software version 4.0 incorporates Digital's first implementation of the new common printer access protocol (CPAP). This protocol is compatible with the local area print server (LAPS) protocol, which was optimized for VMS access and DECnet transport, and with the Reid-Kent protocol, a PostScript-based, TCP/IP-connected print server for a client-server environment. The CPAP protocol supports a variety of data presentation protocols and allows printers to be connected to driving applications by various communications and process-to-process interfaces. The protocol also couples entities running different operating systems across disparate networks. Because of its superior performance, the new CPAP protocol has been accepted by the Open Software Foundation for inclusion in a future release of OSF/1.*

The presentation of computerized data has become a remarkably sophisticated and subtle operation. Video displays now support windows with complex allocations of display space, variable fonts, and real-time user input operations. Printing devices now offer support for publication-quality fonts, line art, and images. These devices can present visual objects on a variety of media, from many sources, and in variable orientations and presentation modes. In addition, both video and printing devices are now decoupled from dedicated computing environments, and are shareable from many hosts and by many users or programs.

Now, only the simplest printing devices are limited to presenting just characters, and many users are finding such restricted capabilities inadequate. Also, most printing devices still require dedicated connections to single computers. However, more printers now offer full network accessibility; i.e., network printers are capable of offering sophisticated services to a wide variety of users and their applications.

The paper entitled "Design of the DECprint Common Printer Supervisor for VMS Systems" in this issue of the *Digital Technical Journal* describes access methods and interrelations among services that provide for these increasingly sophisticated data presentation capabilities.[1] The printer access protocol (PAP), a service interface in the DECprint architecture, couples the printer supervisor component to the logical printer for presenting data and otherwise controlling a physical printing device. The common printer access protocol (CPAP) described in this paper provides the fundamental services required by a printer supervisor for the presentation of data and collection of accounting information. In addition, the CPAP supplies easier network access between printer supervisors and printers, as well as ancillary control of printers for network management and device configuration. The CPAP also provides services to distribute the processing requirements of the printer itself, most notably a mechanism for delivery of network font services. This last capability allows a printer to offer what amounts to virtual services, i.e., the ability to configure itself dynamically to the demands of a print job without the involvement of the printer supervisor.

This paper begins with a discussion of the influence of existing protocols and the DECprint architecture on our CPAP design goals. The sections that follow present the printer session concepts and the functional interface between the protocol and applications. We then describe the implementation of the new protocol in a server environment, including interoperability, compatibility, and the translation of the older PrintServer protocol. At the close of the paper, we discuss ongoing standardization issues.

## *History*

The PrintServer 40, Digital's first fully networked printer, was first shipped in 1986. Its local area print server (LAPS) protocol was analogous to later

printer access protocols. The PrintServer 40 was a ground-breaking product for Digital, and the LAPS protocol was a major aspect of the PrintServer development effort, portions of which date back to 1983. The LAPS protocol was designed and developed with particular product-oriented deliverables in mind, and was optimized for VMS access and DECnet transport. While this protocol predates much of the architectural work now being implemented in Digital's printing products, it was (and still is) a significant element of PrintServer architecture and implementation.

Work began on more general PAPs in 1987 as part of the early work on the DECprint architecture (known at the time as the Printing Systems Model). The specifics of what would become the CPAP emerged in late 1988 in two internal papers by Brian Reid and Chris Kent of Digital's Western Research Laboratory. These papers presented the initial design concepts for a PostScript-based, TCP/IP-connected (transmission control protocol/internet protocol) print server in a clearly defined client-server environment. This print server protocol came to be known as the Reid-Kent protocol.

### *Design Rationale and Goals*

By early 1988, design goals for (and constraints on) a PAP were well understood, and had been collected and published as part of Digital's Printing Systems Model. Chief among these goals and constraints was the need to support a variety of data presentation protocols, and to allow printers to be connected to driving applications by a variety of communications and process-to-process interfaces.

The increasing corporate commitment to open systems made it clear that a PAP would also have to couple entities running various operating systems across different networks. Thus, the DECprint PAP architecture team decided early in the design process that a PAP should be designed for public access; that is, the specification for the protocol should be put into the public domain and submitted for industry standardization.

Interoperability is a most serious constraint. Digital has a strong tradition of maintaining backward compatibility within and among its product families. In a distributed processing environment, however, backward compatibility takes on the added burden of interoperability. Multiple clients must communicate with multiple servers, any of which can be upgraded to new versions of supported protocols asynchronously. Addressing this problem was a major conceptual test in the first implementation of a CPAP server. This is discussed in more detail in the section The CPAP Server Implementation.

The Reid-Kent protocol met many of the technical design requirements for a new PAP. It was built on industry-standard components, and contained no proprietary technology that would prevent its publication.

However, certain PAP design goals were not covered by the Reid-Kent protocol in its 1988 version.

- There was no facility to select a specific page description language (PDL) for printers supporting multiple interpreters.
- There was no method for soliciting the capabilities and media available on the printer.
- The only language supported was English (contrary to the corporate guidelines for internationalization).
- Data sent from the printer was not categorized; user-specific information was mixed with operator and service data.
- No means was provided to solicit the status of the printer.
- There was no encoding to discriminate between binary and text files.

However, these flaws were largely omissions from the design goals, not fundamental conflicts with them. The architecture team decided that the Reid-Kent protocol could be extended to address these omissions without serious conflict. In fact, the necessary extensions were designed to allow clients and servers conforming to the original Reid-Kent protocol to remain in conformity with the full CPAP specification.

## *Architecture*

The CPAP is primarily a communication-oriented protocol, i.e., the presentation of its function is closely coupled with its encoding. The major syntactic features of the CPAP derived from the Reid-Kent protocol are the following.

- All encodings are ASCII strings. This eases the generation of protocol streams and ensures independence from the underlying communications channels.
- No data fields are fixed length. This provides for extensibility of the protocol and eases the generation of a protocol stream.

- Multiple channels of communication use the same basic format. Common parsing of separate channels simplifies implementations.
- Simple numeric tokens define the operators.

### Session Concepts

The CPAP architecture defines separate contexts for each type of work the CPAP can perform. Each context requires that a separate session be established for its own tasks, and each session involves the creation and use of a separate network connection between the controlling client and the server. Each connection identifies the type of session the initiator requires. The CPAP defines three different session types: print, management, and console.

The set of CPAP operators allowed for a session is restricted to those needed to support that type of session. All session types have access to printer status and configuration information. In addition, multiple concurrent sessions are permitted. Print sessions and management sessions may have one or more virtual circuits active to a printer at a time. The use of multiple circuits permits the streaming of data to the printer over logically separate channels, thereby eliminating application protocol overhead for the most frequent operations. In contrast, console sessions use a single virtual circuit for exchange of data with remote terminals.

*Print Sessions* Print sessions usually consist of a series of documents printed for a user on a given host by a printing service (a "printer supervisor" as defined by the DECprint architecture). With the operators provided by the CPAP, the printing service can determine the language interpreters, printer options, fonts, prologues, and media that are currently installed at the server. These operators also provide the current operational state, number of jobs queued to the printer, and the current job status. These features permit the printing service to select the printer (server) that can satisfy the user's request and to determine a method for submitting the job to the printer.

Once the printing service has begun a session and identified itself, it identifies the user and the user's job code to the printer. This information may be used by the printer to provide usage information to a centralized accounting service. The printing service can then present documents to the printer. A transaction between the printing service and the printer establishes which interpreter the printer will use for each document and which virtual circuit will be used for its transmission.

Selection of the proper virtual circuit for transmission of documents to the printer is performed by passing tokens from the printer to the printing service. The tokens are then mapped to whichever virtual-circuit service is being used by both the printing service and the print server. This mapping approach avoids passing network-specific information within the protocol. Not only does the approach make the CPAP independent of the networks on which it might run, it ensures that the network services need no knowledge of CPAP encodings. Such virtual-circuit mapping is critical to allow CPAP client-server processing to be implemented in a heterogeneous, internetworking environment.

During the printing of the document, some data presentation interpreters (PostScript, for example) send data back to the user or print service. In addition, the printer may run out of paper or toner, may have a full output tray, or may encounter other exception conditions not directly related to the interpretation of page description data. The CPAP categorizes such conditions and delivers relevant messages to the user, the operator, or the event logs.

Upon completion of the job, the printing service is notified of the media used, the number of pages printed, and the printer processing time required to complete the job. The protocol also includes a provision to abort jobs, e.g., an improperly formed document that might otherwise hang the printer.

*Management Sessions* The CPAP supports certain printer services through management hosts. A management host is a network entity (not necessarily the same entity as the printing service) with which the printer can exchange information or request services. Such services include

- Time service
- Centralized event logging
- Centralized accounting
- Program loading and configuration
- Font services

An important aspect of the CPAP is that the printer is always passive with regard to initiating management services. A candidate management host advertises that it has services to offer, and a print server accepts or rejects the offer. Once a

connection with one or more management hosts is established, the printer may use such hosts as servers for time synchronization, configuration file access, and font lookup. Additional functions for these hosts may be loading program images, event logging, accounting, and general file access.

File naming to access general file services is a problem that needs special attention if the server and the protocol are to maintain independence from the host operating systems. Commonly used files are identified in the CPAP by reserved tokens, such as $CONFIG, $DEFAULTS, $RESOURCES, and $SETUP. Arbitrary path names are allowed, but can access only a limited domain (from a known root directory) to preserve file system independence and to maintain security.

Translation to the host's services is provided by the host itself. This permits the printer to be served by different hosts using a wide variety of operating systems (and their implicitly different file-naming conventions and syntaxes) without any awareness of a management host's implementation by the server.

*Console Sessions* A console session is a form of printer management. The content of the data exchanged during a console session is specific to the printer, and is not specified by the CPAP. Services performed within a console session might include

- Operator services, such as telling a printer what media have been loaded (e.g., by color, weight, or transparency), or setting physical printer defaults (e.g., duplex versus simplex, or default medium selection)
- Network management configuration services, such as controlling domain access to or from the printer
- Troubleshooting or debugging services

Digital's implementation of console services on current PrintServer products conforms to the Enterprise Management Architecture.

### *Application Program Interface*

The functional interface to any protocol provides an additional abstraction between an application and a protocol. This abstraction answers many of today's software application needs, including interoperability, portability, modularity, and reusability across multiple architectures. An application programming interface (API) that allows access to all CPAP facilities is included in the protocol's specification.

A connection block, which is passed as a parameter to all functions, provides support for various printer types, their device identifications, and descriptors for command and data channels. This support includes separate command and data channels for printers supporting multiple virtual circuits or channels. Just as in the case of the datastream form of the protocol, the API form allows separate channels for data and commands.

A separate command channel allows ease of control flow between client and server. This may include the client receiving the server's status or events, or the client sending aborts to the server. For devices that support only a single channel, the generic printer driver can set both command and data channels to the same value. For supporting multiple jobs active at the same time (job overlap), a job identification (ID) parameter is passed with all functions.

To support various message types, the address of a read-callback routine is passed to the open printer function along with a pointer to read-callback arguments. These arguments may signal various events, or may consist of messages for the user, operator, accounting, or resources available in the printer.

An early version of the generic functional interface was part of MIT Project Athena's Palladium Print System. The printer supervisor in Digital's LN03R ScriptPrinter product was modified to create a generic printer interface for both the ScriptPrinter device and the PrintServer family. This conversion from an API-accessible base took one week to execute, whereas it typically takes six months of effort to develop a new printer supervisor for a device as complex as the PrintServer product.

## *The CPAP Server Implementation*

The implementation of a protocol gives rise to problems different from those related to its design. When defining the architecture, one strives to provide an ideal that includes all of the desired features in an elegant manner. When performing an implementation, one finds that elegance often has to take a back seat to pragmatics. This is especially true when the new protocol is intended to replace two different protocols in a new version of an existing product. Merely implementing the new protocol

is not enough—the implementation must somehow coexist with the protocols being replaced.

Digital's first production implementation of the CPAP was targeted for the DEC PrintServer Supporting Host software version 4.0, which loads and drives the PrintServer family of printers. For the rest of this paper, we refer to this software by the PrintServer product designation of LPS version 4.0.

We started the implementation by modifying Digital's ULTRIX PrintServer client, which already used the Reid-Kent subset of the CPAP, to use DECnet network transport and run on the VMS operating system. We then updated the LPS server code to permit either DECnet or TCP/IP transport. This was accomplished by using the direct-to-port communication features of the VAXELN operating system. The server establishes a circuit using the appropriate transport and then spawns a process for dealing with each incoming connection. Thus, the same code can service print sessions, management sessions, and console sessions without concern for the type of network transport.

The CPAP was, by design, directly upward-compatible with the Reid-Kent protocol subset. However, Digital's PrintServer offerings prior to LPS version 4.0 were LAPS-based, and LAPS was not CPAP-compatible. To permit users of existing PrintServer printers to continue to use these products, we had to find a way for the new CPAP implementation to coexist with the older LAPS application protocol. We achieved this coexistence by having the server perform translations from the older protocol to the new one in the server itself. When the client establishes the initial connection, the server senses which protocol is being used by the client system. If the initial message indicates the use of LAPS, the server spawns incoming and outgoing filters to deal with the incoming connection, and a new internal circuit replaces the network connection to handle the interpretation of the CPAP.

The coding of the LAPS filters was the last step in implementation before testing began. The PrintServer 20, PrintServer 40, PrintServer 40 plus, and the new turbo PrintServer 20 all had to be tested using both LAPS and the Reid-Kent subset of the CPAP. In addition, the new implementations of the management client and the console client on the VMS system required verification. This verification entailed a multitude of tests using the LPS symbiont running on older versions of the VMS operating system, the newer common print symbiont (CPS), several versions of the ULTRIX operating system, and a source kit version running on a Sun Microsystems workstation.

Unfortunately, this testing uncovered latent defects in the implementation of the existing products. We had to analyze each of these defects and plan corrective action. Since updating the existing products in the field is a difficult process (both technically and procedurally), we corrected most of the defects by altering the server to deal with the problems. Retesting was performed over several baselevels to ensure that our changes caused no regression.

At one of the early baselevels, the interface between the network distribution software and the server's PostScript interpreter was updated to use a stream-based connection in place of the previous packet protocol. This update permitted the new CPAP data channel to be mapped by reference to the input of the PostScript PDL or any other PDL supported by the printer. This change alone permitted the performance of the server to be maintained even when the server was translating from the old LAPS protocol to the CPAP.

In general, development proceeded incrementally, i.e., key features were identified and added with each baselevel. While this technique limits the complexity of producing the product, it raises an important business issue. Specifically, the provision of enhanced services in a client-server environment often exposes aspects of the proverbial "chicken-and-egg" situation. There is little call to offer enhanced features in a server if clients have not been programmed to solicit the features. However, clients are not readily upgraded to solicit features that might not be widely available.

The LPS version 4.0 project team met its backward compatibility design goals by including the LAPS-to-CPAP filters. In doing so, they undercut the need to provide the enhanced feature support that the CPAP was designed to deliver, since existing clients (earlier versions) could not avail themselves of the added features. In addition, the risks of including full CPAP support in LPS version 4.0 (possible increase in time to market, and the creation or exposure of more latent defects in all supported environments) seemed to outweigh the benefits. However, a last-minute change to use the new protocol's data channel for loading fonts yielded such a large performance advantage that resistance to using the new features crumbled, and the project team was allowed to submit the full protocol to field test.

## Standardization

Network printing became widely available in the mid-1980s, but products from different vendors were not compatible. Network printing protocols were largely proprietary efforts by vendors who had developed them for their own printer products. Digital's PrintServer 40 and its LAPS protocol were typical in this regard. By the late 1980s, network printing was an established and competitive technology, but there was still little interoperability among the various vendors' products.

In the absence of printing protocol standards, the Internet Engineering Task Force (IETF) formed a Network Printing Protocol working group in early 1990. This group's charter was to examine printing protocols then in existence or under development, assess their applicability to Internet-wide use, and suggest changes. Digital's representatives to the IETF working group on the Palladium Printing Systems standardization reported the interest shown in Digital's Reid-Kent protocol. Thus, in July of 1990, Digital submitted a version of the PAP that was under consideration by the DECprint PAP architecture team.

Early consideration of this PAP by IETF and the LPS version 4.0 implementation effort ran concurrently. This provided a unique opportunity for Digital's implementers to obtain feedback from a very knowledgeable architectural community. In turn, they could report implementation experiences that affected the review and progress of the specification towards standardization. Implementations of CPAP clients and servers by companies other than Digital are in progress.

As part of Project Athena's Palladium Printing System, the CPAP has been accepted by the Open Software Foundation for inclusion in a future release of OSF/1.

A draft of the CPAP is being circulated among Internet members for comment. Meanwhile, work on future enhancements continues. Work is now in progress to specify a superset of the existing protocol that deals with authentication and encryption to strengthen security. This work is being done in the spirit of the original migration from the Reid-Kent protocol to the CPAP; i.e., the security features being added will not adversely impact users who do not need the new features.

## Acknowledgments

The CPAP effort has been the work of many developers. Chris Kent and Brian Reid drafted the base architecture and created the first prototype implementations. Jim Jones championed the protocol in the DECprint PAP architecture team (Alan Guenther, Tom Hastings, Jim Jones, Tom Powers, and Eric Rosen) and coded the LPS version 4.0 server. Carol Gallagher wrote the LAPS filters to translate from the old protocol to the new. Mike Augeri and John McLain ported the management and console clients to the VMS system from the ULTRIX system. J. K. Martin rewrote the Berkeley Software Development (BSD) source kit to use the new protocol. Ajay Kachrani developed our ULTRIX and MIT Athena clients and represented the protocol during early phases of the IETF standardization effort. Many others supported these efforts, and others are yet beginning to develop new CPAP clients. We thank them all for their efforts.

## Reference

1. R. Landau and A. Guenther, "Design of the DECprint Common Printer Supervisor for VMS Systems," *Digital Technical Journal*, vol. 3, no. 4 (Fall 1991, this issue): 43–54.

*Guido Simone*
*Jeffrey A. Metzger*
*Gary Vaillette*

# *Design of the Turbo PrintServer 20 Controller*

*The turbo PrintServer 20 controller is a performance enhancement of the original PrintServer 20 system controller. The turbo controller was developed to enable PostScript code to execute faster and thus improve page throughput for complex documents. The RETrACE analysis system was designed to analyze the performance of the original PrintServer 20 system and estimate expected performance future systems. The turbo controller's processor and its three subsystems for memory, write buffer, and bit-map data transfer were selected based on the analysis results. Performance tests conducted on both the original and the turbo PrintServer 20 indicate the enhanced processing performance of the turbo controller.*

In 1988 the turbo controller project was conceived as a means of extending the life of the PrintServer 20 platform by introducing a performance-enhanced system controller. The system controller in the PrintServer 20 is housed within and powered by the printer or "print engine"; it is a concise implementation of a single-board computer containing a CPU, a memory subsystem, an Ethernet interface, and a printer interface. It supplies an environment in which a multitasking software system manages communications with remote clients and with the print engine, performs data conversion from the page description language (PostScript) to bit-map images, and provides management of physical print engine resources.

The original controller provided a maximum print speed of 20 pages per minute, but this performance could not be maintained when the document included complex text, graphics, or images. To improve page throughput for complex documents, a controller was needed on which PostScript code could execute faster. To enhance performance, the competition was moving toward controllers based on new industry-standard reduced instruction set computer (RISC) processors. Therefore, to be competitive, Digital's new controller was required to improve performance by five to eight times that of the original controller, which had been based on the rtVAX microprocessor.

As challenging as the performance improvement would be to achieve, budgetary pressures forced restrictions on the implementation strategy. We were to use existing, qualified chips wherever possible in order to avoid new part qualification costs and application-specific integrated circuit (ASIC) development costs.

Early investigations indicated that the performance target was indeed achievable with existing inexpensive RISC processors, as well as a high-speed Digital proprietary VAX processor. A RISC processor would require porting a 2.5-megabyte (MB) software system, which was far beyond the scope of the project. The highest performance VAX processor and the associated support chips, which would not cause a problem with the software system, were far too expensive to be considered. Alternatives were therefore limited to less expensive, lower speed VAX processors: the low-risk, 60-nanosecond (ns) CMOS VAX or CVAX processor was proven, and the higher speed and more cost-effective "system on a chip" or SOC processor was under development. Either choice would have a minimal impact on the software system and would provide a cost-effective solution.

The original performance estimates for the CVAX and the SOC processors in general-purpose processing environments were below the lower bound of the performance target. The design team was also uncertain of the actual execution characteristics of the PrintServer software. For these reasons, it was decided to begin the project with a performance analysis of the original controller to determine the expected performance of a design based on either processor.

This paper discusses the problems encountered during our analysis and the solutions devised by the Hardcopy Systems Engineering Group to overcome them. The RETrACE tool suite, a performance analysis system, is described and the analysis results are provided. The paper then discusses the hardware architecture of the turbo controller and ends with a presentation of the performance results obtained for standard PostScript benchmarks.

## *Performance Analysis of the Original Controller*

The PrintServer 20 software system consists of a VAXELN operating system, an Adobe Systems, Inc. PostScript interpreter, and a substantial amount of software to manage communications and resources. The task of analyzing its performance was complicated by two additional factors. First, the software system's behavior depended on the characteristics of the user's PostScript document. PostScript is an interpreted programming language. Thus, like any computer program, low-level machine performance can be dramatically affected by the program being executed. Second, and more painful, the proprietary nature of the PostScript interpreter prohibited us from obtaining code sources, and discussing its internal architecture with engineers from Adobe Systems.

While the characterization of a complex, partially proprietary, real-time software system is difficult, it is not impossible. Programmer counter address (PC) traces have offered many systems designers very detailed insight into the execution performance and characteristics of systems. PC traces provide a means to observe a system at a macroscopic level, allowing a view of the complex interactions between the hardware and software systems. System designers can use captured address traces from current machine performance to extrapolate expected performance of future systems and help them make architectural trade-offs.

### *The RETrACE Analysis System*

The RETrACE tool suite was created to provide a nonintrusive means of capturing real-time PC traces and analyzing the captured addresses. The tool suite consists of both hardware and software components.

In order to keep expenses at a minimum, existing hardware was used wherever possible. Only one small module had to be developed to complete the RETrACE hardware platform.

The RETrACE hardware consists of the following:

- Two interconnect boards boot and operate a system controller on a table top. Developed as part of the original PrintServer 20, the boards connect the controller to a print engine and an Ethernet.
- The PrintServer 20 server controller was modified for use as an intelligent trace buffer system.
- The PrintServer 20 server controller's memory capacity (12MB) was extended using the standard 4MB memory module used on the Kanji version of the PrintServer 20.
- The RETrACE mother board was developed specifically for this tool suite. It contains a 32-bit wide, first-in, first-out (FIFO) buffer and two loosely coupled state machines.
- A standard PrintServer 20 system controller and print engine were used as the "system under observation."
- The console terminal was selected from the standard VT series of terminals.

A diagram of the RETrACE hardware system is shown in Figure 1.

The RETrACE mother board performed the data capture, using the modified controller's memory as a large buffer. The board monitored the processor bus of the system under observation by copying all addresses and communications between the rtVAX processor and its external floating-point unit. This copied data was placed into a FIFO buffer that in turn was written into the memory of the modified controller using a direct memory access (DMA) device. Since a standard PrintServer 20 controller and its optional memory expansion provide 16MB of storage, approximately 3 seconds of real-time execution address traces could be captured. The data capture continued until the trace buffer memory was exhausted, at which point the data was uploaded over a network connection to a VAX VMS computer for analysis.

Due to the design of the original PrintServer 20 system, many large data areas and code sections were mapped into different explicit memory spaces. This subdivision provided a means of determining which code function was executing in any given segment of the address trace. With a simple statistical study it was possible to generate software execution histograms and to determine many of the characteristics of the system, including translation

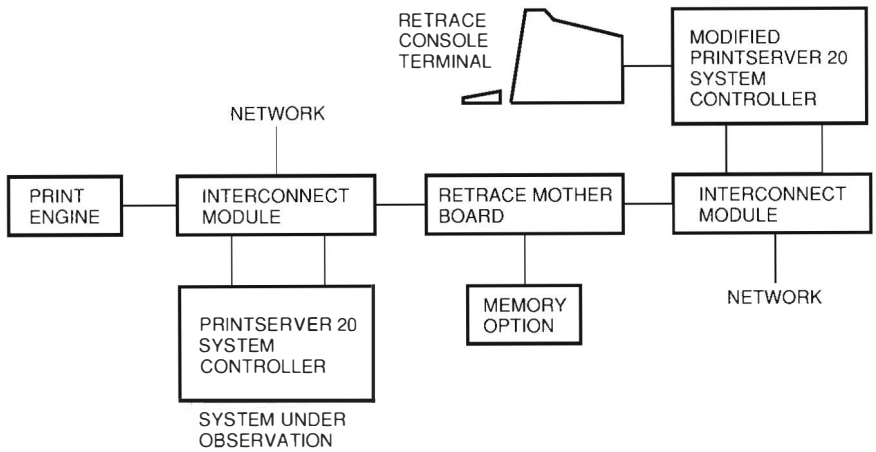



*Figure 1 RETrACE Analysis System Hardware*

buffer, floating point, instruction stream (I-stream) versus data stream (D-stream), read versus write, and interrupt performance. Hit rates for fully associative caches of separate I-stream and D-stream, as well as a combined I- and D-stream cache, were also provided. These hit rates were determined for first-level write-through caches from 128 bytes up to 256 kilobytes (KB). Thus an upper bound for an optimum-performance cached memory system was determined.

Both processors under consideration possessed the ability to access a memory subsystem at speeds greater than that achievable with existing low-cost dynamic random-access memory (DRAM) technology. The performance numbers predicted by the processor groups indicated that cached memory subsystems were required. Because these subsystems can be expensive and their performance is subject to the peculiarities of the software that executes on them, a multilevel memory simulator was developed to allow accurate studies to be performed on proposed cache architectures.

The simulator was configured at run-time to simulate an arbitrary hierarchical memory system that was *N* levels deep, with an arbitrary size, associativity, performance, and behavior at each level. The memory level nearest the processor was defined as the first level, and the last as main memory. The simulator processed a trace file by walking each address in the file through the memory hierarchy starting nearest the processor at the first level. If a copy of the address was found at a given memory level, then a hit was signaled and the next address was processed. If that address was not found, then a miss was signaled and the simulator would proceed to the next level of memory in the hierarchy.

Whenever a hit occurred at a given level, it was logged and all levels of memory in the hierarchy above it would allocate entries based on their defined allocation rules. While this procedure indicated the memory system performance for a proposed architecture, the overall system performance was still unknown. Using a simple rule based on the average execution time per address for the existing controller, and scaling that time based on the clock speed increase of proposed processors, an overall performance number was estimated for a system based on either processor with any arbitrary memory architecture.

## Benchmark Selection

The RETrACE tools suite provided the components required to study the execution characteristics of the PrintServer system without changing the characteristics of its normal operation. The only difficulty was to narrow the focus of the benchmark list to provide a representative sample of PostScript documents to print. Due to time constraints, the list was limited to five benchmarks.

*BM1* The BM1 benchmark stresses those aspects of the system that convert the mathematical representations of characters to bit-map representations, which comprise the form that is printed. This benchmark uses several fonts in standard character orientations, stressing both very large and small character sizes.

*BM2* Of the same type as BM1, this benchmark stresses the transforms from mathematical to bit-mapped character representations; however, the characters printed are at arbitrary orientations with sizes ranging from typical to very small.

*BM3* The BM3 benchmark is one of the standard benchmarks for PostScript performance qualification. It is a simple 41-page document that contains several different fonts. The benchmark is designed to characterize the standard text-handling performance of a printer. This benchmark is printed twice to ensure that all characters to be printed have been converted from mathematical outlines to bit-map representations of the characters. Thus the focus of the benchmark is to move the text data through the system, to copy the character bit maps to the 1MB region in memory that contains the image to be printed, and to print the image. It should be noted that this is the only benchmark that printed at engine speed on the PrintServer 20 system controller powered by the rtVAX system.

*HOUSE* A binary image file, the HOUSE benchmark was used to stress the communications aspects of the PrintServer system.

*SCHEM* The SCHEM benchmark was a vector representation of a logic schematic. This benchmark was used to stress the PostScript interpreter's ability to interpret nonnative PostScript code and to exhibit the characteristics of drawing vectors.

## *Analysis Results*

The thrust of the analysis was to provide credible evidence to support architectural and implementation trade-offs. The major areas of focus were

- Memory system organization
- Printer interface performance
- Main memory bandwidth
- Overall system performance

*Memory System Organization* The statistical analysis of the trace information provided many clues to direct our investigation toward the optimum memory system architecture. The overall read-to-write ratio for the observed benchmarks ranged from as low as 4.3:1 up to 5.5:1, which means for a write-through cache system with a theoretical 100 percent read hit rate, the writes would degrade the overall hit rate to approximately 81 to 84 percent. As the analysis of the data progressed, it was understood that the write data must be studied very closely since it could have a dramatic impact on the overall cache miss rate. During the cache model simulations, the hit rates of the I-stream were between 85 to 90 percent. However, the D-stream hit rates were between 35 to 45 percent, with writes accounting for 60 to 90 percent of the total D-stream misses. To achieve the greatest positive effect on the hit rate of the system, enhancement of write-miss performance was the most advantageous. The two options to improve this performance were either to implement a write-back cache or to add a write buffer to the system. Further cache simulations showed that a write buffer would provide an 8 to 16 percent overall system performance improvement, which was equal to that of a write-back cache. The write buffer, however, was the more straightforward solution to implement.

Cache analysis revealed that the processors required different memory architectures. The CVAX had an internal 1KB, two-way set associative cache. This was to be configured as a mixed I- and D-stream cache. An additional 32KB to 64KB, two-cycle write-through cache was to be added externally. This would also be configured as a mixed I- and D-stream cache. A single-longword, two-cycle write buffer would provide enough buffering to reduce the dramatic impact of write misses. The SOC was proposed to have an internal write-back cache between 5KB and 8KB, with each 1KB region making up a single set. Cache simulations indicated that with a minimum internal mixed I- and D-stream cache of 5KB, five-way set associative, an external data cache would have to be over 64KB to have even a negligible effect on overall system performance. Therefore no external cache was recommended. To mitigate the write-miss penalty, a two-cycle write buffer of 4 to 6 longwords was recommended.

As an acceleration technique, the original PrintServer 20 controller contained a memory access capability that allowed data written to memory to be logically ORed with data that was already stored. This technique was particularly useful when the software system was writing the image that was ultimately printed. As part of the process of generating an image to print, the individual characters appearing on a page must be copied from a region of memory called the font cache to another region called the frame buffer. The frame buffer contains the actual data that is sent to the print engine.

To complicate things, the data written to the frame buffer must be able to overlay data that may already be there, thus requiring a logical OR function.

When a document was printing at or near the maximum engine speed of 20 pages per minute, analysis showed this low-level copying function consumed approximately 20 percent of the total system time allotted to generate and print one page. Thus a logical OR function in the memory system would reduce the number of memory data cycles from "2 reads 1 write" to "1 read 1 write," and reduce the impact from a second read occupying a useful cache location. Without this capability, the degradation would be between 5 and 10 percent of overall system performance when printing at or near 20 pages per minute. Therefore memory capability with a logical OR function was recommended.

*Printer Interface Performance* When a PrintServer 20 is printing, every page that exits the printer requires the 1MB frame buffer to be copied from memory to the print engine interface. Changing a program-controlled printer interface to one driven by a DMA device provided two significant advantages. The first was to reduce the real-time requirements on the PrintServer software system, and the second was to allow for a limited degree of parallelism on the controller. The parallelism was due to the ability of the processor to continue to execute from its cache memory system while the DMA device accessed memory. The processor only stops executing when a cache miss occurs.

*Main Memory Bandwidth* With a CVAX processor configured as recommended in the section Memory System Organization, the main memory system bandwidth requirement of the processor was 60 percent. For the SOC, it was 70 percent when an existing DRAM controller was used. A DMA-driven printer interface required 15 percent, and an Ethernet interface required nominally 4 percent with bursts up to 20 percent. Each subsystem was scrutinized to reduce its required memory bandwidth. The resulting recommendation was to add a 32-bit bus to the memory subsystem to provide a dedicated channel for all data being sent to the printer interface. This provision would reduce required memory bandwidth for the printer interface from 15 percent to about 7 percent. The system would then have a nominal memory bandwidth requirement of 71 percent for a CVAX system and 81 percent for an SOC.

*Overall System Performance* The execution characteristics of the original PrintServer 20 provided some interesting surprises. Most floating-point calculations were performed in double precision; and even more interesting, for each floating-point operation, there was a floating-point conversion from single to double precision, and then back again. Since the precise operations were not required, a simple compiler switch removed the conversions and provided a 3 percent overall system performance improvement for floating-point-intensive PostScript documents. A second surprise came from the results of the BM3 benchmark, which indicated a translation buffer hit rate of 85 percent. At the time of the discovery, the PrintServer 20 was configured with a standard MicroVAX processor; however, by substituting an rtVAX, which uses one less memory access to reference its page tables, an 11 percent system performance improvement was achieved. With this improvement, the rtVAX processor provided enough power to allow the original PrintServer 20 to ship with its 20-page-per-minute designation. This information led the turbo controller designers to determine that the translation buffer of the SOC would be large enough for all the entries required.

## Results

The final analysis revealed that the expected performance of a CVAX or SOC processor would place either design on the low side of the performance requirement. Therefore close attention to detail would be required during the implementation phase of the project as every ounce of performance mattered. The expectation was to have a choice between an SOC processor with a 40-ns cycle time and a CVAX processor with a 60-ns cycle time. The performance improvements of the two processors are compared in Table 1.

**Table 1 Performance Improvement Relative to Original PrintServer 20 Controller**

| Benchmark | SOC Processor | CVAX Processor |
|---|---|---|
| BM1 | 4.7 | 3.7 |
| BM2 | 4.9 | 4.0 |
| BM3 | 4.3 | 3.3 |
| HOUSE | 4.9 | 4.2 |
| SCHEM | 4.7 | 3.7 |

As the project schedule progressed, the risk associated with the new SOC processor decreased. As this risk window collapsed, it was understood that a turbo controller based on the SOC processor would not only perform better, but would also cost less as it would not require an external cache.

## Turbo Controller Hardware Design

The turbo controller was destined for a relatively high-end printer. Therefore the hardware architecture had to provide maximum performance, even though this implementation would increase costs. Based on the results obtained during RETrACE analysis, the hardware design had the following implementation goals:

- The SOC would provide the CPU, the floating-point accelerator (FPA), and the cache subsystem. No second-level cache would be implemented.
- A four- to six-entry write buffer would be implemented.
- The transfer of bit-map data to the print engine would require a 32-bit DMA subsystem with scan-erase capability.
- The memory subsystem would support OR-mode memory access by the CPU and scan-erase access by the DMA controller.

Although both the SOC and rtVAX chips comply with the VAX architecture standard and both are conceptually very similar, they have significant differences in the bus interface. For example, the SOC uses a quadword cycle (one 32-bit address followed by two 32-bit data reads) to fill one internal cache block, while the rtVAX processor, which does not support caching, does not use this type of cycle. Also, the clocking system on the SOC was enhanced, and the timing relationships between signals were modified to improve performance.

The changes to the SOC bus interface, plus the required functional changes revealed by RETrACE analysis, meant that very little of the original PrintServer 20 controller design could be applied to the new controller. One of the first questions to be answered before the design of the turbo controller could begin, was whether or not one or more ASICs would be required for the design. This question had to be answered for three subsystems:

- Main memory
- Write buffer
- Bit-map data transfer subsystem

In each case existing chips satisfied some of the requirements for the subsystem. In the end these chips met all our requirements, but only because they were used in ways not originally intended by the chip designers.

### Main Memory

Since the SOC has a bus interface that is compatible with the CVAX chip, the most obvious chip to use as a main memory controller was the CVAX memory controller (CMCTL) chip.[1] It responds to all bus cycles generated by the SOC, and since it was already used on a number of platforms supported by the VAXELN operating system, its use greatly simplified porting VAXELN to the turbo controller. However, the turbo controller requires two special memory modes that are not provided directly by the CMCTL, namely OR mode and scan-erase mode. It was essential to devise a way to include these two modes if the CMCTL were to be used.

OR-mode memory is a technique used to improve performance during the writing of the page bit map into memory (scan conversion). During normal memory operation (called replace mode), the destination operand in memory is replaced by the source operand. During an OR-mode write cycle, the destination operand is modified as follows:

- For each logical zero in the source data being written, the corresponding destination bit in memory remains unchanged.
- For each logical one in the source data being written, the corresponding destination bit in memory is written with the corresponding bit in the pattern register.
- The pattern register is a 32-bit register which determines the "color" pattern of the "ink" being written on the page.

Figure 2 shows a portion of the logic between the CMCTL and the memory array that implements the OR-mode function in hardware. The OR-mode operation is accomplished by inverting the source data and connecting it to 32 independent write enables of the memory array. When a zero is written, it is inverted and the write cycle for that bit becomes a read cycle, thus preventing any change to the memory contents. When a one is written, it is inverted and the write is allowed to occur, but the data actually written depends on the value previously written into the pattern register.

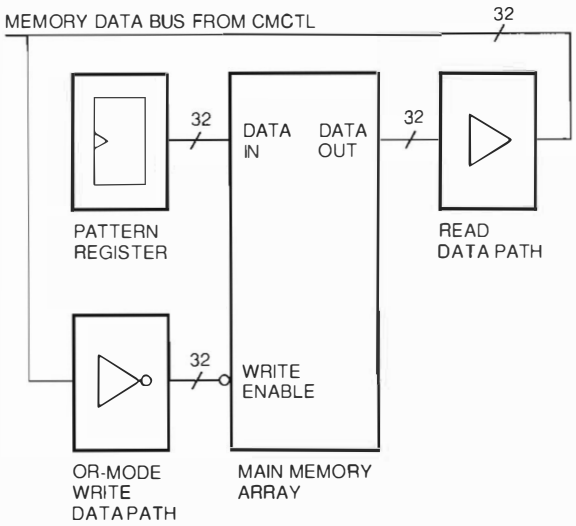



*Figure 2 OR-mode Circuit*

Two features of the CMCTL chip make it possible to implement OR-mode memory. First, its 64MB address space is divided into 4 arrays of 4 banks (16 banks total). Second, the CMCTL chip can selectively disable parity checking on an array.

The large address space of the CMCTL allows the use of 2 arrays for replace mode and 2 arrays for OR mode, since the turbo controller supports up to 32MB of memory. The control signals of the two sets of arrays are combined such that OR mode and replace mode access the same physical memory, though in different ways. Parity error detection is disabled on the OR-mode arrays; thus a read-through OR-mode address space cannot cause a parity error. This is necessary because OR-mode write cycles may corrupt parity. Normally any bit map created using OR-mode write cycles is read using OR-mode read cycles.

The other special mode required for the main memory system is called scan-erase mode. It is an operating mode designed to improve bus utilization during the transfer of the bit map from main memory to a FIFO buffer connected to the printer data lines. This mode is made possible by a side effect of the error-correcting code (ECC)/parity generation logic in the CMCTL. Any time a masked write occurs (any write other than an aligned longword, such as a byte write), the destination longword must first be read by the CMCTL, then combined with the bytes to be written in order to generate the parity or ECC check bits for that longword.

Three operations occur during a single scan-erase cycle. Refer to the circuit drawing in Figure 3.

1. The bus master asserts the signal's "bit-map load" and "bit-map erase" and requests a masked write. The CMCTL performs a read, and the bit map is read onto the memory data bus.
2. Bit-map data is automatically transferred from the memory data bus into the FIFO buffer.
3. The CMCTL performs a write. However, since the bit-map erase signal has disabled the data path and the pull-down resistors have set the data-in lines to all zeros, the write cycle, which was intended by the designers of the chip as a masked write, has in fact become a memory clear operation.

## *Write Buffer*

The LR3220 chip was chosen as the base for the write buffer subsystem. It provides a six-entry FIFO buffer for address, data, and byte mask and detects whether the processor has requested a read at a memory location for which a write is still pending. It also supports two operating modes: LR3000 mode and Harvard mode.

If it were not for the Harvard-mode feature, it would have been more difficult to include the LR3220 chip into the turbo controller. The LR3000 processor, for which this chip was designed, has staggered address timing. Some of the address and byte-mask bits are asserted on the falling edge of the clock, and the remaining bits are asserted on the rising edge of the clock. When the LR3220 chip is configured in LR3000 mode, the processor subsystem must meet these timing requirements. However, when the LR3220 chip is configured in Harvard mode, all address, data, and byte-mask information is read at the same rising clock edge.

The basic strategy for including the write buffer into the turbo controller was to insert the write buffer between the SOC and the rest of the system as shown in Figure 4. The SOC would issue read and write requests to the write buffer, and the write buffer would issue read and write requests to the rest of the system. During CPU cycles the SOC and the write buffer have a master-slave relationship in which the SOC is the master. The relationship between the write buffer and the rest of the system is also a master-slave relationship; however, the write buffer is the master. In fact, the write-buffer output interface must look almost identical to the SOC.

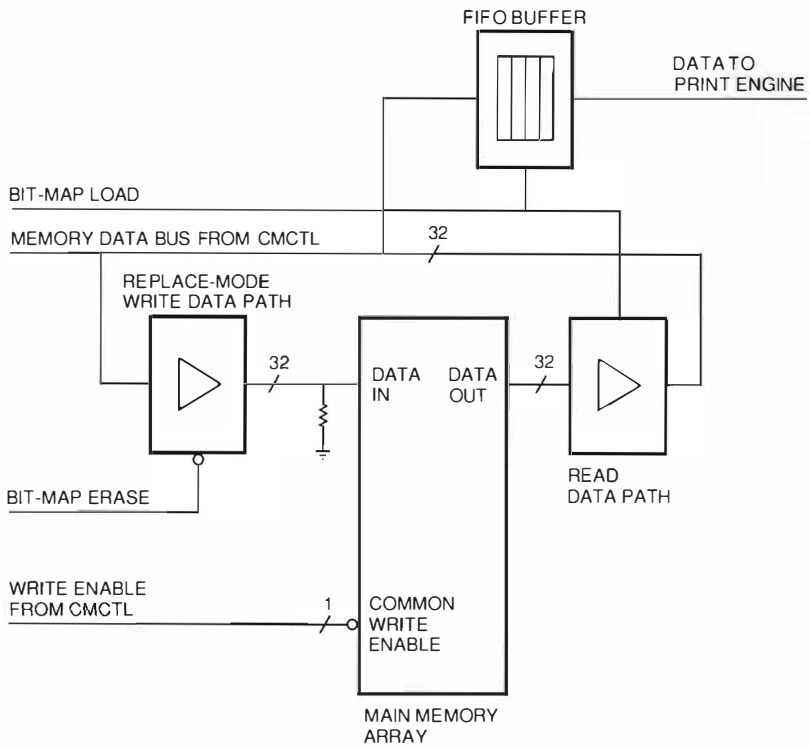



*Figure 3 Scan-erase Circuit*

The structure of the write-buffer subsystem is shown in Figure 5. The bus interface unit responds to read or write requests from the SOC. During write cycles, the bus interface writes the data into the LR3220 chip and immediately alerts the SOC to terminate the cycle quickly. Whenever one or more entries in the LR3220 chip have data, the bus cycle generator (BCG) removes the next entry and issues a write request to the appropriate subsystem.

The write-buffer subsystem allows the SOC to "read around" the write buffer, provided the address being read does not have a pending write in the

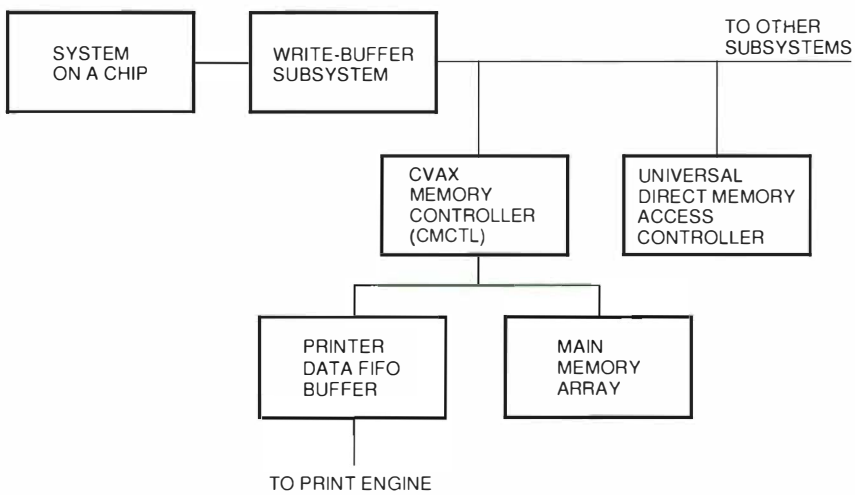



*Figure 4 Interconnection of Turbo Controller Subsystems*

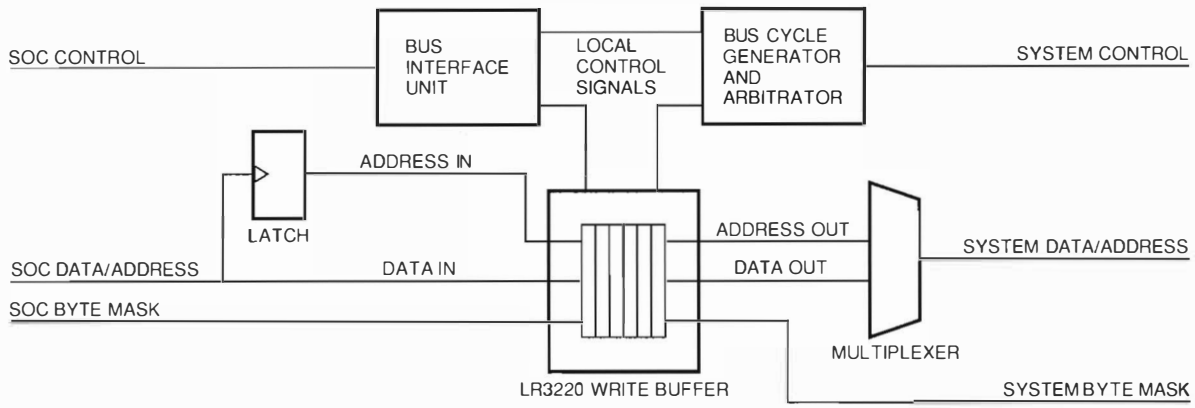



*Figure 5 Write Buffer*

LR3220. To handle this, the BCG includes an arbitration circuit. When the SOC requests a read cycle, the bus interface unit of the write buffer passes the request to the BCG. The BCG responds once it has completed any write cycle currently in progress, provided that the address to be read does not have a pending write in the write buffer. When the slave device being read acknowledges the BCG, the acknowledgment is passed back to the bus interface and finally to the SOC to terminate the cycle. The BCG then resumes its task of removing entries from the LR3220 chip and issuing writes to the rest of the system.

In order to maintain data coherency, the write-buffer subsystem enforces some additional protocols.

- All writes to any location other than main memory require a write-flush cycle; that is, the bus interface must wait until the LR3220 chip is empty before writing the data to it. Furthermore, the bus interface must wait until the BCG has finished the cycle before it acknowledges the SOC and allows it to perform the next cycle.
- All reads to any location other than main memory require a read flush, which has the same restrictions as a write flush. These restrictions are required to avoid the possibility of reading around a pending I/O space write, which often has side effects to other addresses.
- The write-buffer subsystem must pass all DMA bus transactions to the SOC to ensure that all cached memory locations that are modified by DMA cycles have their corresponding cache entry invalidated.

## *Bit-map Transfer Subsystem*

The bit-map transfer subsystem transfers bit-map data, created by the PostScript interpreter, to the print engine. It is composed of the 32-bit DMA controller, a FIFO subsystem, and scan-erase logic in main memory as described in the section Main Memory.

The main requirements for the 32-bit DMA controller were

- 32MB address range
- Ability to transfer 32 bits at a time
- Ability to transfer the frame buffer forward (incrementing the source address) or backward (decrementing the source address)

None of the available DMA controller chips met all our requirements, but the AMD 9516 universal DMA controller (UDC) met some of them. The UDC is a 16-bit DMA controller with a 16MB address range and the ability to increment or decrement the source address. There were two drawbacks to the use of this chip. The software would have to ensure that the frame buffer was always within the lower 16MB of memory, and the UDC would use twice as much bus bandwidth since it could transfer only 16 bits at a time.

It was proposed that the UDC could be used as a full 32-bit DMA controller if it was connected to the bus "incorrectly" by shifting the data/address lines to the left by one bit. That is, data/address line 0 on the UDC would be connected to data/address line 1 on the bus; data/address line 1 of the UDC would be connected to data/address line 2 on the bus; etc. This type of connection doubles the address range of the chip and causes the source address on the

bus to increment by 4 bytes (32 bits) instead of 2 bytes (16 bits).

This decision had a few implementation impacts. For example, the register definitions were now incorrect, since all the bits in all the registers were shifted one bit to the left. However, once the software was modified to compensate for this, the UDC functioned properly as a 32-bit DMA controller. When combined with the scan-erase feature of main memory, it allowed us to achieve our bit-map transfer goal of reading 32 bits from memory, loading it into the FIFO subsystem, and clearing the memory location, all in a single DMA cycle.

## Performance

In this section, the performance of the original PrintServer 20 is compared to the enhanced performance of the turbo PrintServer 20.

Except for performance, the original PrintServer 20 and the turbo PrintServer 20 have identical functional capabilities. Table 2 lists the five functional subsets that were characterized for performance on both printers. The first four functional subsets were rated using the PostScript real-time operator; they measure the elapsed CPU time needed to complete a test. The last functional subset was rated according to the rate of pages exiting the printer. The term "DECnet/DPS" refers to the DECnet job (a job is one of several multiprocessing tasks running on the controller) and the "distributed PrintServer software" job. The term "printer system" refers to the complete printer system, including the PostScript job and the printing overhead jobs. The printer system was rated according to the rate of pages exiting the printer.

Table 3 reports the general attributes of the five files that were run with the RETrACE system and characterized for performance.

**Table 2 Functional Subsets of the Printers**

| Functional Subsets | Characterization |
|---|---|
| PostScript job | Math operations per second |
| PostScript job | Text: characters per second |
| PostScript job | Graphics: vector inches per second |
| DECnet/DPS jobs | DECnet/DPS: kilobytes per second |
| Printer system excluding DECnet/DPS | Image printing: square inches per second |

**Table 3 Benchmark File Attributes**

| File Name | General Attributes of File |
|---|---|
| BM1.PS | Contains 39 pages of text with 13 fonts of various sizes. Some text strings are at varying angles. |
| BM2.PS | Contains 1 page of spiral text of various point sizes. |
| BM3.PS | Contains 41 pages of text with 5 fonts. |
| HOUSE.PS | Contains a 1-page bitonal image of 3000 blocks (DECnet limited). |
| SCHEM.PS | Contains a 65-page schematic of graphics (vectors) and text. |

### *Math Operators Performance of the PostScript Job*

Figure 6 illustrates the controllers' performance results in math operations per second. The test determines the time needed to perform 50,000 primitive math operators (e.g., adding two numbers 50,000 times) during a PostScript test document. The real-time operator reads the current time, and the repeat construct repeats the math operator. This test measures the performance of the CPU only.

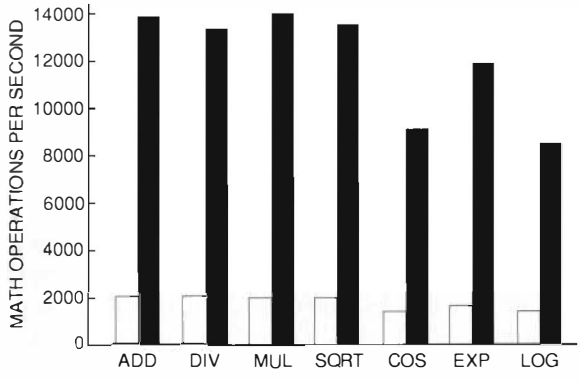



*Figure 6 PostScript Job Performance with Math*

### *Text Performance of the PostScript Job*

Figure 7 compares the text performance of the PostScript job on the original controller and the turbo controller. The test determines how long it takes the PostScript job to compose 250,000 equally

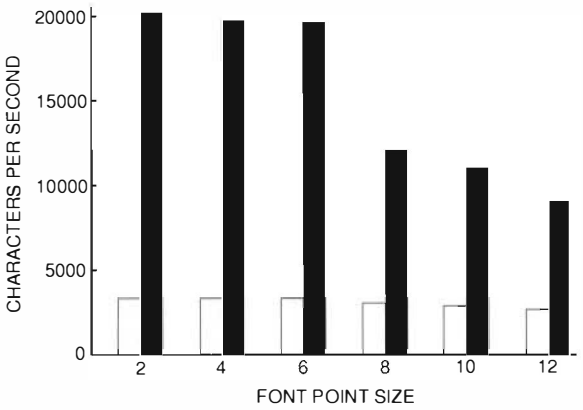



*Figure 7 PostScript Job Performance with Text*

sized characters to the page buffer in memory, which eventually is sent to the print engine to be printed.

## *Graphics Performance of PostScript Job*

An important means of characterizing graphics performance is in vector inches per second. Figure 8 shows the results obtained by running a PostScript vector program in which all vectors are at 45 degrees and vector lengths are from 0.1 inch to 3 inches.

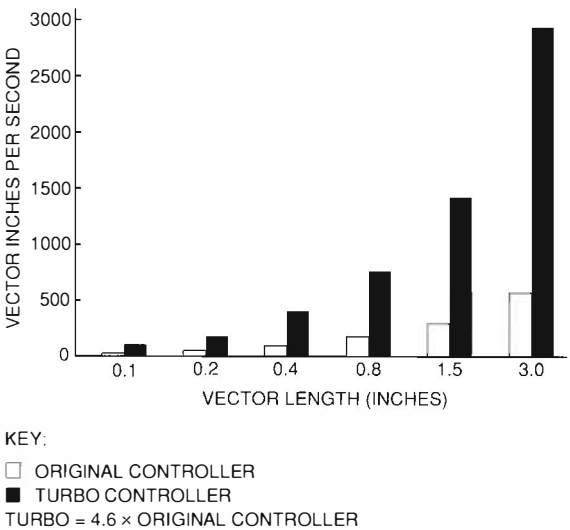



*Figure 8 PostScript Job Performance with Graphics*

## *Image Performance*

The image test characterized the complete printer system, including the PostScript job and the printing overhead jobs, but excluding the DECnet/DPS time required to transfer an image file to a printer. Three one-square-inch bitonal images at device resolution were placed into the user dictionary and were used repeatedly during the performance measurement. The result of using these precached images was to eliminate the DECnet and DPS software time that would be required to transfer a full-page image from a host to the printer. Performance was measured by printing 10 pages of 80 square inches of image per page.

The pages were printed landscape and portrait to measure the image performance both on axis and off axis. (On axis means that the printer sequentially prints all bits of a word from the image on a single scan line. Off axis by 90 degrees means that the printer prints one bit from each word and does not print the next bit in the word until it is at the same position on the next scan line.) Figure 9 shows the results of the image performance test in square inches per second.

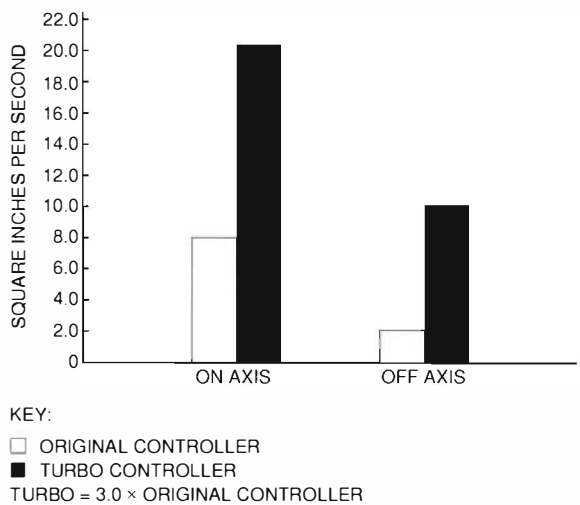



*Figure 9 Image Performance Measurement of the Printing System*

## *DECnet/DPS Jobs Performance*

DECnet/DPS transfer rates can be ignored for text and graphics files, but these rates can consume most of the time needed to print large image files. For example, a single, letter-size page of image contains more than 1MB of image data, but the

corresponding PostScript file contains more than 2MB. Because the image data is represented in American standard code for information interchange (ASCII) hexadecimal characters in PostScript, 8 bits of the PostScript file are needed to represent 4 bits of image data.

To measure DECnet/DPS, a PostScript file of 1 MB of comments was sent to the printer. The clock was started when the beginning of the file was received by the PostScript interpreter and stopped when the end of the file was received. The assumption of this test method was that the PostScript interpreter can parse comment lines much faster than DECnet/DPS can transfer them.

The DECnet/DPS transfer rate is basically proportional to the slower of the host and printer processors. Figure 10 shows the DECnet/DPS results.

### *RETrACE Benchmark Files*

The benchmark files listed in Table 4 are characterized both by the elapsed time from file arrival to file printed and by the amount of CPU time used to print the job. For example, in the BM3 benchmark, the speed is limited by the 20-page-per-minute print engine, but the CPU time needed to print the file can be used as a performance measurement.

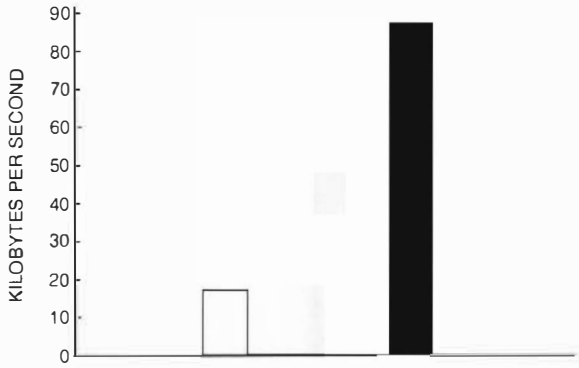



*Figure 10 DECnet/DPS Jobs Performance*

## *Summary*

The turbo controller enhanced the performance of the PrintServer 20 printer system. Its design was prompted by the need to maintain print speed performance for complex documents containing text, graphics, and images. The RETrACE system was designed to analyze the PrintServer 20 system to determine which architectural changes would provide the greatest improvement in PostScript performance. By optimizing hardware only in areas where it was truly worthwhile, we were able to use existing chips and reduce development costs. The subsystems of the turbo controller hardware that were optimized as a result of this analysis were the processor (SOC which provided CPU, floating-point accelerator, and cache subsystem), a memory subsystem with OR-mode and scan-erase access, a write-buffer subsystem, and a 32-bit DMA subsystem. Results of the performance tests for five benchmarks, including PostScript jobs, indicate the levels of enhanced performance.

## *Acknowledgments*

Chris Mayer designed and implemented the RETrACE multilevel cache simulator. He developed a ticketing algorithm that simplified the management of delayed behavior memory constructs such as write buffers.

## *Reference*

1. D. K. Morgan, "The CVAX CMCTL—A CMOS Memory Controller Chip," *Digital Technical Journal*, vol. 1, no. 7 (August 1988): 139–143.

**Table 4 Benchmark Files Characterized by Elapsed Time and CPU Time (Seconds)**

| Benchmark File | Original CPU | Turbo CPU | Original Elapsed | Turbo Elapsed | delta CPU | delta Elapsed |
|---|---|---|---|---|---|---|
| BM1 | 7585 | 1707 | 7735 | 2050 | 4.4 | 3.8 |
| BM2 | 238 | 51 | 241 | 51 | 4.7 | 4.7 |
| BM3 | 56 | 15 | 128 | 120 | 3.7 | 1.1* |
| HOUSE | 67 | 15 | 106 | 31 | 4.5 | 3.4 |
| SCHEM | 2802 | 625 | 3073 | 675 | 4.5 | 4.6 |

*Limited by engine.

# Further Readings

*The* Digital Technical Journal *publishes papers that explore the technological foundations of Digital's major products. Each* Journal *focuses on at least one product area and presents a compilation of papers written by the engineers who developed the product. The content for the* Journal *is selected by the Journal Advisory Board. Digital engineers who would like to contribute a paper to the* Journal *should contact the editor at RDVAX::BLAKE.*

Topics covered in previous issues of the *Digital Technical Journal* are as follows:

**Availability in VAXcluster Systems/ Network Performance and Adapters**
*Vol. 3, No. 3, Summer 1991*
Discussions of VMS volume shadowing, VAXcluster application design, and new availability features of local area VAXcluster systems, together with details of high-performance Ethernet and FDDI adapters, and an analysis of FDDI LAN performance

**Fiber Distributed Data Interface**
*Vol. 3, No. 2, Spring 1991*
The FDDI LAN system and Digital's products that support this technology, with an overview and papers on the physical and data link layers, Common Node Software, bridge and concentrator devices and related management software, and an ULTRIX network adapter

**Transaction Processing, Databases, and Fault-tolerant Systems**
*Vol. 3, No. 1, Winter 1991*
The architecture and products of Digital's distributed transaction processing systems, with information on monitors, performance measurement, system sizing, database availability, commit processing, and fault tolerance

**VAX 9000 Series**
*Vol. 2, No. 4, Fall 1990*
The technologies and processes used to build Digital's first mainframe computer, including papers on the architecture, microarchitecture, chip set, vector processor, and power system, as well as CAD and test methodologies

**DECwindows Program**
*Vol. 2, No. 3, Summer 1990*
An overview and descriptions of the enhancements Digital's engineers have made to MIT's X Window System in such areas as the server, toolkit, interface language, and graphics, as well as contributions made to related industry standards

**VAX 6000 Model 400 System**
*Vol. 2, No. 2, Spring 1990*
The highly expandable and configurable midrange family of VAX systems that includes a vector processor, a high-performance scalar processor, and advances in chip design and physical technology

**Compound Document Architecture**
*Vol. 2, No. 1, Winter 1990*
The CDA family of architectures and services that support the creation, interchange, and processing of compound documents in a heterogeneous network environment

**Distributed Systems**
*Vol. 1, No. 9, June 1989*
Products that allow system resource sharing throughout a network, the methods and tools to evaluate product and system performance

**Storage Technology**
*Vol. 1, No. 8, February 1989*
Engineering technologies used in the design, manufacture, and maintenance of Digital's storage and information management products

**CVAX-based Systems**
*Vol. 1, No. 7, August 1988*
CVAX chip set design and multiprocessing architecture of the midrange VAX 6200 family of systems and the MicroVAX 3500/3600 systems

**Software Productivity Tools**
*Vol. 1, No. 6, February 1988*
Tools that assist programmers in the development of high-quality, reliable software

**VAXcluster Systems**
*Vol. 1, No. 5, September 1987*
System communication architecture, design and implementation of a distributed lock manager, and performance measurements

**VAX 8800 Family**
*Vol. 1, No. 4, February 1987*
The microarchitecture, internal boxes, VAXBI bus, and VMS support for the VAX 8800 high-end multiprocessor, simulation, and CAD methodology

**Networking Products**
*Vol. 1, No. 3, September 1986*
The Digital Network Architecture (DNA), network performance, LANbridge 100, DECnet-ULTRIX and DECnet-DOS, monitor design

**MicroVAX II System**
*Vol. 1, No. 2, March 1986*
The implementation of the microprocessor and floating point chips, CAD suite, MicroVAX workstation, disk controllers, and TK50 tape drive

**VAX 8600 Processor**
*Vol. 1, No. 1, August 1985*
The system design with pipelined architecture, the I-box, F-box, packaging considerations, signal integrity, and design for reliability

Subscriptions to the *Digital Technical Journal* are available on a yearly, prepaid basis. The subscription rate is $40.00 per year (four issues). Requests should be sent to Cathy Phillips, Digital Equipment Corporation, MLO1-3/B68, 146 Main Street, Maynard, MA 01754, U.S.A. Subscriptions must be paid in U.S. dollars, and checks should be made payable to Digital Equipment Corporation.

Single copies and past issues of the *Digital Technical Journal* can be ordered from Digital Press at a cost of $16.00 per copy.

### Technical Papers by Digital Authors

R. Al-Jarr, "A Methodology for Evaluating Decision Making Architectures for Automated Manufacturing Systems," *Eleventh IFAC Conference* (August 1990).

S. Angebranndt, R. Hyde, D Luong, and N. Siravara, "Integrating Audio and Telephony in a Distributed Workstation Environment," *Proceedings of the Summer 1991 USENIX Conference* (June 1991).

S. Batra, M. Mallary, and A. Torabi, "Frequency Response of Thin-film Heads with Longitudinal and Transverse Anisotropy," *IEEE Intermag '90* (April 1990).

R. Csencsits, N. Riel, J. Dion and S. Arsenault, "Interfacial Structure and Adhesion of Metal-on-polyamide," *International Symposium for Testing and Failure Analysis* (October 1990).

R. Csencsits, J. Rose, R. St. Amand, L. Elliott, A. Hartzell, L. Kisselgof, and J. Lloyd, "Aluminum Interconnect Microstructure and Its Role in Electromigration," *International Symposium for Testing and Failure Analysis* (October 1990).

J. Delahunty and T. Kielty, "Automated Pareto Analysis for Continuously Improving a VLSI Fabrication Area's Process Stability," *Advanced Semiconductor Manufacturing Conference* (September 1990).

S. Dell, "Promoting Equality of the Sexes through Technical Writing," *Society for Technical Communication* (August 1990).

B. Doyle and K. Mistry, "A Lifetime Prediction Method for Hot-carrier Degradation in Surface-channel P-MOS Devices," *IEEE Transactions on Electron Devices* (May 1990).

E. Freedman and Z. Cvetanovic, "Efficient Decomposition and Performance of Parallel PDE, FFT, Monte Carlo Simulations Simplex and Sparse Solvers," *IEEE Supercomputing '90* (November 1990).

A. Gardel and P. Deosthali, "Hub-centered Production Control of Wafer Fabrication," *Advanced Semiconductor Manufacturing Conference* (September 1990).

A. Hartzell, "Introduction of Argon as a Heat Transfer Gas in a Single Wafer RIE System," *International Symposium for Testing and Failure Analysis* (October 1990).

A. Heyman and J. Thottuvelil, "Linear Averaged and Sampled Data Models for Large Signal Control of High Power Factor AC-DC Converters," *IEEE Power Electronics Specialists* (June 1990).

L. Hill, "Video Signal Analysis for EMI Control," *IEEE Electromag '91* (1991).

L. Hill and A. Metsler, "Video Subsystem Design for EMI Control," *IEEE Electromag '91* (1991).

S. Kasturi, "Forced Convection: The Key to the Versatile Reflow Process," *NEPCON East '90* (June 1990).

D. Mirchandani and P. Biswas, "Characterization and Modeling Ethernet Performance of Distributed DECwindows Applications," *ACM Sigmetrics* (May 1990).

W. Metz, "Automated On-line Optimization of an Epitaxial Process," *International Semiconductor Manufacturing Science Symposium* (May 1990).

K. Mistry, B. Doyle, and D. Krakauer, "Impact of Snapback Induced Hole Injection on Gate Oxide Reliability in N-MOSFET's," *IEEE Electron Device Letters* (October 1990).

C. Pan, "Gas Lubrication," *ASME/STLE Tribology Conference* (October 1990).

A. Philipossian, "Fluid Dynamics Analysis of Thermal Oxidation Systems via Residence Time Distribution (RDT)," *Electromechanical Society* (October 1990).

M. Sidman, "Convergence Properties of an Adaptive Runout Correction System," *ASME Winter Meeting* (November 1990).

M. Sidman, "Parametic System Identification on Logarithmic Frequency Response Data," *IEEE Transactions on Automatic Control* (September 1991).

D. Skendzic, "Two Transistor Flyback Converter Design for EMI Control," *IEEE Symposium on Electromagnetic Compatibility* (August 1990).

A. Smith and W. Goller, "New Domain Configuration in Thin-film Heads," *Intermag '90* (April 1990).

H. Smith and J. Beagle, "SIMS for Accurate Process Monitoring in CoSi2-on-Si MOSFET Technology," *Secondary Ion Mass Spectrometry* (September 1989).

J. Thottuvelil, "Using SPICE to Model the Dynamic Behavior of DC-to-DC Converters Employing Magnetic Amplifiers," *IEEE Applied Power Electronics Conference* (March 1990).

R. Ulichney, "Frequency Analysis of Ordered Dither," *Hard-copy Output OE/LASE 89 SPIE '89 Proceedings* (1991).

R. Ulichney, "Challenges in Device Independent Image Rendering," *Applied Vision Optical Society of America Technical Digest Series '89* (1991).

E. Zimran, "Performance Efficient Mapping of Applications to Parallel and Distributed Architectures," *International Conference on Parallel Processing* (August 1990).

### Digital Press

Digital Press is the book publishing group of Digital Equipment Corporation. The Press is an international publisher of computer books and journals on new technologies and products for users, system and network managers, programmers, and other professionals. Proposals and ideas for books in these and related areas are welcomed.

The following book descriptions represent a sample of the books available from Digital Press.

**VAX/VMS: Writing Real Programs in DCL**
Paul C. Anagnostopoulos, 1989, softbound, 409 pages, Order No. EY-C168E-DP-EEB ($29.95)

This book contains information that can help the reader learn to write powerful and well-organized programs in DCL, the command language for the VAX/VMS operating system. The text includes a review of the syntax and semantics of DCL and a discussion of significant issues in the development of serious DCL software. Programming paradigms are presented, as well as the correct way to implement them. The book presents good programming techniques and helps the student to make effective use of the VMS operating system.

**X WINDOW SYSTEM TOOLKIT: The Complete Programmer's Guide and Specification**
Paul J. Asente and Ralph R. Swick, 1990, softbound, 1000 pages, Order No. EY-E757E-DP-EEB ($44.95)

This book consists of two parts, "Programmer's Guide" and "Specification." "Programmer's Guide" describes how to use the X Toolkit to write applications and widgets, and includes many examples. Each chapter in this part contains an application writer's section and a widget writer's section. Application programmers need to read the widget writer's sections only if they are curious about what is going on behind the scenes; widget programmers should read both sections. "Specification" provides a complete and concise description of every component of the X Toolkit Intrinsics, as standardized by the MIT X Consortium. The level of detail in this part is sufficient to enable a programmer to create a new implementation of the X Toolkit.

**PRODUCTION SOFTWARE THAT WORKS: A Guide to the Concurrent Development of Realtime Manufacturing Systems**
John A. Behuniak, Iftikhar Ahmad, and Ann M. Courtright, 1992, softbound, 204 pages, Order No. EY-H895E-DP-EEB ($24.95)

This is a practical guidebook for manufacturing managers and process engineers who must develop better process methodologies to stay competitive and for developers of realtime manufacturing software who need to cut time and costs from their work. The presentation, which provides useful advice and easy-to-follow procedures, addresses three basic tasks of realtime software development

in a manufacturing plant: (1) managing the design of the system; (2) setting up and managing a development organization; and (3) implementing tools for successful completion and management.

**UNIX FOR VMS USERS**
Philip E. Bourne, 1990, softbound, 368 pages, Order No. EY-C177E-DP-EEB ($28.95)

This book emphasizes the practical aspects of making the transition from the VMS to the UNIX operating system. Every concept presented is illustrated with one or more examples, comparing how to perform a particular task in each of the two operating systems. The book is organized in a logical order and covers the following topics: fundamental concepts to be grasped before touching the keyboard, the first terminal sessions, the first commands, editing, communicating with users, resource utilization, using devices, more advanced commands, using high-level languages, programming the operating system, text processing, and networking. Appendixes provide extensive cross-reference tables to make this a valuable reference tool for even the experienced UNIX user.

**LOGISTICAL EXCELLENCE: It's Not Business as Usual**
Donald J. Bowersox, Patricia J. Daugherty, Cornelia L. Drogue, Richard N. Germain, and Dale S. Rodgers, 1992, 300 pages, Order No. EY-H953E-DP-EEB

This book focuses on the interpretation of research findings that have been compiled to help managers who seek to improve logistical competency within their organization. It provides a sequential model, the best practices of "excellent" logistics managers with supportive statistical evidence, and extensive coverage of Electronic Data Interchange in the logistics process. It also includes a brief overview of the expanding role that logistics has recently played in the overall corporate strategy of increasing speed and quality. To facilitate interest and ease of reading, an action-oriented case dialogue runs throughout the eight chapters.

**WRITING VAX/VMS APPLICATIONS USING PASCAL**
Theo de Klerk, 1991, hardbound, 748 pages, Order No. EY-F592E-DP-EEB ($39.95)

Written for the professional application programmer on the VAX/VMS operating system using the VAX Pascal programming language, this is the first book to actually discuss the construction of real VMS applications. It sets forth a methodology for producing high-quality, professional VMS applications by focusing on the aspects of the VMS operating system crucial to every well-written application.

**THE DIGITAL GUIDE TO SOFTWARE DEVELOPMENT**
The staff of the Corporate User Information Products (CUIP/ASG), Digital Equipment Corporation, 1989, softbound, 239 pages, Order No. EY-C178E-DP-EEB ($27.95)

*THE DIGITAL GUIDE TO SOFTWARE DEVELOPMENT* is the first published description of the methodology that Digital uses to design and develop its software. For the engineer and other professionals associated with the creation and marketing of software applications, this book gives a rare look at the practices of an industry leader and provides a model for others who wish to introduce software engineering methods and tools into their own companies. Also discussed are the use of selected VMS case tools to expedite the process; the roles of teams and team leaders; the use of review meetings and documents; and formal procedures for testing and maintenance. The guide includes numerous diagrams and tables, clear guidelines for the coding and documentation of software modules, a listing of related VMS documentation, and coding guidelines for VAX C.

**DIGITAL GUIDE TO DEVELOPING INTERNATIONAL SOFTWARE**
The staff of the Corporate User Information Products (CUIP/ASG), Digital Equipment Corporation, 1991, softbound, 381 pages, Order No. EY-F577E-DP-EEB ($28.95)

This book introduces the ground-breaking packaging and design guidelines recommended by Digital for products destined for overseas markets. Already used by more than 400 independent software vendors and development groups, as well as by Digital engineers, this book offers an approach that greatly simplifies the steps required to adapt software to local markets once the parent product has been released. The book features a description of Digital's international product model, a scheme for separating the core functions of a product from those that require translation or modification for

specific markets. Also included are guidelines for developers working in DECwindows, VMS, and ULTRIX environments; special considerations involved in preparing a product for multibyte Asian languages or for multilanguage environments; and appendixes with information on the systems issues in computer architecture.

**USING MS-DOS KERMIT: Connecting your PC to the Electronic World, Second Edition**
Christine M. Gianone, 1991, softbound, 344 pages with software disk included, Order No. EY-H893E-DP-EEB ($34.95)

As in the first edition, this software package leads the novice step by step through installation, communication setup, terminal emulation, file transfer, and script programming, and also serves as a complete reference work for the experienced user. Complete with 5¼-inch diskette containing the official MS-DOS KERMIT Version 3.11 program from Columbia University, this revision includes a new section on local area networks, additional material on running Kermit in windowed environments such as Microsoft Windows and Quarterdeck DesqView, a new appendix containing tables of the escape sequences used by Kermit's text and graphics terminal emulators, and expanded descriptions of many of Kermit's features.

**ENTERPRISE NETWORKING: Working Together Apart**
Raymond H. Grenier and George S. Metes, 1991, hardbound, 260 pages, Order No. EY-H878E-DP-EEB ($29.95)

To successfully compete in the next century, companies must recognize and adapt to exponential changes, including the dispersion of markets and resources and acceleration in market demands. *ENTERPRISE NETWORKING: Working Together Apart,* describes how management can support this distributed electronic information environment and move through planned transitions to a new organization, confident they will prosper. Intended for individuals in charge of directing transition of information-focused groups that extend across geographies, this book is segmented into four parts. The Introduction, Part I, defines the assumptions and realities. Part II focuses on Capability Based Environments. Part III discusses Simultaneous Distributed Work, both Goals and Processes, and Continuous Design and Quest for Quality. The Epilogue, Part IV, concludes with three appendices detailing Benchmarking, Building Networks, and Networking Capabilities.

**THE ART OF TECHNICAL DOCUMENTATION**
Katherine Haramundanis, 1992, softbound, 267 pages, Order No. EY-H892E-DP-EEB ($28.95)

Written primarily for novice and aspiring technical writers within the computer industry, *The Art of Technical Documentation* has unique features, including its advice on planning and process, research techniques, use of graphics, audience analysis, definition of quality, standards, and careers that are valuable to experienced technical writers as well. Haramundanis views the practice of technical writing as being different from that of scientific writing, and closer to investigative reporting. In keeping with this premise, this book is not a style guide that deals with all aspects of typography and copy editing, but instead presents the distilled knowledge of the author's many years experience.

**A COMPREHENSIVE GUIDE TO Rdb/VMS**
Lilian Hobbs and Kenneth England, 1991, softbound, 352 pages, Order No. EY-H873E-DP-EEB ($34.95)

The Rdb/VMS relational database system was developed by Digital Equipment Corporation for VAX computers using the VMS operating system. This system is one of a number of information management products that work together to facilitate the sharing of information. The Rdb/VMS system is used, for example, in high-performance transaction processing systems. This book is based on Rdb/VMS Version 4.0, which Digital made available to customers at the end of 1990, and thus includes the latest functionality.

**DIGITAL GUIDE TO DEVELOPING INTERNATIONAL USER INFORMATION**
Scott Jones, Cynthia Kennelly, Claudia Mueller, Marcia Sweezey, Bill Thomas, and Lydia Velez, 1992, softbound, 214 pages, Order No. EY-H894E-DP-EEB ($24.95)

Designed for the busy professional, this book presents models that extend beyond Digital and English speaking countries in a quick read/reference format. Nine chapters and four appendices outline methods for creating written, visual,

and verbal information for cost-effective translation. Primarily for information specialists, including writers, editors, illustrators, course developers, and their managers, this book will also help software developers and students enhance their background in technical communication.

**PRACTICAL KNOWLEDGE ENGINEERING: Creating Successful Commercial Expert Systems**
Richard V. Kelly, Jr., softbound, 212 pages, Order No. EY-F591E-DP-EEB ($28.95)

This book is a concise guide to practical methods for initiating, designing, building, managing, and demonstrating commercial expert systems. It is a front-line report of what works (and what does not) in the construction of expert systems, drawn from the author's decade of experience gained working on such projects in all major areas of application for American, European, and Japanese organizations. It also briefly reviews the knowledge representation, programming, and management techniques commonly used to implement expert systems today, and describes the intellectual, organizational, financial, and managerial issues that knowledge engineers face daily in performing their jobs. Among the topics covered are: prospecting for "legitimate" problems; forecasting costs, establishing project metrics and writing specifications; preparing for system "demos"; interviewing and selecting engineering team members; and solving common difficulties in design and implementation.

**COMPUTER PROGRAMMING AND ARCHITECTURE: The VAX, Second Edition**
Henry M. Levy and Richard H. Eckhouse, Jr., 1989, hardbound, 444 pages, Order No. EY-6740E-DP-EEB ($38.00)

This book is both a reference for computer professionals and a text for students. A systems approach helps the reader understand the issues crucial to the comprehension, design, and use of modern computer systems. Using the VAX computer as an example, the first half of the book is a text suitable for a complete course in assembly language programming. The second half of the book describes higher-level systems issues in computer architecture, namely, support for operating systems and operating systems structures. virtual memory, parallel processing, microprogramming, caches, and translation buffers.

**VMS FILE SYSTEM INTERNALS**
Kirby McCoy, 1990, softbound, 460 pages, Order No. EY-F575E-DP-EEB ($49.95)

*VMS FILE SYSTEM INTERNALS,* based on VMS Version 5.2, is a book for system programmers, software specialists, system managers, applications designers, and other VAX/VMS users who need to understand the interfaces to and the data structures, algorithms, and basic synchronization mechanisms of the VMS file system. This system is the part of the VAX/VMS operating system responsible for storing and managing files and information in memory and on secondary storage. The book is also intended as a case study of the VMS implementation of a file system for graduate and advanced undergraduate courses in operating systems.

**DECNET PHASE V: An OSI Implementation**
James Martin and Joe Leben, 1992, hardbound, 572 pages, Order No. EY-H882E-DP-EEB ($49.95)

This book provides a first in-depth look at DECnet Phase V and the important issues that must be resolved in the design and implementation of very large networks. It presents key Open Systems Interconnection (OSI) concepts and shows how DECnet Phase V hardware and software products implement international standards associated with the OSI model.

**VAX/VMS OPERATING SYSTEM CONCEPTS**
David Miller, 1991, hardbound, 512 pages, Order No. EY-F590E-DP-EEB ($44.95)

This book begins with an overview that centers on one visible aspect of an operating system, terminal input and output; it proceeds into well-organized chapters on process definition, paging and memory management, security, protection and privacy; and it concludes with a chapter on operating systems at Digital Equipment Corporation. Each chapter provides an introduction, theoretical discussion, generally recognized solutions, algorithms and data structures, and questions to encourage review of the central concept presented.

**THE VMS USER'S GUIDE**
James F. Peters, III and Patrick J. Holmay, 1990, softbound, 304 pages, Order No. EY-6739E-DP-EEB ($28.95)

This up-to-date guide for new VMS users provides a sequence of steps for learning the VMS operating

system and includes hands-on experiments with step-by-step instructions. The book also can be used as a reference for commands and utilities. *THE VMS USER'S GUIDE,* reflecting VMS Version 5, provides complete VMS coverage—from logging in to creating command procedures; contains a thorough discussion of files and directories; covers both the EDT and the EVE editors in detail; and introduces programming with VAXTPU. The guide includes learning aids in each chapter, such as summaries that contain tables of the commands introduced in the chapter, exercises to reinforce and extend the skills learned, and review quizzes.

**THE MATRIX: Computer Networks and Conferencing Systems Worldwide**
John S. Quarterman, 1990, softbound, 719 pages, Order No. EY-C176E-DP-EEB ($49.95)

This is the first reference book to describe in detail the extensive yet largely unpublicized web of public and private networks and conferencing systems that has spread to virtually every corner of the world. The first half provides extensive background information on the history, terminology, standards, protocols, technologies, worldwide networked communities, and probable future course of networking systems throughout the world. The second half describes specific conferencing systems and the interconnections between them—according to geographic region worldwide. Maps are included when available. Syntaxes and gateways are provided for sending mail from one system to another. Additional chapters discuss a number of well-known worldwide networks, including the Internet and selected commercial systems. Two appendices provide essential information on public data networks worldwide and on selected legal issues.

**X AND MOTIF QUICK REFERENCE GUIDE**
Randi J. Rost, 1990, softbound, 369 pages, Order No. EY-E758E-DP-EEB ($24.95)

Based on the newly released X Window System Version 11, Release 4 and Motif Version 1.0, this one-volume guide combines three major reference works on XLib, X Toolkit Intrinsics, and Motif programming libraries in a compact, easy-to-access format. Features include complete descriptions of approximately 400 XLib routines, 200 X Toolkit Intrinsics, and 200 Motif routines. The guide is organized into five major reference sections—"X Protocol," " XLib," "X Toolkit Intrinsics," "Motif," and "General X"; all routines and data structures are organized alphabetically within each of these sections.

**FIFTH GENERATION MANAGEMENT: Integrating Enterprises through Human Networking**
Charles M. Savage, 1990, hardbound, 267 pages, Order No. EY-C186E-DP-EEB ($28.95)

This book explores the challenges managers face as their organizations transition from the industrial era to the new era of knowledge networking. The author contends that new technologies like computer integrated manufacturing (CIM) will not be successful until organizations transform their structures from the steep hierarchies of second generation management to the flattened networks of the fifth generation. The book contains two parts. In Book 1, "Five Days that Changed the Enterprise," Savage narrates a case study of senior executives confronting the problems of a traditional organization as they work to transform their company into a networked organization. In Book 2, "Integrating Enterprises through Human Networking," Savage draws on contemporary management literature and his own consulting experiences to present a logical case for his recommendations. A concluding chapter offers ten practical considerations that organizations must address to prepare for change.

**X WINDOW SYSTEM: The Complete Guide to Xlib, PROTOCOL, XLFD, and ICCCM, X Version 11, Release 4, Second Edition**
Robert W. Scheifler and James Gettys, with Jim Flowers, Ron Newman, and David Rosenthal, 1990, softbound, 851 pages, Order No. EY-E755E-DP-EEB ($49.95)

By combining four MIT X Consortium standards into one volume, this book is the most complete and up-to-date X Window System reference available. In addition to the four standards, also included are instructive diagrams, a detailed glossary, and a comprehensive subject-oriented index. The book consists of four main parts, each with a standard specification produced by the MIT X Consortium for X Version 11, Release 4: Part I, "Xlib–C Language X Interface"; Part II, "X Window System Protocol"; Part III, "Inter-Client Communications Conventions Manual"; and Part IV, "X Logical Font Description."

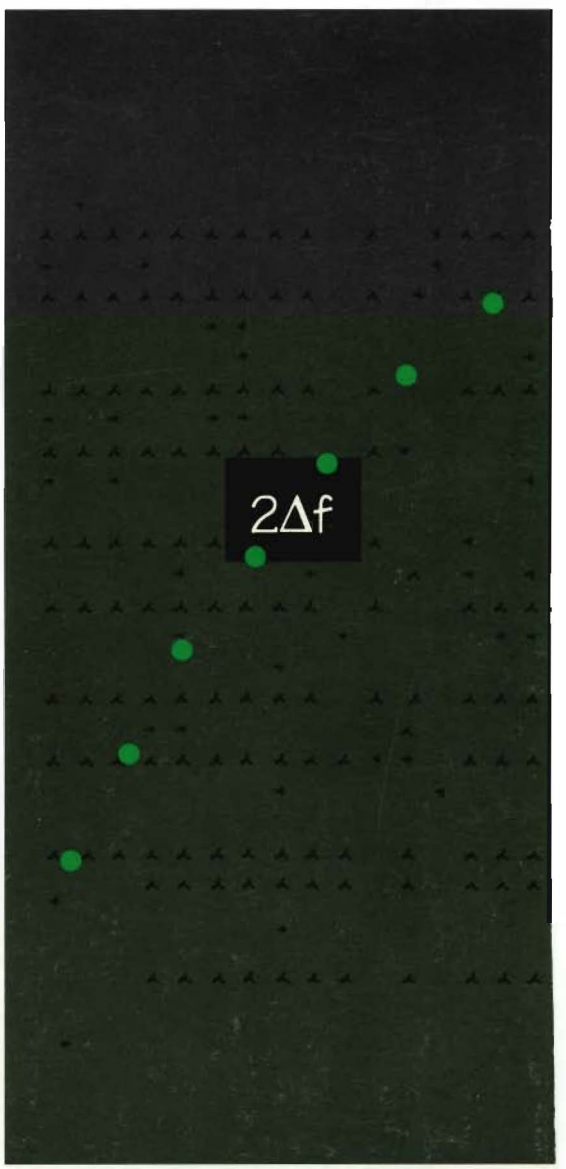
To receive a copy of our latest catalog or further information on these or other publications from Digital Press, please write:

Digital Press
Department EEB
1 Burlington Woods Drive
Burlington, MA 01803-4539

Or, you can order by calling DECdirect at 800-DIGITAL (800-344-4825).

When ordering be sure to refer to Catalog Code EEB.

digital™




ISSN 0898-901X

Printed in U.S.A. EY-H889E-DP/91 12 02 18.0 DBP/NRO 