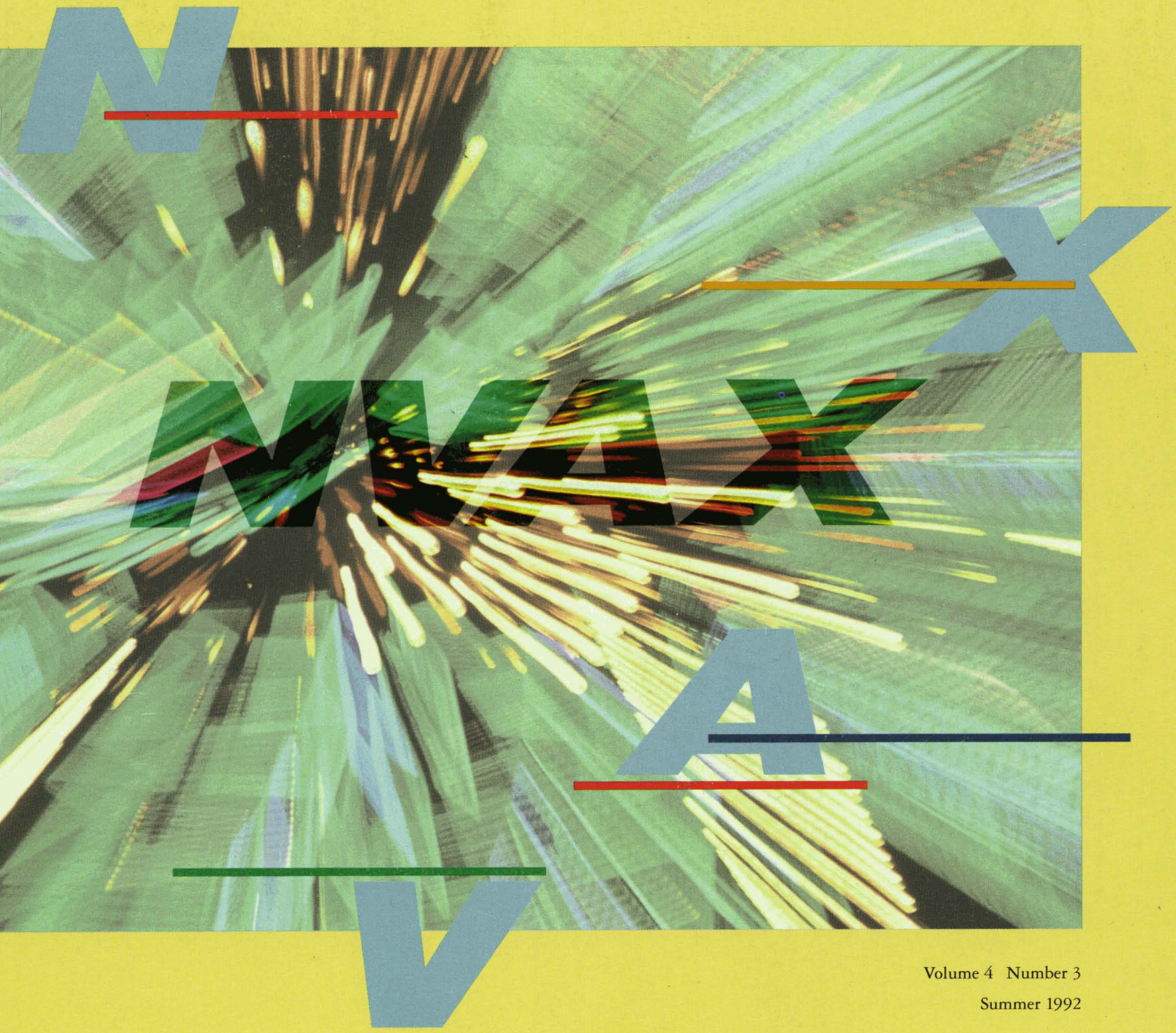
NVAX-microprocessor VAX Systems

Digital Technical Journal

Digital Equipment Corporation



Volume 4 Number 3

Summer 1992

The *Digital Technical Journal* is published quarterly by Digital Equipment Corporation, 146 Main Street MLO 1-3/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 01830-4597.

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-J884E-DP

Book production was done by Quantic Communications, Inc.

Cover Design

The NVAX microprocessor is Digital's fastest VAX implementation and the common theme of papers in this issue. Our cover graphic joins the NVAX project code name with an image of speed — the chip's most salient characteristic and the performance advantage it brings to a range of new VAX systems.

The cover design is by Deb Anderson of Quantic Communications, Inc.

Contents

Editor's Introduction

Jane C. Blake
Editor

The NVAX microprocessor is a high-performance, single-chip implementation of the VAX architecture. It is today's fastest VAX microprocessor and the CPU at the heart of the mid-range, low-end, and workstation systems described in this issue of the *Digital Technical Journal.*

The NVAX chip is not only fast, with cycle times as low as 11 ns, but also holds a unique position in the Digital family of microprocessors: NVAX is both an upgrade path for existing VAX systems and a migration path to Alpha AXP systems. In their paper on the NVAX and NVAX+ chips, Mike Uhler, Debra Bernstein, Larry Biro, John Brown, John Edmondson, Jeff Pickholtz, and Rebecca Stamm present an overview of the complex microprocessor designs and relate how RISC techniques are used in this CISC machine to achieve dramatic increases in performance over previous implementations.

Increases in performance are also attributable to the CMOS-4 0.75-micrometer process technology in which the NVAX is implemented. In their paper about the verification of the physical design, Dale Donchin, Tim Fischer, Frank Fox, Victor Peng, Ron Preston, and Bill Wheeler describe the methods and the CAD tools created to manage the complexity of a chip with 1.3 million transistors.

The rigorous use of the CAD tools and thorough simulation-based testing resulted in highly functional first-pass chips. In his paper about the logical verification, Walker Anderson discusses the successful strategies used to ensure no "show stopper" bugs existed in the design. Highlighting major strategies, he reviews the behavioral models and pseudorandom exercisers at the core of the verification effort.

Each system design team chose a different approach to take advantage of NVAX performance and to meet system-specific requirements. In a paper on the new mid-range VAX 6000 multiprocessing system, Larry Chisvin, Gregg Bouchard, and Tom Wenners explain the module design decisions that supported the goals of 6000-series compatibility and time to market. Of particular interest are the schedule and performance benefits derived from developing a routing and control interface chip.

The engineers for new low-end deskside systems also chose to develop custom chips—a memory controller chip, memory module, and an I/O controller. Jon Crowell, Kwong Chui, Tom Kopec, Sam Nadkarni, and Dean Sovie discuss the chip functions that were key to exceeding the performance goal of three times that of the previous VAX 4000.

For the low-end VAX 4000 Model 100 system and the MicroVAX 3100 desktop servers, designers saved significant time by "borrowing" existing components from proven systems. Jon Crowell and Dave Maruska relate decisions that allowed them to double performance and complete the work within the extraordinarily short time of nine months.

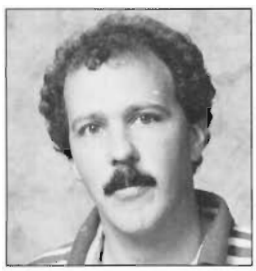
The newest VAXstation workstation, based on NVAX, is the Model 90. Mike Callander, Lauren Carlson, Andy Ladd, and Mitch Norcross present their design methodology. Most significant for development was the decision to implement new logic in programmable technology, which allowed bug fixes in minutes rather than weeks.

Not about system design but rather error handling in 6000 systems, Brian Porter's paper describes an approach that reduces the amount of unique coding traditionally required for error handling. He details the development of sophisticated error handling routines that accommodate the complexity of the symmetric multiprocessing VAX 6000 models.

The editors thank Mike Uhler of the Semiconductor Engineering Group, who ensured that the standards of excellence applied to NVAX development were applied to the development of this issue. Also, this issue is notable editorially because it is the first in which papers have been formally refereed. I thank Gene Hoffnagle, editor of the *IBM Systems Journal,* for encouraging the use of the referee process in any journal worthy of the name. DTJ issues will continue to be refereed so that we may offer engineering and academic readers informative and relevant technical discussions.

Jane Blake

Biographies



Walker Anderson Principal engineer Walker Anderson is a member of the Models, Tools, and Verification Group in the Semiconductor Engineering Group. Currently a co-leader of the logical verification team for a future chip design, he led the NVAX logical verification effort. Before joining Digital in 1988, Walker was a diagnostic and testability engineer in a CPU development group at Data General Corporation for eight years. He holds a B.S.E.E. (1980) from Cornell University and an M.B.A. (1985) from Boston University.

Debra Bernstein Debra Bernstein is a consultant engineer in the Semiconductor Engineering Group. She worked on the CPU design and architecture for the NVAX microprocessor and the VAX 8700/8800 systems and is currently co-architecture leader for a future Alpha AXP processor. Deb received a B.S. in computer science (1982, cum laude) from the University of Massachusetts in Amherst. She holds one patent, has two patent applications pending, and has coauthored several technical papers.

Larry L. Biro Larry Biro joined the Electronic Storage Development (ESD) Group in 1983, after receiving an M.S.E.E. from Rensselaer Polytechnic Institute. While in ESD, he contributed to the advanced development of solid-state disk products. Larry joined the Semiconductor Engineering Group as a custom circuit designer on the NVAX E-box. Later, as a member of the NVAX+ chip implementation team, Larry designed the clock and reset logic, coordinated back-end verification efforts, and co-led the chip debugging. Currently, he is the project leader for a future, single-chip VAX implementation.

Gregg A. Bouchard Gregg Bouchard is a senior hardware engineer with the Semiconductor Engineering Group. His current responsibilities include the module design of a DECchip 21064 daughter card that contains a CPU-to-bus interface. Previously, Gregg worked on chip design of the NVAX-to-XMI bus interface for the VAX 6000 Model 600, and field programmable gate array chips for the VAXstation 4000 Model 90. Gregg joined Digital in 1986 after receiving his B.S.E.E. from the Rochester Institute of Technology. He also holds an M.S.E.E. from Northeastern University and has a patent pending related to hardware queue structure.

John F. Brown After receiving an M.S.E.E. from Cornell University in 1980, John Brown joined the engineering staff at Digital. At present, he is a consultant engineer working on Alpha AXP microprocessor advanced development. John's previous responsibilities include managing the design of the instruction decode section of the NVAX microprocessor. He also made technical contributions to the VAX 6000 Model 200 and 400 chip sets, and was hardware engineer for the extended floating-point enhancement to the VAX-11/780 system. John holds two patents and has seven applications pending.

Michael A. Callander, Sr. Michael Callander is a principal engineer in Digital's Semiconductor Engineering Group. Mike was the technical leader for the VAXstation 4000 Model 90 system. His previous experience with Digital includes design and architectural specification for various CPU modules and systems, including the VAX 8200 and the VAX 6000 Model 400 and Model 500. Mike received his B.S.E.E. from the University of Massachusetts in 1982 and joined Digital upon graduation. He has authored several technical papers and has a number of patent applications pending.

Lauren M. Carlson A senior hardware engineer in the Semiconductor Engineering Group, Lauren Carlson is currently working on the design of a peripheral chip set for a new microprocessor. Previously, she designed the VAXstation 4000 Model 90 CDAL-to-EDAL adapter chip (CEAC) gate array, which is part of the I/O subsystem. Lauren also contributed to the design of the VAXstation 4000 Model 90 system module and another VAX system CPU module. Prior to this, Lauren worked in the Advanced VAX Development Group. She received her B.S.E.E. from Worcester Polytechnic Institute in 1986 and joined Digital in 1987.

Lawrence Chisvin A principal hardware engineer in the Semiconductor Engineering Group, Larry Chisvin is involved in the design of modules and systems based on the DECchip 21064 microprocessor. Larry also provides technical support for customers, including Alpha AXP architecture presentations and example designs and application notes. Previously, he worked on processor and memory modules for the VAX 6000 Model 600. He holds a B.S.E.E. (summa cum laude) from Northeastern University and an M.S.E.E. from Worcester Polytechnic Institute. He is a member of the IEEE Computer Society and the ACM.

Kwong-Tak A. Chui Kwong-Tak Chui is a principal hardware engineer in the Semiconductor Engineering Group. He is working on the design of the I/O controller section of a new CPU. Kwong was the project leader for the I/O controller chip of the NVAX chip set used in the VAX 4000 Model 400, 500, and 600 systems. Since joining Digital in 1985, he has worked on four other VLSI chip design projects for the VAX 3000 and VAX 4000 series computers. Kwong holds a B.S. in computer engineering (1985) from the University of Illinois at Urbana-Champaign and an M.S.E.E. (1989) from Cornell University.

Jonathan C. Crowell An engineering manager in the Entry Systems Business Group, Jon Crowell was the project leader and system engineer on the VAX 4000 Models 100, 400, 500, and 600 and the MicroVAX 3800, 3900, and 3100 Model 90 systems. He is now working on the design of the next generation of VAX 4000 systems. Previously, Jon worked in the Systems Integration Group qualifying Q-bus devices and DSSI adapters and storage devices. He joined Digital in 1986. Jon received a B.S.E.E. (1981) and an M.S.E.E. (1986) from Northeastern University. He hold six patents and is an active member of IEEE.

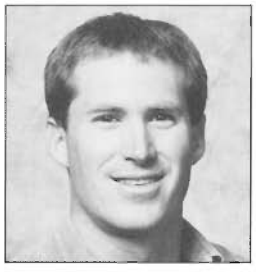
Dale Donchin Dale Donchin manages schematic entry and layout verification CAD tool development in the Semiconductor Engineering Group. He facilitated the use of CAD tools for NVAX design, primarily for layout and physical chip verification. Dale is presently performing in a similar capacity for a new microprocessor design based on the Alpha AXP architecture. He joined Digital in 1978, and was previously a development manager in the RSX operating system group. Dale holds a B.S.E.E. (1976, honors) and an M.S.E.E. (1978) from Rutgers University College of Engineering and is a member of IEEE and ACM.

John H. Edmondson John Edmondson is a principal engineer in the Semiconductor Engineering Group. At present, he is co-architect of a future RISC microprocessor. Previous to this, he was a member of the VAX 6000 Model 600 CPU chip design team. Before joining Digital in 1987, John designed minicomputers for five years at Canaan Computer Corporation. He also worked at Massachusetts General Hospital for two years, researching applications of technology to anesthesia and intensive care medicine. John received a B.S.E.E from the Massachusetts Institute of Technology in 1979.

Timothy C. Fischer Tim Fischer is a senior hardware engineer with the Semiconductor Engineering Group. He is working on the design of a floating-point unit for a future high-performance microprocessor based on the Alpha AXP architecture. Prior to this work, Tim was a member of the E-box design team and contributed to the design of global clock generation and distribution for the NVAX microprocessor. He also worked on the design of the bus interface unit on the NVAX+ chip. Tim joined Digital in 1989 after receiving his M.S. in computer engineering from the University of Cincinnati.

Thomas F. Fox Frank Fox, a consulting engineer in the Semiconductor Engineering Group, co-led the implementation of the NVAX microprocessor and consulted with the Advanced Semiconductor Development Group on the design of the CMOS-4 technology. He joined Digital in 1984 and worked on the implementation of the CVAX microprocessor. Frank received a B.E. degree from University College Cork, National University of Ireland (1974), and a Ph.D. degree from Trinity College, Dublin University (1978), both in electrical engineering. He has published papers on ultrasonic instrumentation, MRI scanners, and VLSI design.

Thomas E. Kopec Principal engineer Tom Kopec was a member of the Entry Systems Business Group that designed the VAX 4000 Models 200 through 600, and the MicroVAX 3500 and 3800 systems. He also led an Alpha AXP development project. Recently, Tom joined the Assistive Technologies Group to work on compact speech-synthesis and text-to-voice systems. He received a B.S.E.C.E. (1980, with honors), concentrating in microwave engineering and signal processing, and an M.S.E.C.E. (1985), concentrating in image processing and computer graphics, both from the University of Massachusetts at Amherst.



Andrew R. Ladd Andy Ladd is a principal engineer in the Semiconductor Engineering Group and is the project leader for the VAXstation 4000 Model 90 CPU and low-cost graphics module designs. Previously, he provided timing verification support for the DECchip 21064 and co-designed two bus interface chips for the VAX 6000 Model 400 CPU module. Andy joined Digital in 1986. He received his B.S. in computer engineering from the University of Illinois (1984) and his M.S. in computer science and engineering from the University of Michigan (1991). Andy is a member of the IEEE Computer Society, Tau Beta Pi, and Eta Kappa Nu.

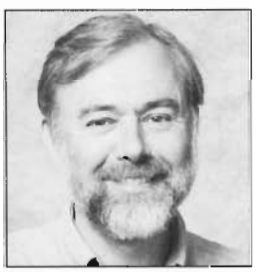
David W. Maruska Principal engineer David Maruska is a member of the Entry Systems Business Group and is presently involved in the design of the next generation of VAX 4000 CPUs. He was the lead designer for the KA50 and KA52 CPUs and project leader for the VAX 4000 Model 200 system, the KZQSA Q-bus-to-SCSI adapter, and the Futurebus+ exerciser. Dave joined Digital in 1982, after receiving a B.S. in computer engineering from Boston University. He worked on graphics workstations for Mosaic Technologies and Raster Technologies from 1983 to 1985 and then returned to Digital in 1986.

Samyojita A. Nadkarni Sam Nadkarni, a principal hardware engineer in the Semiconductor Engineering Group, is currently the project leader for a set of support chips for the DECchip 21064 processor. She was the project leader for the NMC chip used in the VAX 4000 Model 400, 500, and 600 systems. She also worked on memory controller/bus adapter chips for the VAX 4000 Model 300 and MicroVAX 3500 systems. Sam joined Digital in 1985 and holds a Bachelor of Technology (1983) from the Indian Institute of Technology and an M.S. (1985) from Rensselaer Polytechnic Institute.

Mitchell O. Norcross Senior engineer Mitch Norcross has been designing and analyzing digital subsystems since he joined Digital in 1986. As a member of the Semiconductor Engineering Group, Mitch designed a gate array for the VAXstation 4000 Model 90 and contributed to the design and analysis of several system and CPU modules. Prior to joining SEG, Mitch designed a gate array for Digital's first fault-tolerant VAX system, the VAXft 3000. He received a B.E. in electrical engineering (1985) and an M.S. in computer engineering (1987), both from Manhattan College. Mitch holds one patent on fault-tolerant system design.

Victor Peng Victor Peng is a consultant engineer in the Semiconductor Engineering Group. He received his B.S. in electrical engineering from Rensselaer Polytechnic Institute in 1981, and his M.S. in electrical engineering from Cornell University in 1982. Victor joined Digital in 1982. He was a member of the design team for the VAX 8200/8300 memory interface chip and patchable control store chip. He led the implementation of the VAX 6000 Model 400 floating-point chip and was co-manager of the NVAX chip design team.

Jeffrey D. Pickholtz Jeffrey Pickholtz received an A.S. (1977) in specialized technology from Penn Technical Institute and a B.S.E.E.T. (1989) from Central New England College. He joined Digital in 1977 as a technician and worked on a variety of midrange computers. More recently, his responsibilities have included technical contributions to the VAX 6000 Models 400 and 600, and VAX 7000 Model 600 chip sets. Currently, Jeff is a senior engineer in the Semiconductor Engineering Group, leading the implementation of a future CMOS microprocessor chip.

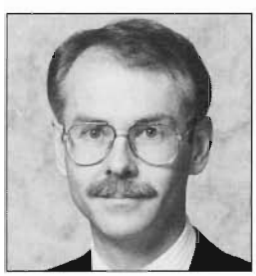


Brian Porter As a consulting software engineer in the Systems Group of VMS Development, Brian Porter was responsible for CPU error handling in the VAX 6000 family. Prior to this work, he was responsible for support of VAX systems and was an author and maintainer of the VMS error log utility SYE. Brian is the author of the original VMS striping driver, which was later developed by others into the VMS striping driver product. He currently works in the Executive Group of VMS Development and is responsible for symmetric multiprocessing. He has two patents pending on memory error handling. Brian joined Digital in 1973.

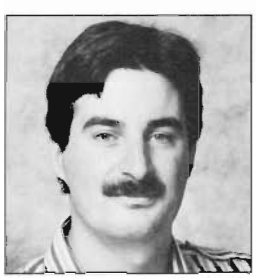
Ronald P. Preston Ronald Preston is a principal engineer in the Semiconductor Engineering Group. Since joining Digital in 1988, he has worked on the design of several microprocessors. Ron was the circuit design and implementation leader for the E-box on the NVAX microprocessor. He is currently designing the instruction issue logic for a superscalar RISC microprocessor. Prior to joining Digital, he worked on the design of CMOS microcontrollers for Signetics Corporation. Ron received his B.S.E.E. in 1984 and his M.S.E.E. in 1988 from Rensselaer Polytechnic Institute. He is a member of Eta Kappa Nu and IEEE.

Dean A. Sovie Design engineer Dean Sovie joined Digital in 1981 and is a member of the Electronic Storage Development Group. He is currently involved in the battery backup laser memory design. Prior to this work, he contributed to the design of the high-performance memory module used in the VAX 4000 Model 400, 500, and 600 systems. Dean also helped design memories for the VAX 6000, VAX 4000 Model 200, and PDP-11 systems. In addition to his present responsibilities, he is pursuing a degree in electrical engineering at Northeastern University.

Rebecca L. Stamm Rebecca Stamm is a principal hardware engineer in the Semiconductor Engineering Group. She led the design of the backup cache, the bus interface, and the pin bus for the NVAX CPU chip. Rebecca then led the chip debug team from tapeout through final release of the design to manufacturing. Since joining Digital in 1983, Rebecca has been engaged in microprocessor architecture and design. She holds two patents and has seven applications pending. Rebecca received a B.A. in history from Swarthmore College and a B.S.E.E. from the Massachusetts Institute of Technology.



G. Michael Uhler Michael Uhler is a senior consultant engineer in the Semiconductor Engineering Group, where he leads the advanced development effort for a new high-performance microprocessor. As chief architect for the NVAX and REX520 microprocessors, Mike was responsible for the CPU architecture, performance evaluation, behavioral modeling, CPU microcode, and CPU and system debug. He received a B.S.E.E. (1975) and an M.S.C.S. (1977) from the University of Arizona and joined Digital in 1978. Mike is a member of IEEE, ACM, Tau Beta Pi, and Phi Kappa Phi and holds eight patents.



Thomas M. Wenners Thomas Wenners is a senior hardware engineer in the Semiconductor Engineering Group. He is responsible for the design of a next-generation VAX workstation CPU module and for CPU module designs of DECchip 21064 microprocessor-based products. Tom's previous work includes the module design of the VAX 6000 Model 600, module design and signal integrity support on ESB products, and analysis and evaluation of advanced chip and module packaging. Tom joined Digital in 1985. He received a B.S.E.E. (1985, cum laude) and an M.S.E.E. (1990) from Northeastern University.

William R. Wheeler A principal engineer in the Semiconductor Engineering Group, Bill Wheeler performed architectural definition work for the NVAX microprocessor and was project leader of the NVAX M-box. Prior to this work, he designed the I-box unit for the third-generation VLSI VAX implementation. He is currently involved in advanced development work for improving custom design tools and methods. Before joining Digital in 1983, Bill received both his B.S.E.E. (1982) and his M.S.E.E. (1983) from Cornell University. He has coauthored three papers on VLSI designs and holds three patents.

Foreword

Robert M. Supnik
Corporate Consultant,
Vice President
Technical Director,
Alpha AXP and VAX Systems

If, as the popular saying goes, "Once is happenstance, twice is coincidence, three times is concerted action," then four consecutive instances of outstanding engineering achievement must be even more significant.

Since 1985, Digital has designed, developed, and shipped four generations of leadership VAX microprocessors and CMOS-based systems:

- In 1985, the MicroVAX chip and resulting systems (such as the MicroVAX II and the VAXstation 2000)
- In 1987, the CVAX chip and resulting systems (such as the MicroVAX 3800, the VAX 6000-200, and the VAXstation 3100)
- In 1989, the Rigel chip and resulting systems (such as the VAX 4000-300, the VAX 6000-400, and the VAXstation 4000-60)
- In 1991, the NVAX chip and resulting systems (such as the VAX 4000-500, the VAX 6000-600, and the VAXstation 4000-90)

The first three were described in the *Digital Technical Journal* issues of March 1986, August 1988, and Spring 1990, respectively; the last is the subject of this issue.

NVAX and its systems are the culmination of everything Digital and its engineers have learned about chip and system design over the last decade. The teams involved drew on many disciplines of hardware engineering, from microarchitecture to whole-system verification, to produce products of unparalleled performance and quality. The results speak for themselves.

- From its initial shipment in October 1991 through today (a year later), NVAX was (and is) the fastest shipping CISC microprocessor in the world, whether measured by clock rate, SPECmarks, or transactions per second.
- NVAX had fewer bugs after design completion, and went from tape-out to production more quickly than any microprocessor in Digital's history.
- NVAX systems, spanning the range from workstation through mainframe, all shipped on or ahead of schedule, meeting or exceeding predicted performance.

An outstanding engineering achievement indeed!

The roots of NVAX can be traced back a decade to two distinct engineering programs: the High-end Systems Group's studies and implementations of highly pipelined VAX systems; and the Semiconductor Operations Group's projects in process development and microprocessor design.

The High-end Systems Group started work on highly parallel VAX systems in 1979, designing and building the VAX 8600—the first VAX to include overlapped operand decoding (see the *Digital Technical Journal,* August 1985). At the same time, a research team described HyperVAX, a hypothetical fully pipelined design. Although HyperVAX was never built, its microarchitecture had a strong influence on the design of the VAX 9000, Digital's ECL mainframe (see the *Digital Technical Journal,* Fall 1990). And the microarchitecture of the VAX 9000, in turn, was the basis for NVAX.

The Semiconductor Operations Group also started work in 1979, formulating a multiyear program for the development of both semiconductor process technology and leading-edge microprocessors. This program spanned the years 1983 to 1987 and encompassed the development of the V-11, MicroVAX II, and CVAX microprocessors. In 1986, the plan was extended through 1991, encompassing the development of Rigel, Mariah (a Rigel variant), and a fourth-generation VLSI VAX code-named NVAX.

The goals for NVAX were ambitious. First, its targeted performance was more than 25 times faster than the VAX-11/780 (more than 10 times faster than the just-introduced CVAX chip), requiring significant improvements in both microarchitectural efficiency and in cycle time. Second, the chip development schedule coincided with the

semiconductor process development schedule, requiring breakthroughs in concurrent development of product and process. And third, the time allotted from chip design completion to system shipment was the shortest in Digital's history, requiring unprecedented accuracy in chip and system design and verification.

As in past projects, work in various disciplines—semiconductor process development, chip microarchitecture and circuit design, microprocessor design tools, chip and system verification tools, and system design—cascaded from process through systems. First to start was a team from Advanced Semiconductor Development (ASD), which designed, simulated, and introduced into manufacturing CMOS-4, Digital's fourth generation of CMOS technology (see the *Digital Technical Journal*, Spring 1992). Building on prior technology generations, CMOS-4 contained many features—three layers of metal interconnect, salicide, precision resistors, local interconnect, deep diffusion ring—which directly supported the performance requirements of NVAX. In addition, ASD and Semiconductor Manufacturing pioneered new techniques for process transfer and qualification which dramatically shortened the time required to debug and qualify the CMOS-4 process.

In parallel, a design team from the Semiconductor Engineering Microprocessor Group initiated microarchitectural and circuit studies. The team started with the VAX 9000, but they quickly discovered that the difference in implementation media (multichip ECL gate arrays for the VAX 9000, single-chip custom CMOS for NVAX) required significant changes and new concepts. The microarchitecture sub-team used abstract and detailed performance models, studies from existing VAX systems, and experience with past designs to drive quantitative decisions about features and functions in NVAX. At the same time, the circuit sub-team formulated the overall design, circuit, and clocking methodologies for the chip and established the feasibility of the target cycle time, chip size, and layout floor plan.

As the microarchitectural concepts solidified, the design team realized that NVAX would be the largest and most complex chip ever designed at Digital, and that it would place unprecedented stress on the capabilities of both designers and design tools. Accordingly, they initiated a partnership with the Semiconductor Engineering CAD Group to improve current tools and to develop new tools. In addition to traditional areas like simulation, CAD development focused on improvements to productivity and accuracy through design synthesis, electromigration analysis, and capacitance and resistance extraction.

The size and complexity of the design, as well as the stringent schedule constraints, also dictated an early start on verification issues. The verification strategy formed an integral part of the design effort from the outset. The verification team developed tools and strategies for verifying the microarchitecture, the microcode, the logic, the circuits, the chip as a component, and the chip in a system.

Lastly, the various system groups—data center systems, office systems, workstations—began designing systems to utilize the NVAX chip's capabilities. Each group was able to build on the work done in past VAX systems and designed an NVAX-based system that functioned both as an upgrade of past systems and as a formidably competitive new system in its own right.

The work of these project teams dovetailed perfectly. NVAX completed design and taped-out in late November 1990, just as the CMOS-4 process was ready for chip prototyping. Due to the outstanding work of the chip design, system design, CAD, and verification teams, first-pass parts booted the VMS operating system at speed in early March 1991. The process team qualified CMOS-4 in October 1991, and systems using second-pass parts shipped for revenue that same month—three months ahead of schedule—with performance significantly greater than the original goal.

Clearly, the outstanding results from all the NVAX engineering projects are neither happenstance nor coincidence; rather, they represent concerted action—team excellence and individual brilliance—at its finest. Hundreds of people contributed to the outcome. This issue of the *Digital Technical Journal* is their story.

G. Michael Uhler
Debra Bernstein
Larry L. Biro
John F. Brown III
John H. Edmondson
Jeffrey D. Pickholtz
Rebecca L. Stamm

The NVAX and NVAX+ High-performance VAX Microprocessors

The NVAX and NVAX+ CPU chips are high-performance VAX microprocessors that use techniques traditionally associated with RISC microprocessor designs to dramatically improve VAX performance. The two chips provide an upgrade path for existing VAX systems and a migration path from VAX systems to the new Alpha AXP systems. The design evolved throughout the project as time-to-market, performance, and complexity trade-offs were made. Special design features address the issues of debug, maintenance, and analysis.

The NVAX and NVAX+ CPUs are high-performance, single-chip microprocessors that implement Digital's VAX architecture.[1] The NVAX chip provides an upgrade path for existing systems that use the previous generation of VAX microprocessors. The NVAX+ chip is used in new systems that support Digital's DECchip 21064 microprocessor, which implements the Alpha AXP architecture.[2,3] These two NVAX chips share a basic design.

The high-performance, complementary metal-oxide semiconductor (CMOS) process used to implement both chips allows the application of pipelining techniques traditionally associated with reduced instruction set computer (RISC) CPUs.[4] Using these techniques dramatically improves the performance of the NVAX and NVAX+ chips as compared to previous VAX microprocessors and results in performance that approaches and may even exceed the performance of popular industry RISC microprocessors.

The chip design evolved throughout the project as the goals influenced the schedule, performance, and complexity trade-offs that were made. The two primary design goals were time-to-market, without sacrificing quality, and improved VAX CPU performance. Our internal goal was for the NVAX CPU performance to be more than 25 times the performance of a VAX-11/780 system in a datacenter system. Achieving these goals required meeting aggressive schedules and thus concentrating on the high-leverage design points and on an unprecedented verification effort.[5]

Support for multiple system environments, compatibility with previous VAX products and systems, and a means to migrate from traditional VAX systems to the new Alpha AXP platforms were also important design goals. These goals had a profound impact on the design of the cache protocols and the external bus interfaces. NVAX and NVAX+ engineers worked closely with engineers in Digital's systems groups during the definition of these operations.

The paper begins by comparing the basic features of the NVAX and NVAX+ chips and then describes in detail the chip interfaces and design elements. This description serves as the foundation for the ensuing discussion of design evolution and trade-offs. The paper concludes with information about the special design features that address the issues of debug, maintenance, and analysis.

Comparison of the NVAX and NVAX+ Chips

The NVAX and NVAX+ chips are identical in many respects, differing primarily in external cache and bus support. NVAX is intended for systems that use previously designed VAX microprocessors. The following systems currently use the NVAX chip: the VAXstation 4000 Model 90; the MicroVAX 3100 Model 90; the VAX 4000 Models 100, 400,

500, and 600; and the VAX 6000 Model 600.[6,7,8,9] NVAX supports an external write-back cache that implements a directory-based broadcast coherence protocol that is compatible with earlier VAX systems.[10]

NVAX+ is designed for systems that use the DECchip 21064 microprocessor implementation of the Alpha AXP architecture and is currently used in the VAX 7000 Model 600 and the VAX 10000 Model 600 systems. NVAX+ supports an external cache and bus protocol that is compatible with that of the DECchip 21064 microprocessor. In existing systems, NVAX+ is configured to support an external write-back cache that implements a conditional write-update snoopy coherence protocol.[11]

The two CPU chips provide both the means to upgrade installed VAX systems, thus protecting previous investments, and a migration path from a VAX microprocessor to a DECchip 21064 microprocessor in the new Alpha AXP systems.

Chip Interfaces

The NVAX chip interfaces to an external write-back cache (B-cache) through a private port with tag and data static random-access memories (RAMs) on the module, as shown in Figure 1. The size and speed of the cache are programmable, allowing the chip to accommodate a range of possible system configurations.

The NVAX data and address lines (NDAL) constitute a 64-bit bidirectional external bus with associated control signals that operates at one-third the frequency of the CPU from clocks provided by the CPU. Addresses and data are time-multiplexed and, to provide high performance, are overlapped with arbitration for future transactions and acknowledgment of previous transactions. The NDAL bus protocol allows up to two disconnected reads and multiple write-backs to be outstanding at the same time, using identifiers to distinguish the different transactions. External interrupt requests are received through dedicated lines and arbitrated by logic in the CPU.

The NVAX+ chip interfaces to an external write-back B-cache implemented with tag and data static RAMs on the module through a port shared with system control logic, as shown in Figure 2. Responsibility for controlling the cache port is shared between NVAX+ and the system environment; the NVAX+ chip handles the common cases of read hit and exclusive write, and the system environment provides cache policy control for other events. The size and speed of the cache can be configured to allow a range of possible system configurations.

The DECchip 21064 data and address lines (EDAL) constitute a demultiplexed, bidirectional bus with 29 bits of address, 128 bits of data, and the associated control signals. This bus operates at one-half, one-third, or one-fourth the frequency of the CPU from clocks provided by the CPU. The speed of the system clocks can be programmed to accommodate various RAM and system speeds. At power-up time, initialization information, including RAM timing, and diagnostics are loaded from a serial read-only memory (ROM) into the on-chip cache. The external interrupt handling is similar to that of the NVAX chip.

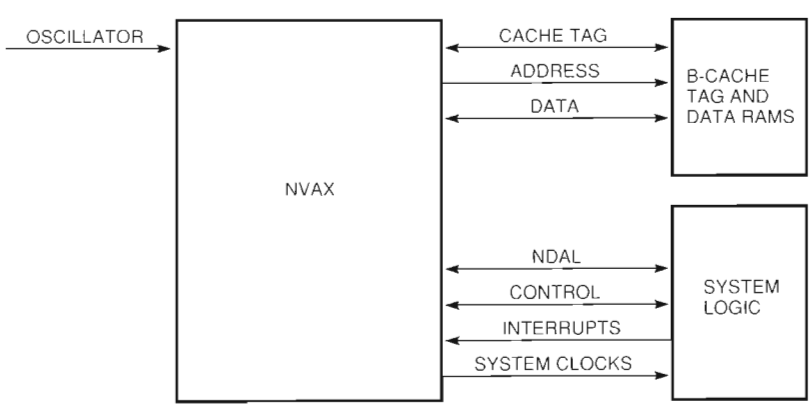


Figure 1 NVAX Chip Interface Block Diagram

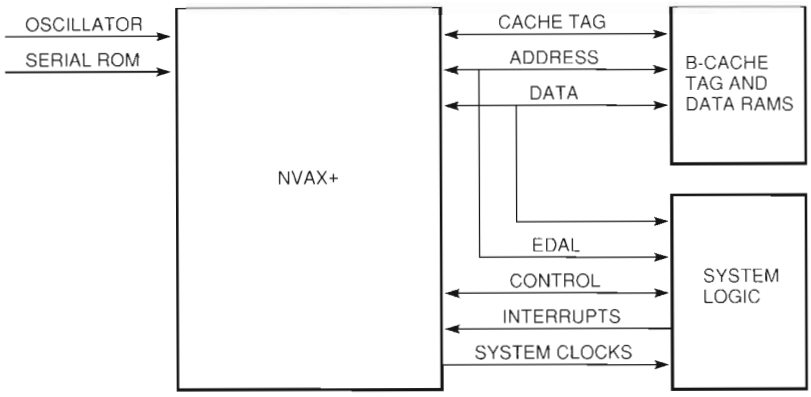


Figure 2 NVAX+ Chip Interface Block Diagram

Electrical and Physical Design

Process technology, clocking scheme, clock frequency, and die specifications are elements of the electrical and physical design of the NVAX and NVAX+ chips. Both chips are implemented in Digital's fourth-generation complementary metal-oxide semiconductor (CMOS-4) technology. CMOS-4 is a 0.75-micrometer, 3.3-volt process with support for 5-volt input signals at the pins. The CMOS-4 process is optimized for high-performance microprocessors and provides short (0.5-micrometer) channel lengths and three layers of metal interconnect. This robust and reliable process has been used to produce NVAX chips in volume for more than a year and is the same CMOS process used in the DECchip 21064 microprocessor.

NVAX and NVAX+ use a four-phase clocking scheme, driven by an oscillator that operates at four times the internal clock frequency. The oscillator frequency is divided by an on-chip, finite-state-machine clock generator; a low-skew clock distribution network is used for both internal and external clocks.

To meet the needs of the system designer, the two chips are designed for use at various frequencies. At present, NVAX is used in systems at internal clock frequencies of 83.3 megahertz (MHz) (12-nanosecond [ns] clock cycles), 74.4 MHz (14-ns clock cycles), and 62.5 MHz (16-ns clock cycles). NVAX+ is used in systems at a frequency of 90.9 MHz (11-ns clock cycles).

Each chip contains 1.3 million transistors on a die that is 16.2-by-14.6 millimeters in size. NVAX is packaged in a 339-pin, through-hole pin grid array. NVAX+ is packaged in a 431-pin, through-hole pin grid array.

Architectural Design

The NVAX/NVAX+ design is partitioned into five relatively autonomous functional units: the instruction fetch and decode unit (I-box), the integer and logical instruction execution unit (E-box), the floating-point execution unit (F-box), the address translation and primary cache interface (M-box), and the external cache and system bus interface (C-box). Queues placed at critical interface boundaries normalize the rate at which the units process instructions. A block diagram of the NVAX and NVAX+ core is shown in Figure 3.

The I-box

The I-box fetches and decodes VAX instructions, evaluates operand specifiers, and queues operands in canonical form for further processing. Included in the I-box is a 2-kilobyte (KB), direct-mapped virtual instruction cache (VIC) with 32-byte cache blocks. For reliability, the VIC includes parity protection on both tags and data.

During each cycle, the I-box attempts to fetch 8 bytes of instruction data from the VIC and place this data in an empty slot in the prefetch queue (PFQ). A VIC miss incurs a three-cycle penalty if the requested data is found in the primary cache. PFQ data is then decoded into the next VAX instruction component, which may be one of the following: operation code (opcode) and first specifier or branch displacement, subsequent specifier, or implicit specifier (an imaginary specifier included to improve the performance of some instructions). The I-box enters the opcode-related information into the instruction queue, the pointers to source and destination operands into their respective source and destination

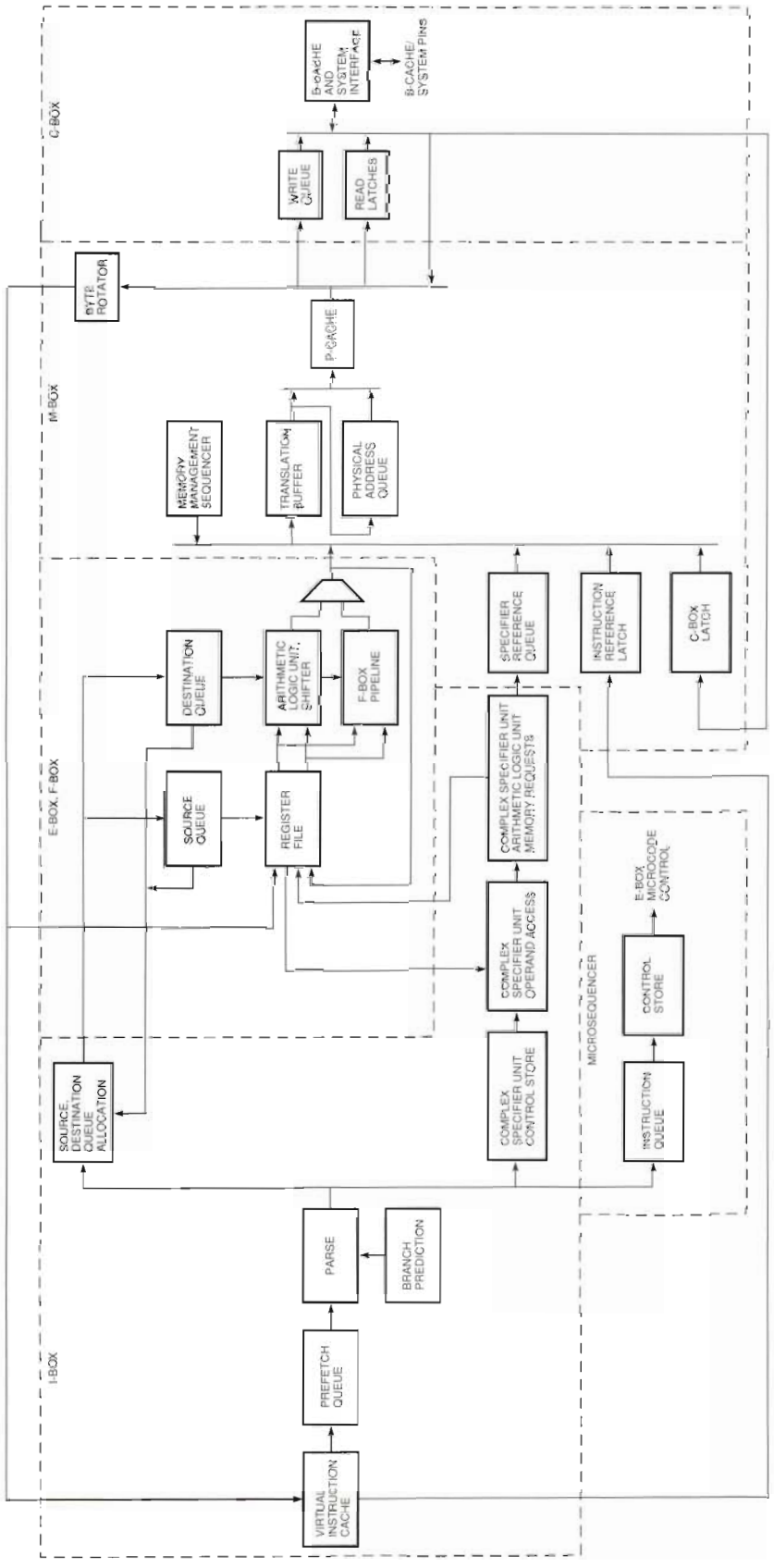


Figure 3 Block Diagram of the NVAX/NVAX+ Core

queues, and the branch-related information into the branch queue.

For operand specifiers other than short literal or register mode, the I-box decode logic invokes the pipelined complex specifier unit (CSU) to compute the effective address and initiate the appropriate memory request to the M-box. The CSU is similar in function to the load/store unit on many traditional RISC machines.

The I-box automatically redirects the program counter (PC) to the target address when it decodes one of the following instruction types: unconditional branch, jump, and subroutine call and return. The branch-taken penalty is two cycles for any conditional or unconditional branch. To keep the pipeline full across conditional branches, the I-box includes a 512-bit by 4-bit branch prediction array. The prediction is entered in the branch queue by the I-box and compared with the actual branch direction by the E-box. If the I-box predicts incorrectly, the E-box invokes a trap mechanism to drain the pipeline and restart the I-box at the alternate PC. A branch mispredict incurs a four-cycle penalty for a branch that is actually taken and a six-cycle penalty for a branch that is not taken.

The E-box

The E-box is responsible for the execution of all non-floating-point instructions, for interrupt and exception handling, and for various overhead functions. All functions are microcode-controlled, i.e., driven by a microsequencer with a 1,600-word control store and a 20-word patch capability. Since the control store does not limit the cycle time, we chose to implement a single microcode control scheme, rather than hardwire control for the simple instructions and provide microcode control for the remaining instructions.

The E-box begins instruction execution based on information taken from the instruction queue. References to specifier operands and results are made indirectly through pointers in the source and destination queues. In this way, most E-box instruction flows do not need to know whether operands or results are in register, memory, or instruction stream.

To improve the performance of certain critical instructions, the E-box contains special-purpose hardware. A mask processing unit finds the next bit set in a mask register and is used in the following instructions: FFC, FFS, CALLS, CALLG, RET, PUSHR, and POPR. A population counter provides the number of bits set in a mask and is used in the CALLS, CALLG, PUSHR, and POPR instructions. In addition, microcode can operate the arithmetic logic unit (ALU) and shifter independently to produce two computations per cycle, which can significantly improve the parallel operation of the complex instructions.

In addition to normal instruction processing, the E-box performs all power-up functions and interrupt and exception processing, directs operands to the F-box, and accepts results from the E-box. To guarantee that instructions complete in instruction stream order, the E-box orchestrates result stores and instruction completion between the E-box and E-box.

The F-box

The F-box performs longword (32-bit) integer multiply and floating-point instruction execution. The E-box supplies operands, and the F-box transmits results and status back to the E-box.

The F-box contains a four-stage, floating-point and integer-multiply pipeline, and a nonpipelined, floating-point divider. Subject to operand availability, the F-box can start a single-precision, floating-point operation during every cycle, and a double-precision, floating-point or integer-multiply operation during every other cycle.

Stage 1 of the pipeline calculates operand exponent difference, adds the fraction fields, performs recoding of the multiplier, and computes three times the multiplicand. Stage 2 performs alignment, fraction multiplication, and zero and leading-one detection of the intermediate results. Stage 3 performs normalization, fraction addition, and a miniround operation for floating-point add, subtract, and multiply instructions. Stage 4 performs rounding, exception detection, and condition code evaluation.

Stage 3 performs a miniround operation on the result calculated to that point to determine if a full-round operation is required in Stage 4. To do this, a round operation is performed on only the low-order three (for single-precision) or six (for double-precision) fraction bits of the result. If no carry-out occurs for this operation, the remaining fraction bits are not affected and the full stage 4 round operation is not required. If the full round is not required, stage 4 is dynamically bypassed, resulting in an effective three-stage pipeline.

The M-box

The M-box is responsible for address translation, access checking, and access to the primary instruction and data cache (P-cache). The M-box accepts requests from multiple sources and processes these requests in an order that reflects both the priority of the request and the need to maintain instruction stream ordering of memory reads and writes. Address translation and cache access are fully pipelined; the M-box can start a new request at the beginning of every cycle.

The M-box performs address translation and access checking by means of a 96-entry, fully associative translation buffer (TB) with parity protection. If a TB miss occurs, the M-box automatically invokes a hardware miss sequence that calculates the address of the page table entry (PTE) that maps the page, fetches the PTE from memory, refills the TB, and restarts the reference. TB allocation is performed using a not-last-used scheme, which is similar to a round-robin but guarantees that the most recently referenced entry will not be overwritten. The M-box reports access violations and page faults to the E-box, and E-box microcode processes these misses with hardware support from the M-box.

The M-box also translates memory destination operand addresses provided by the I-box and saves the corresponding physical address in the physical address (PA) queue. When the E-box stores a result, the M-box matches the data with the next address in the PA queue and converts this data to a normal write request. The PA queue is also used to check for conflicts in read requests to a location in which nothing has been written.

The P-cache is an 8KB, two-way set-associative cache with 32-byte blocks and parity protection on tags and data. The P-cache can be configured to cache instructions, data, or both, and usually has the latter configuration. For compatibility with the DECchip 21064 microprocessor, the NVAX+ P-cache can also be configured into a direct-mapped organization.

The NVAX C-box

The NVAX C-box maintains the interface to the external B-cache and to the NDAL bus. The C-box receives read and write requests from the M-box and monitors the NDAL for activity that would require an invalidate operation in either cache. Consecutive writes to the same quadword (64 bits) are merged into a single quadword datum by packing logic placed at the input of an eight-entry quadword write queue.

The C-box can accept one instruction read request and one data read request from the M-box. Conflict logic in the write queue allows nonconflicting read requests to be processed before queued write requests are performed. Conflicts are resolved by processing write queue entries until the conflicting write is completed.

The C-box supports four B-cache sizes: 128KB, 256KB, 512KB, and 2 megabytes (MB). The system designer can independently select tag and data RAM speeds to meet system requirements, regardless of the frequency at which the CPU is running. The B-cache block size is 32 bytes, and both tag and data RAMs are protected with error correction code (ECC) that corrects single-bit errors and detects both double-bit errors and full 4-bit RAM failures. The B-cache implements a directory-based broadcast coherence protocol in conjunction with a memory directory containing one bit per 32-byte block. Each memory directory bit indicates if the associated block is valid in memory or has been written and exists in a cache. Unwritten blocks may exist in multiple caches in the system. Written blocks may exist in exactly one cache.

An attempt to write to a block that is not both valid and already written in the B-cache causes the C-box to request write permission from memory by means of a special NDAL bus read command. The memory controller will not respond to any NDAL bus transactions to a block that is written in a cache. Instead, it waits for the CPU, which contains an updated copy of the block, to write the block back to memory and then completes the original transaction. All CPUs in the system monitor the NDAL bus for read and write transactions and compare the address against their B-cache tags. If a match is found, the cache block is either written back to memory, invalidated, or both, depending on the transaction type and the state of the block in the cache.

The NDAL protocol fully supports multiprocessing implementations and does not require any special chip variants to construct a multiprocessor system. The C-box invokes invalidate or write-back requests as required to keep the B-cache and P-cache coherent with NDAL activity.

The NVAX+ C-box

The NVAX+ C-box provides the interface between the internal functional units and the EDAL pin bus

implemented by the DECchip 21064 microprocessor. This C-box interface includes the basic interface control for the external B-cache and for the memory and I/O system. The NVAX+ C-box receives read and write requests from the M-box. These requests are queued and arbitrated within the C-box and result in cache or system access across the EDAL. The NVAX+ C-box also maintains cache coherency by sending invalidate requests to the M-box when requested by external logic.

The NVAX+ C-box implementation provides many of the same features and performance enhancements available in the NVAX C-box. Included is support for software-programmable B-cache speeds (one-half, one-third, or one-fourth times the CPU frequency) and sizes (128KB to 8MB), write packing, write queuing, and read-write reordering. In addition, the NVAX+ C-box supports the newer platforms and increases the degree to which NVAX+ is compatible with the DECchip 21064 microprocessor. NVAX+ C-box features include programmable system clock speeds, I/O space-mapping, and a direct-mapped option on the P-cache.

A major difference between the NVAX and NVAX+ implementations is in the B-cache coherence protocol. Rather than mandate a fixed B-cache coherence protocol, the NVAX+ implementation allows systems to tailor the protocol to their particular needs. NVAX+ cache coherency is implemented jointly by off-chip system support logic and by the CPU chip, with relevant information passed between the two over the EDAL bus. To allow duplicate cache tag stores (if they exist) to be properly updated, the NVAX+ C-box provides information to off-chip logic, indicating when the internal caches are updated. External logic notifies the NVAX+ C-box when an internal cache entry needs to be invalidated because of external bus activity.

Existing systems configure the B-cache to implement a conditional write-update snoopy protocol carried out using shared and written signals on the system bus. Writes to shared blocks are broadcast to other caches for conditional update in those caches. A CPU that receives a write update checks the NVAX+ P-cache to determine if the block is also present in that cache. If the block is present, the B-cache update is accepted and written into the B-cache, and the P-cache is invalidated. If the data is not present in the P-cache, the B-cache is invalidated. This results in a write-update protocol for data that was recently referenced by a CPU (and hence is valid in the P-cache) and reduces to a write-invalidate protocol for data that was not recently referenced.

To accommodate the programmable nature of both the system and cache clock frequencies, the NVAX+ C-box supports nine different combinations of cache and system clock frequencies. This support allows efficient use of the chip in a wide range of different performance class systems.

Pipeline Operation

The NVAX and NVAX+ chips implement a macropipeline. Multiple VAX macroinstructions are processed in parallel by relatively autonomous functional units with queued interfaces at critical boundaries. Each functional unit also has an internal pipeline (micropipeline) to allow a new operation to start at the beginning of every cycle. The pipeline operation can be logically depicted, as shown in Figure 4.

In pipeline segment S0, instruction stream data is read from the VIC. The next VAX instruction component is parsed, and queue entries are made in segment S1. For short literal and register specifiers, no other processing is required. Requests for further processing for all other specifiers are queued to the CSU pipeline, which reads operand base addresses in segment S2, calculates an effective address, and makes any required M-box request contained in segment S3. If an M-box request is made, address translation and P-cache lookup occur in segments S4 and S5.

Instruction execution starts with an E-box control store lookup in segment S2, followed by a register file read of any required operands in segment S3, an ALU and/or shifter operation in segment S4, and a potential result store or register file write in segment S5. If an M-box request is required, e.g., for a memory store, the request is made in segment S4; address translation or PA queue access occurs in segment S5; and a P-cache access occurs in segment S6.

Floating-point and integer-multiply instruction execution starts in the E-box, which transfers operands to the F-box. The four-stage F-box pipeline is skewed by half a cycle with respect to the E-box pipeline, beginning halfway through segment S4. The fourth segment of the F-box pipeline is conditionally bypassed if a full-round operation is not required. The result is transmitted back to the E-box, logically in segment S5 of the pipeline.

Pipeline bypasses exist for all important cases in the I-box and E-box pipelines, so that there are

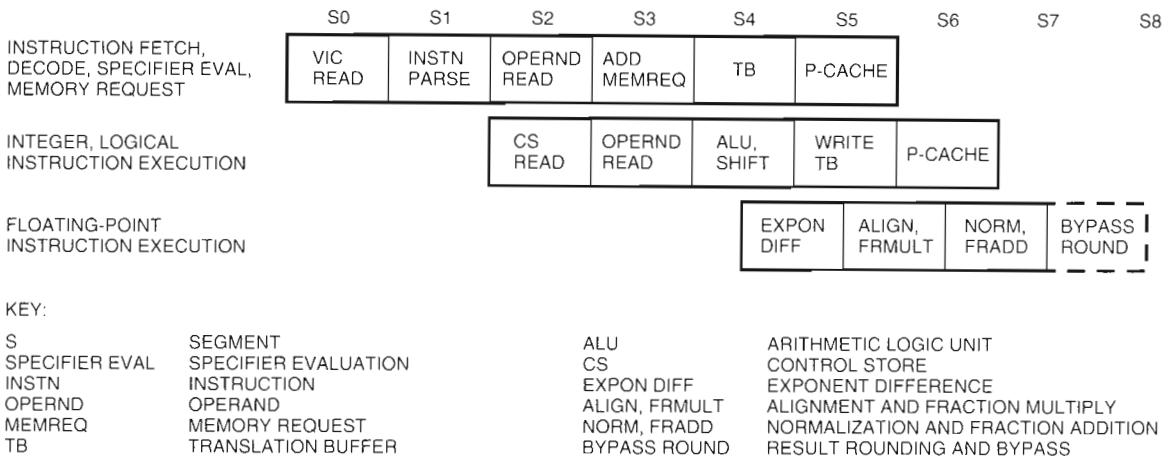


Figure 4 Pipeline Organization

no stalls for results feeding directly into subsequent operands. The M-box processing of memory references initiated as a result of operand specifier processing by the I-box is usually overlapped with the execution of the previous instruction in the E-box, with few or no stalls occurring on P-cache hit.

Design Evolution and Trade-offs

The NVAX and NVAX+ chips are the latest in a line of CMOS VAX microprocessors designed by Digital's engineers and represent a continuing evolution of architectural concepts from one implementation to the next. The preceding chip design was the CPU for the VAX 6000 Model 400 system.[12] To meet the time-to-market and performance goals, we had to modify the NVAX/NVAX+ design throughout the project.

One of the early vehicles for making design trade-offs was the NVAX performance model, which predicts CPU and system performance and aids in quantifying the performance impact of various design options. The performance model is a detailed, trace-driven model which can be easily configured by changing any of a variety of input parameters. The model stimuli used were 15 generic timesharing and 22 benchmark instruction trace files that were captured by running actual programs on existing VAX systems.

The following sections describe the evolution of the chip design, including the number of chips, the pipelining technique used, and various cache issues.

Number of CPU Chips

The VAX 6000 Model 400 core CPU implementation is a three-chip design: a processor chip, with a small on-chip primary cache; a floating-point chip; and a secondary cache controller, with internal cache tags. The initial attempt at NVAX CPU definition was a two-chip design. One chip contained the I-box (with a 4KB VIC), the E-box, the F-box, and the M-box (with a 16KB, direct-mapped P-cache). The second chip held the C-box and the B-cache tag array. The project design goals, especially time-to-market, led to a single-chip solution, rather than a two-chip design.

To condense the design from two chips to one, we halved the sizes of the VIC and the P-cache and moved the B-cache tags to external static RAMs, leaving the B-cache controller on-chip. Later, we were able to reduce the penalty of halving the size of the P-cache by making it two-way set associative rather than direct mapped. With these changes, the performance model showed a performance loss of less than 1.4 percent across all the traces, relative to the two-chip design, with a worst-case penalty of 3.9 percent.

There are strong advantages to the single-chip solution.

- Designing a single chip takes less time.
- This design requires the production and maintenance of only one design database and one mask set.
- Latency to the B-cache is shorter.
- An off-chip tag store provides more flexibility in B-cache configurations.

Macropipelining

Run-time performance is the product of the cycle time, the average time to execute an instruction (cycles per instruction [CPI]) and the number of instructions executed. CMOS process improvements made it possible to decrease the NVAX/NVAX+ cycle time with respect to the previous generation of VAX microprocessors, thus improving the first factor in run-time performance.

The VAX 6000 Model 400 CPU design uses traditional microinstruction pipelining, i.e., micropipelining, to achieve some amount of overlap and to decrease the CPI. However, using micropipelining techniques would not reduce the NVAX/NVAX+ CPI to the level required to meet the performance goals of the NVAX/NVAX+ projects. We achieved this reduction by using RISC design and implementation techniques referred to as macropipelining. In a macropipelined architecture, the I-box acts much like a load/store engine, dynamically prefetching operands prior to instruction execution. Using the macropipeline technique in the NVAX and NVAX+ CPUs makes it possible to retire one basic complex instruction set computer (CISC) macroinstruction per cycle, as in a simple RISC design. Although macropipelining introduced considerable complexity into the NVAX/NVAX+ design, this complexity resulted in a significant performance improvement over a traditional micropipelined design.

Number of Specifiers per Cycle

The NVAX/NVAX+ I-box can parse at most one opcode and one VAX specifier per cycle. The I-box design initially considered was capable of parsing two specifiers per cycle. Although this parsing scheme represented significant complexity and circuit risk, intuitively, it seemed important to quickly retire specifiers in the I-box in order to keep the macropipeline full. However, the performance model predicted a maximum performance improvement of less than two percent on our traces, and we decided to limit complexity and schedule risk by parsing only one specifier per cycle.

F-box Design

The NVAX F-box design is highly leveraged from the VAX 6000 Model 400 F-chip design. Rather than start from scratch, we integrated the existing design onto the NVAX and NVAX+ CPU chips and added a final-stage bypass mechanism. In addition, unlike the original F-chip implementation, the NVAX/NVAX+ control of the F-box allows a fully pipelined operation, which significantly improves floating-point performance over the F-chip design. Although a totally new design would have had shorter floating-point latencies, the combination of a fully pipelined operation and a final-stage bypass allowed us to achieve our performance goal, while meeting our time-to-market goal.

Cache Coherence

Performance studies with the previous generation of VAX microprocessors clearly indicate that system bus write bandwidth limits performance unless an external write-back cache is implemented. In addition, the VAX architecture required that we implement the cache coherence protocol in hardware.

The NVAX implementation uses a directory-based coherence protocol for compatibility with existing and planned target system platforms. The NDAL bus supports multiple outstanding read and write requests, which allows the microprocessor to utilize the capability of the system bus to process these operations in a pipelined fashion. We investigated the possibility of implementing both directory-based and snoopy coherence protocols, but time-to-market considerations and the opportunity to optimize the design for performance in existing system platforms outweighed the desirability of supporting snoopy protocols.

For the NVAX+ implementation, the coherence policy is determined by hardware external to the NVAX+ chip, in the given system. The NVAX+ cache and system interface allows the system environment to implement a variety of coherence protocols. Compatibility with the DECchip 21064 interface definition required limiting NVAX+ to one outstanding external cache miss. However, this limitation is more than offset by the significantly better main memory access times achieved in target systems.

One significant advantage of the NVAX+ scheme is that most policies associated with the external cache are determined by hardware outside the NVAX+ chip (such as the coherence policy), allowing the chip to be used in a wide variety of systems. Implementing the DECchip 21064 interface on NVAX+ greatly reduces the hardware engineering investment required to deliver a VAX CPU and an Alpha AXP CPU in the same system environment.

For both the NVAX and the NVAX+ chips, cache coherence is maintained for the P-cache by keeping

it a subset of the external cache. Externally originated invalidate requests are forwarded to the P-cache only when the block is in the external cache. This minimizes the number of P-cache cycles spent processing invalidate requests. The two-way set-associative P-cache might have been slightly more effective if it were not a subset of the larger direct-mapped external cache. However, this effect is far less significant than the effect of expending a P-cache cycle for every external invalidate event.

Virtual caches almost always have lower latency than physical caches and usually do not require a dedicated translation buffer. The VAX architecture supports the use of a VIC by allowing the cache to be incoherent with respect to the data stream, i.e., not updated with recent writes by the CPU containing the VIC, or by any other CPU. However, some mechanism must be defined to make the VIC coherent with the data stream. In the VAX architecture, the execution of the VAX return from interrupt or exception (REI) instruction performs this function.

We chose to perform a complete flush of the VIC as part of the execution of every REI instruction. Because an REI always follows a process context switch, a flush during an REI removes the process-specific virtual addresses of the previous process and prevents conflict with (potentially identical) virtual addresses for the new process. We could have also chosen to keep the VIC coherent with the data stream and implement per-process qualifiers that would have made per-process virtual addresses unique. However, coherence would have required both an invalidate address and a control path to the VIC, and some form of backmap to resolve virtual address aliases. Per-process qualifiers would have required a VAX architecture change and significant operating system software changes. To reduce project risk, we chose to flush the VIC on every REI instruction.

Cache Hierarchy

The NVAX and NVAX+ chips have three levels of cache hierarchy: the VIC, the P-cache, and the B-cache. The VIC and P-cache are fully pipelined and have minimum latency, which allows instructions to be fetched and processed in parallel at very high rates.

The default P-cache configuration causes VIC misses to be looked up in the P-cache. This lookup process is advantageous since the VIC typically experiences a smaller miss penalty because latency for P-cache hits is roughly one-third that for external cache hits. The disadvantage is that instruction fills can result in a higher P-cache data stream miss rate, because they replace data that is likely to be referenced again. We used the performance model with available traces to determine that looking up VIC misses in the P-cache generally resulted in higher performance. In specific applications, higher performance can be achieved by not looking up instruction references in the P-cache. As a result, we implemented P-cache configuration bits that allow system designers to implement either scheme. By default, NVAX and NVAX+ systems are configured to enable both instruction and data caching in the P-cache, but this may be changed by the console software in certain systems to support prepackaged application systems.

External Cache Size

Both NVAX and NVAX+ support multiple external cache sizes to allow system designers full flexibility in selecting external cache configurations. With existing static RAM technology, smaller external cache configurations are usually faster than larger configurations. Performance modeling indicated that many applications, especially some popular benchmarks, fit entirely in a cache whose size is 512KB or less, resulting in slightly better performance. However, many common applications utilize more memory than will fit in such caches and benefit more from an external cache whose size is 1MB to 4MB, even with the additional latency involved. As a result, our system designs use larger but slightly slower external cache configurations.

Block Size

During the analysis of the previous generation of VAX microprocessors in existing systems, we observed that the 16-byte block size was too small to achieve optimal performance on many applications. As a result, we chose a 32-byte block size for the NVAX and NVAX+ internal caches. This size provides a good balance between fill size and the number of cycles required to do the fill, given 8-byte fill data paths.

For compatibility with installed systems, the size of the NVAX external cache block and the cache fill size is 32 bytes. On NVAX+, the external cache block size may be larger and is 64 bytes in the VAX 7000 Model 600 and VAX 10000 Model 600 systems. Because both systems implement low-latency memory and high-bandwidth buses, the increase in external cache block size results in better performance.

Special Features

The NVAX/NVAX+ design includes several features that supplement core chip functions by providing added value in areas of debug, system maintenance, and systems analysis. Among the features are the patchable control store (PCS) and the performance monitoring hardware.

Patchable Control Store

The base machine microcode is stored in a ROM control store in the E-box. The 1,600-microword capacity of the E-box controls macroinstruction execution and exception handling. The PCS consists of 20 entries that can be configured to replace or supplement the microcode residing in ROM. Each PCS entry contains a content-addressable memory (CAM)/RAM pair that stores the patch microword address and patch microword, respectively. The ROM control store and the PCS are accessed in parallel. Typically, words are fetched from the ROM control store, but if a microword address matches the CAM in one of the PCS entries, then the PCS RAM for that entry supplies the microword, and the ROM output is disabled.

Privileged software controls the loading of the PCS by means of internal processor registers. In system operation, a patch file is normally loaded into the PCS early in the boot procedure, so that any minimal system capable of starting system boot can install patches to the base microcode. This feature presents a way to modify the base NVAX/NVAX+ chip through software; the majority of engineering change orders (ECOs) can be accomplished by releasing new patch files, thus alleviating the need to change the hardware design and retool for the very large-scale integration (VLSI) fabrication.

We booted the VMS operating system within 16 days of receiving first-pass wafers from fabrication, a tribute to a very thorough design verification. However, the continuing rigorous testing on prototype systems revealed several problems with the base microcode and hardware. The PCS mechanism helped to identify, isolate, and work around many of the problems during system debug and thus allowed extensive system testing to continue on first-pass chips.

For example, we used a sequence of PCS patches during system debug to isolate an obscure failure whose symptom was a transfer to virtual address 0. By patching the main microcode exception handling routine to check for this event, we identified the instruction stream sequence that was causing the failure. We refined the patch to place additional checking into various instructions in the sequence. This refinement allowed us to isolate the exact instruction that was causing the transfer to PC 0. With this information, we were then able to reproduce the problem in simulation and correct the second-pass design. Without this diagnostic capability, we probably would have needed weeks or months of additional debug time to isolate the failure.

In addition to using the powerful diagnostic capability of the PCS, we used patches to correct or work around the few functional bugs that remained in the first-pass design. For example, a microcode patch was used to correct a condition code problem caused by a microcode bug during the execution of an integer-multiply instruction. Because the E-box is central to the execution of all instructions, we were also able to use patches to correct hardware problems in other boxes. In one instance, a patch was used to inject a synchronization primitive into the M-box in order to correct an M-box design error. As a result of the simplicity and elegance of this solution, the final second-pass correction was to move the patch into microcode ROM, rather than modify the M-box hardware design.

Performance Monitoring Environment

As computer designs increase in complexity, their dynamic behavior becomes less intuitive. Computer designers rely more and more on empirical performance data to aid in the analysis of system behavior and to provide a basis for making hardware and software design decisions. In addition, multiple levels of logic integration on VLSI chips restrict the collection of this performance data, because many of the interesting events are no longer visible to external instrumentation. The NVAX/NVAX+ chip design includes hardware multiplexers and counters that can be configured to count any of a set of predetermined, internal state changes.

Two 64-bit performance counters are maintained in memory for each CPU in an NVAX/NVAX+ system. The lower 16 bits of each counter are implemented in hardware in the CPU, and at specified points, the quadword counters in memory are updated with the contents of the hardware counters. Privileged software can be used to configure the hardware counters to count any one of a basic set of internal events, such as cache access and hit, TB access and hit, cycle and instruction retire, and cycle and stall. When the 16-bit counters reach a half-full state, the performance monitor requests

an interrupt. The interrupt is serviced in a normal way, i.e., between instructions (or in the middle of interruptible instructions) and at an architecturally specified interrupt priority level. Unlike other interrupts, the performance monitor logic interrupt is serviced entirely in microcode and then dismissed; no software interrupt handler is required.

The microcode component updates the counters in memory when it services the performance monitor interrupt. During a counter update, the microcode temporarily disables the counters, reads and clears the hardware counters, updates the counters in memory, enables the counters, and resumes instruction execution. The base address of the counters in memory is taken from a system vector table and offset by the specific CPU number, creating a data structure in memory that contains a pair of 64-bit counters for each CPU.

Combining the use of hardware, software, and the PCS created a versatile performance monitoring environment—one that goes beyond the scope of the basic hardware capabilities. In this environment, we can correlate the counts with higher-level system events and change the representation of the collected data. For example, microcode can enable the counters every time a process context is loaded and disable the counters when a process context is saved. This feature allows us to set up workloads and gather dynamic statistics on a per-process basis. We can also use PCS patches to modify the memory counter address in order to provide an additional offset based on one of the five VAX processor operating modes: interrupt, kernel, executive, supervisor, or user. This technique provides a new performance counter data structure that collects statistics on a per-mode, per-process, per-CPU basis. Also, microcode patches can be used to add context checks that filter and count various events. For example, we can patch the VAX context switch instruction to count context switches or patch the interlocked instructions to count the number and types of accesses to multiprocessor synchronization locks.

The performance monitoring environment is a powerful tool that we have used to collect the data required to analyze hardware and software behavior and interactions, and to develop an understanding of system performance. We have applied this knowledge to tune the performance of operating systems and application software, and continue to apply the knowledge to improve the design and performance of future hardware and software.

Results and Conclusions

With a focus on time-to-market, we shortened the originally projected NVAX design schedule, from the start of implementation to the completion of the chip design, by 27 percent. We booted the operating system just 16 days after prototype wafers became available. The use of the PCS allowed us to quickly debug and work around the few functional bugs that remained in the first-pass design. Because of the quality achieved in first-pass chips, we were able to shorten the schedule from chip design completion to system product delivery. As a result, systems were delivered to customers four months earlier than the originally projected date.

At the same time, we were able to dramatically improve CPU performance relative to previous VAX microprocessors by implementing a macropipelined design, in which multiple autonomous functional units cooperate to execute VAX instructions. Our internal goal was performance in excess of 25 times the performance of the VAX-11/780 system. We significantly exceeded this goal as demonstrated by the following Standard Performance Evaluation Cooperative (SPEC) Release 1.2 performance ratings:[13]

SPECmark	40.5
SPECfp	48.8
SPECint	30.4

These ratings were measured on a VAX 6000 Model 600 system at the initial announcement and are two to three times higher than those for the previous VAX microprocessor running in the same system. Software and system tuning has subsequently improved the initial numbers on all systems.

The NVAX/NVAX+ design provides an upgrade path and system investment protection to customers with installed VAX systems, as well as a migration path from an NVAX+ microprocessor to a DECchip 21064 microprocessor in the new Alpha AXP systems.

Acknowledgments

We would like to acknowledge the contributions of the following people:

W. Anderson, E. Arnold, D. Asher, R. Badeau, I. Bahar, M. Benoit, B. Benschneider, D. Bhavsar, M. Blaskovich, P. Boucher, W. Bowhill, L. Briggs, S. Butler, R. Calcagni, S. Carroll, M. Case, R. Castelino, A. Cave, S. Chopra, M. Coiley, E. Cooper, R. Cvijetic, R. Davies, M. Delaney, D. Deverell,

A. DiPace, C. Dobriansky, D. Donchin, R. Dutta, D. DuVarney, J. J. Ellis, J. P. Ellis, H. Fair, B. Feaster, T. Fischer, T. Fox, G. Franceschi, N. Geagan, S. Goel, M. Gowan, J. Grodstein, P. Gronowski, W. Grundmann, H. Harkness, W. Herrick, R. Hicks, C. Holub, J. Huber, A. Jain, K. Jennison, J. Jensen, M. Kantrowitz, R. Khanna, R. Kiser, D. Koslow, D. Kravitz, K. Ladd, S. Levitin, J. Licklider, J. Lundberg, R. Marcello, S. Martin, T. McDermott, K. McFadden, B. McGee, J. Meyer, M. Minardi, D. Miner, E. Nangia, L. Noack, T. O'Brien, J. Pan, H. Partovi, N. Patwa, V. Peng, N. Phillips, R. Preston, R. Razdan, J. St. Laurent, S. Samudrala, B. Shah, S. Sherman, W. Sherwood, J. Siegel, K. Siegel, C. Somanathan, C. Stolicny, P. Stropparo, R. Supnik, M. Tareila, S. Thierauf, N. Wade, S. Watkins, W. Wheeler, G. Wolrich, and Y. Yen.

References

1. R. Brunner, ed., *VAX Architecture Reference Manual,* Second Edition, (Bedford, MA: Digital Press, 1991).

2. D. Dobberpuhl et al., "A 200MHz 64b Dual-Issue CMOS Microprocessor," *IEEE International Solid-State Circuits Conference, Digest of Technical Papers,* vol. 35 (1992): 106–107.

3. R. Sites, ed., *Alpha Architecture Reference Manual,* (Burlington, MA: Digital Press, 1992).

4. D. Fite, Jr. et al., "Design Strategy for the VAX 9000 System," *Digital Technical Journal,* vol. 2, no. 4 (Fall 1990): 13–24.

5. W. Anderson, "Logical Verification of the NVAX CPU Chip Design," *Digital Technical Journal,* vol. 4, no. 3 (Summer 1992, this issue): 38–46.

6. M. Callander et al., "The VAXstation 4000 Model 90," *Digital Technical Journal,* vol. 4, no. 3 (Summer 1992, this issue): 82–91.

7. J. Crowell and D. Maruska, "The Design of the VAX 4000 Model 100 and MicroVAX 3100 Model 90 Desktop Systems," *Digital Technical Journal,* vol. 4, no. 3 (Summer 1992, this issue): 73–81.

8. J. Crowell et al., "Design of the VAX 4000 Model 400, 500, and 600 Systems," *Digital Technical Journal,* vol. 4, no. 3 (Summer 1992, this issue): 60–72.

9. L. Chisvin, G. Bouchard, and T. Wenners, "The VAX 6000 Model 600 Processor," *Digital Technical Journal,* vol. 4, no. 3 (Summer 1992, this issue): 47–59.

10. A. Agarwal et al., "An Evaluation of Directory Schemes for Cache Coherence," *Proceedings of the 15th Annual International Symposium on Computer Architecture* (May 1988): 280–289.

11. P. Stenstrom, "A Survey of Cache Coherence Schemes for Multiprocessors," *Computer* (June 1990): 12–24.

12. H. Durdan et al., "An Overview of the VAX 6000 Model 400 Chip Set," *Digital Technical Journal,* vol. 2, no. 2 (Spring 1990): 36–51.

13. *VAX 6000 Datacenter Systems Performance Report* (Maynard, MA: Digital Equipment Corporation, 1991).

Dale R. Donchin
Timothy C. Fischer
Thomas F. Fox
Victor Peng
Ronald P. Preston
William R. Wheeler

The NVAX CPU Chip: Design Challenges, Methods, and CAD Tools

The NVAX CPU chip is a 1.3 million transistor, VAX microprocessor designed in Digital's 0.75-micrometer CMOS-4 technology. It has a typical cycle time of 12 ns under worst-case operating conditions. The goal of the chip design team was to design a high-performance, robust, and reliable chip, within the constraints of a short schedule. Design strategies were developed to achieve this goal, including organization of a chip design flow and new implementation and verification methods. New proprietary CAD tools also played important roles in the chip development.

The NVAX CPU chip is a 1.3 million transistor, VAX microprocessor designed in Digital's 0.75-micrometer fourth-generation complementary metal-oxide semiconductor (CMOS-4) technology. The implementation of the NVAX CPU chip posed many design and complexity management challenges. The combination of the chip performance goal, the high level of integration, and the small feature sizes of the CMOS-4 technology increased the severity of on-chip electrical issues and the difficulty of verifying the physical design. These challenges were intensified by a short design schedule. This paper describes some of the design strategies, methods, techniques, and proprietary computer-aided design (CAD) tools used by the chip design team, which were instrumental in meeting these challenges.

Chip Overview

In order to appreciate the design challenges that were faced, it is necessary to understand the size and complexity of the design. The NVAX CPU chip has a macropipelined microarchitecture and implements the VAX instruction set.[1] Because it is a complex instruction set computer (CISC) architecture, the VAX architecture implements varable length instructions with complex addressing modes, and precise traps and exceptions. The chip is composed of five subchips, or functional boxes, called the I-box, E-box, F-box, M-box, and C-box.

The I-box fetches, parses, and decodes instructions, and predicts conditional branches. The E-box runs under microprogrammed control and executes all instructions, except floating-point and long-word integer multiply instructions, which are executed by the F-box. The M-box processes memory references, including virtual-to-physical address translation. The C-box controls the off-chip backup cache (the second-level cache for data and third-level cache for instructions) and contains the bus interface unit. The chip also has a direct-mapped 2 kilobyte (KB) virtual instruction cache (VIC), a two-way, set-associative 8KB data and instruction primary cache (P-cache), a 12KB control store read-only memory (ROM), a 96-entry, fully associative translation buffer, and a 512-bit by 4-bit conditional branch history random-access memory (RAM) for branch prediction. The chip photomicrograph, with box and large array boundaries outlined, is shown in Figure 1.

The NVAX chip's layout database is composed of over 4,200 unique custom cells, and has a total transistor count of 1.3 million. It was the first product chip to be implemented in Digital's 0.75-micrometer, three metal layer, 3.3-volt (V) CMOS-4 technology.[2] The chip's typical cycle time under worst-case operating conditions is 12 nanoseconds (ns) or 83.3 megahertz (MHz), and it dissipates 14 watts (W). A summary of the chip features is given in Table 1.

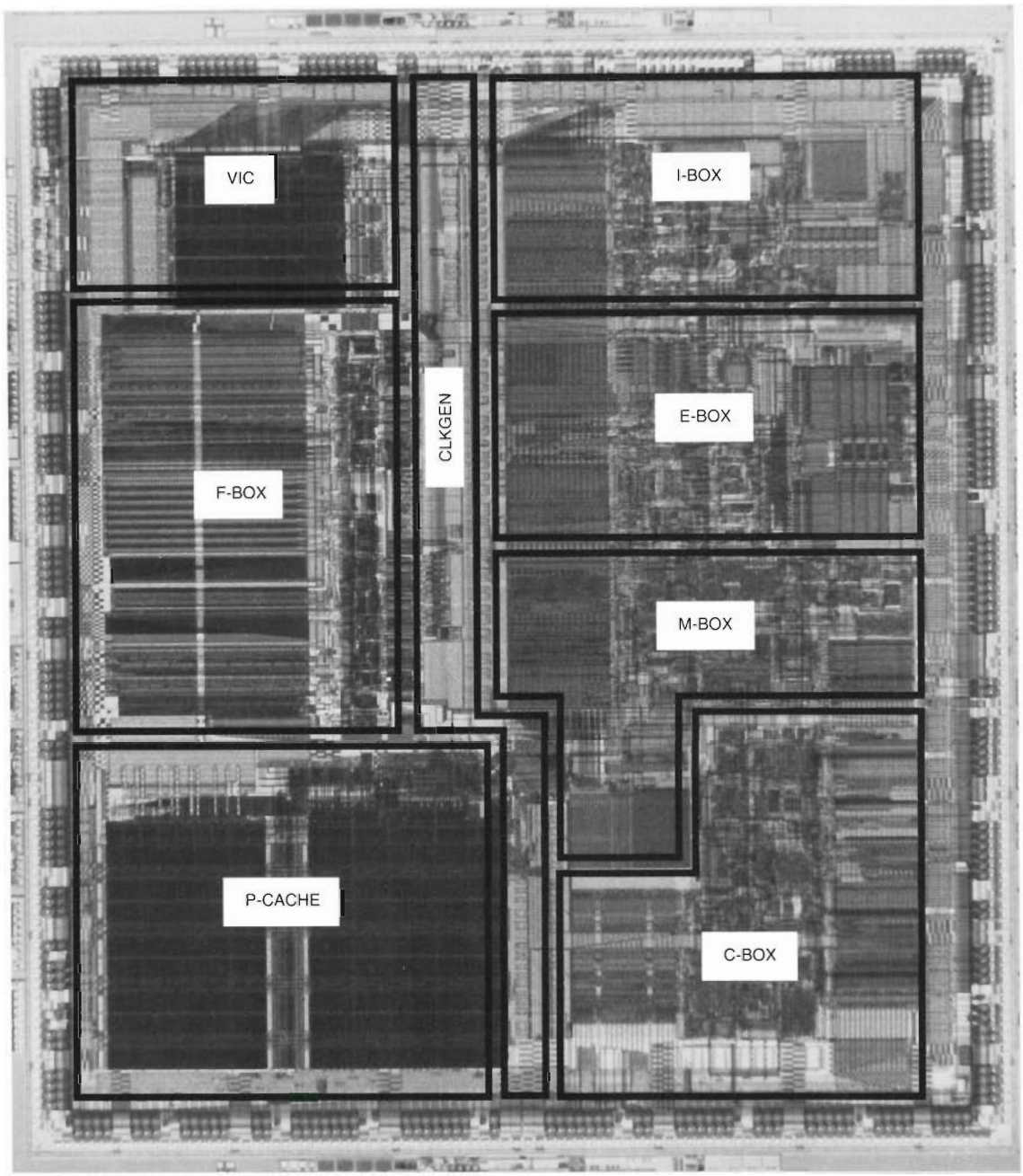


Figure 1 NVAX Chip with Boxes and Large Arrays Outlined

Design Goals and Constraints

Our design goal was to develop a high-performance chip that is electrically robust and reliable. We had to accomplish this within the CMOS-4 process constraints and the design time allotted by the development schedule. Our initial implementation schedule was based on scheduling metrics from previous designs. Due to competitive marketing pressures, this schedule was substantially reduced, making it significantly more aggressive than for previous designs. Our cycle time was constrained to a maximum of 14 ns under worst-case conditions. Electrical reliability had to be guaranteed for cycle times down to 10 ns under worst-case conditions.

Table 1 Summary of Chip Features

Transistors	1.3 million
Die size	16.2 mm by 14.6 mm
Cycle time	12 ns (typical)
Signal pads	261
Supply pads	121
Package	339 pin THPGA*
Power dissipation at 12 ns	14 W (average)

Note: *Through-hole pin grid array

Based on CMOS-4 process limits, the chip die size was constrained by a maximum diagonal length of 21.8 millimeters (mm).

The trade-offs between design time and chip characteristics, such as performance, area, and function, were the dominant issues throughout the project.

Design Strategy and Challenges

To achieve the goal of a 14-ns cycle time, we designed custom circuitry and layouts, including dynamic, asynchronous, and differential logic. To deal with the size and complexity of the chip, together with the schedule constraint, our strategy called for a large design team. Complex, custom very large-scale integration (VLSI) chip designs inherently have high levels of design schedule risk. To reduce our exposure to schedule slips, we made several high-level design decisions early in the project. Wherever possible, we avoided using circuit structures that were time-consuming to analyze. We defined and followed detailed design guidelines to ensure design consistency, robustness, and reliability. We used a top-down design flow and made extensive use of proprietary CAD tools that were expressly developed for high-performance custom VLSI design. Lastly, we minimized chip area by handcrafting layout in sections of the chip where the leverage on reducing overall die size was significant; or when critical path node had to be minimized to satisfy the path timing constraints.

The floor plan was accurately monitored throughout the project. This was essential because initial die estimates indicated that the chip size was close to the maximum size that the CMOS-4 technology could support. These strategic decisions reduced design time and allowed us to focus on achieving a fast CPU cycle time without compromising the quality of the design.

In addition to minimizing risks to the schedule, we had to solve several global design and verification problems to achieve the cycle time of 14 ns. The design team assumed a 10-ns cycle time when it addressed problems that are exacerbated by a faster cycle time, such as interconnect reliability and signal integrity. Some of the key concerns were on-chip power, ground, and low skew clock distribution, and the routing of long signal interconnects. Verification of the custom layout was another challenge, particularly given the schedule constraints. The use of CAD tools was a significant benefit in development of the chip. These issues are addressed in more detail in the remaining sections of this paper.

Design Flow and Management

The chip design team was organized into five semiautonomous groups, each of which focused on the design of a functional unit (C-box, E-box, F-box, I-box, M-box). To ensure design compatibility and consistency across the chip, each team adhered to the same design guidelines and methods. For example, box-level interfaces were rigorously defined, a consistent register transfer level (RTL) modeling style was used, and circuit noise margins, transistor sizing, and other circuit and layout guidelines were observed. The design team followed the top-down, hierarchical design flow depicted in Figure 2, but there was much overlap between the activities. Complexity was managed by thoroughly reviewing and testing the design at each level of abstraction (microarchitecture performance model, RTL model, logic, circuit, and layout), and by using CAD tools to verify that all the design representations were consistent across the levels of abstraction. When making design decisions, we considered the implications across the levels of abstraction. For example, when we made microarchitectural trade-offs, we considered the implications for power consumption, logic complexity, circuit speed and cycle time, layout, die size and, of course, schedule.

Performance Model and Microarchitecture Specification

The NVAX performance model is a software program that models the microarchitecture of the NVAX chip. The architecture team used the model to study the effect of various microarchitectural factors on overall CPU performance and to define the chip's microarchitecture. The performance model was updated as the microarchitecture

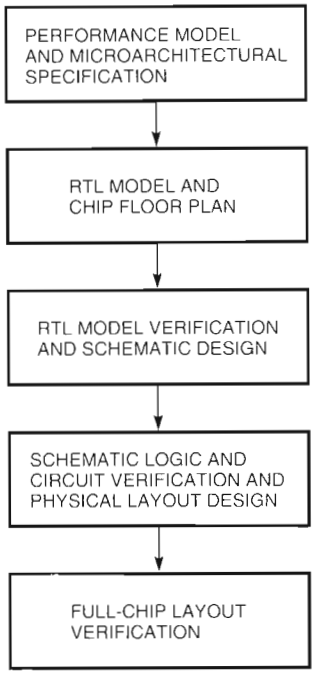


Figure 2 NVAX Design Flow

evolved so that the team could assess the impact of design changes on performance.

The chip microarchitects wrote a textual specification of the chip that documented its function and microarchitecture. As the details of the design were resolved, the specification was updated and expanded to reflect the actual implementation. The functional design verification team used the specification to develop implementation-specific tests.

RTL Model and Floor Plans

The design team developed a detailed RTL model of the chip once the chip specification was completed. This model was written in Digital's proprietary BDS hardware description language. As the BDS code was being written, many logic/circuit feasibility studies were spawned. The model was used to verify that the proposed microarchitecture executed VAX code correctly. It also served to guide logic implementation and was used by system design teams to develop modules based on the NVAX microprocessor.[3,4] The RTL model was updated as the project progressed.

The chip floor plan was devised early in the design to estimate and track the die size and to provide area-impact data for microarchitectural trade-offs. Once the chip-level floor plan was stable, the box design teams developed more detailed box-level floor plans. All floor plans were entered directly into the layout database and maintained throughout the project. Tracking the floor plan at several levels eased the difficulty of integrating the box layouts to form the full-chip composite late in the project. More details on floor plans and the use of CAD tools are described in the section Floor Plan Techniques.

RTL Verification and Schematic Design

We verified the RTL model by running a combination of pseudorandom tests, standard VAX architecture tests, and handwritten implementation-specific tests. In order to identify flaws before time-consuming schematic and layout changes were implemented, we ran regression tests on the model whenever we made changes to the design. To track design changes and issues, designers posted a description of the changes and issues along with the ramifications for other parts of the design in an electronic bulletin board. Each new entry to the bulletin board was mailed electronically to every member of the team. This tracking procedure helped reduce design iterations caused by stale information.

We used the RTL model as a specification for logic/circuit design. To synthesize logic directly from the RTL model for circuits with less critical area and speed constraints, we used a CAD tool, OCCAM. Because these constraints were stringent for a large portion of the chip, engineers designed most of the circuits. We developed a library of common circuit and layout structures to reduce the design and verification effort. We defined and followed detailed design guidelines to ensure design consistency. More information on the types of circuits used on the NVAX chip is being published in "A 100 MHz Macropipelined VAX CMOS Microprocessor."[5]

Schematic Verification and Layout Design

We held design reviews and used CAD tools to check schematics for unintended deviations from the design guidelines. We performed extensive logic simulation on the schematics. We used SPICE for accurate critical path timing analyses, and a static circuit timing analyzer, NTV, to detect unidentified slow paths.[6] (NVAX timing verification is described in a later section.) Figure 3 is a histogram of slow paths as a function of cycle time that was generated by NTV about two months before we taped out the first pass of the chip. Because NTV does not predict circuit delays as accurately as

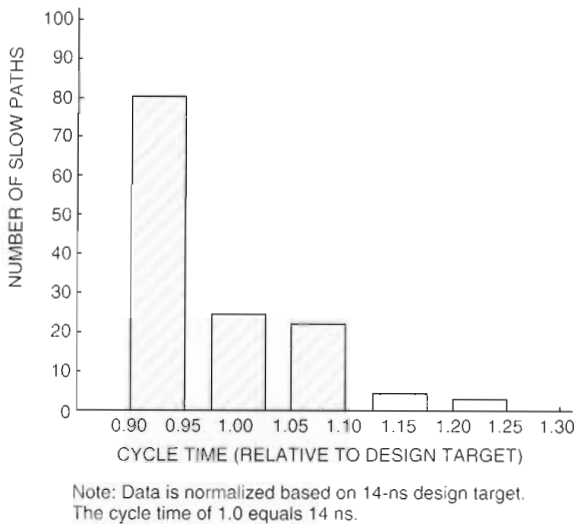


Figure 3 Early Full-chip NTV Histogram

SPICE, all questionable paths flagged by NTV were simulated using SPICE. Those that were found to be slow were redesigned to meet the target cycle time.

Logic and circuit changes resulting from these analyses and the impact of these changes on other design representations and verification tests were tracked on the electronic bulletin board. Since many people were working on the design simultaneously, detailed tracking of changes and open issues proved crucial to meeting our schedule. A single change often required modification and reverification of the RTL model, schematics, layout, verification tests, and in some cases the chip textual specification.

Layout design proceeded on a subbox, or section, basis. A typical section consisted of approximately ten related schematics. Each section was checked for connectivity and correctness after its layout was completed. Sections within a box were then integrated and verified (boxes contained from 5 to 15 sections) before the complete chip composite was assembled.

Schematic verification and layout design were performed concurrently during much of the project. Although this overlap led to design changes that increased the amount of layout rework, the chip development schedule would have been significantly longer if schematic verification and layout design had been done serially. Layout design took about a year to complete.

Layout Verification

Section- and box-level layout verification was performed in parallel with the layout design. Once layout design was completed, the final stages of layout verification ensured that the assembled chip layout met reliability and integrity guidelines for global nodes such as power, clocks, and signals. Most of the layout checks were performed by CAD tools that used the CMOS-4 layout design rules and NVAX-specific electrical rules. More details on layout verification are given in the section Layout Verification Tools.

Floor Plan Techniques

With 1.3 million transistors, nearly half a million signal nodes, and over 16 million polygons on the NVAX die, precise monitoring of the floor plan was critical. From the earliest area estimates, it was clear that the chip size was close to the maximum that the CMOS-4 technology could support. We had to ensure that the die did not exceed the technology limit.

Area Estimation and Placement

Preliminary estimates of the die area were made by extrapolations from previous CPU designs.[7,8] Better area estimates were developed for regular structures, such as the cache arrays and data paths, by creating test layout structures. More accurate floor plans of the control sections of the chip were developed after the RTL model was written. In most cases, the partitioning of the RTL model was a close approximation to the desired schematic and layout partitioning. To estimate the area of random control logic, we used the OCCAM logic synthesis tool, and in many cases Digital's proprietary layout synthesis tool, CLEO.

As seen in the chip photomicrograph in Figure 1, the clock generator and drivers (CLKGEN) were distributed in a narrow north-south channel near the center of the chip. That location was chosen to minimize clock skew. The I-box, E-box, M-box, and C-box subchips are located on the right side of the chip. Their relative positions were chosen to accommodate the flow of the pipelined data in the main data path, which runs north-south just to the right of CLKGEN. The VIC, F-box, and P-cache were placed on the left-hand side of the chip, adjacent to the boxes with which they communicate.

Interconnect Planning and Routing

After we determined the initial placements of all major control sections, we used a new two-level block router called GLOW to route the layout for the interbox and intrabox signals. Routing was performed at the same time schematics for the control

areas were in design. Since layout did not exist for the control blocks, GLOW was allowed to route signals over the blocks with some restrictions, as well as route signals in channels. We influenced how GLOW routed signals by specifying some areas of blocks as opaque (no over-the-block routing permissible) and some as porous (over-the-block routing is permissible if channels are full). Typically, cell blocks that contained regular arrays (such as programmable logic arrays) or critical circuitry were identified as opaque, whereas most random control areas were identified as porous.

The capacitance values of the interbox and intrabox signal interconnects were extracted from the layout and used in circuit simulations of critical paths. In some cases, the placement of sections and the signal routing were altered to reduce interconnect capacitance on critical signals. The use of synthesis tools such as GLOW, OCCAM, and CLEO allowed a much more detailed floor plan to be developed than was typical for prior designs.[7,8] The ability to feed capacitance information from floor plan routing back into SPICE simulations proved invaluable.

Third Metal Layer

The top aluminum interconnect layer (M3) in the CMOS-4 process was specifically designed to meet the electrical requirements of the NVAX chip. The third metal layer was designed for a low sheet resistivity and high current capacity in order to handle the electrical problems associated with the power grid, clock distribution, critical signal routing, and large array design.

Power and Ground Distribution

When the NVAX microprocessor is run at maximum speed, it draws a direct current of about 5 amperes. Due to CMOS switching transients, the alternating current peaks are considerably higher. Distributing power (V_{dd}) and ground (V_{ss}) across the chip while keeping power grid voltage drops (IR) under 300 millivolts (10 percent of minimum V_{dd}) was a major challenge. To address this constraint and meet interconnect reliability goals, we used the low-resistance M3 layer extensively to distribute V_{dd} and V_{ss}. As shown in Figure 4, we designed the right-hand side of the chip to be covered with an interdigitated array of alternating V_{dd} and V_{ss}

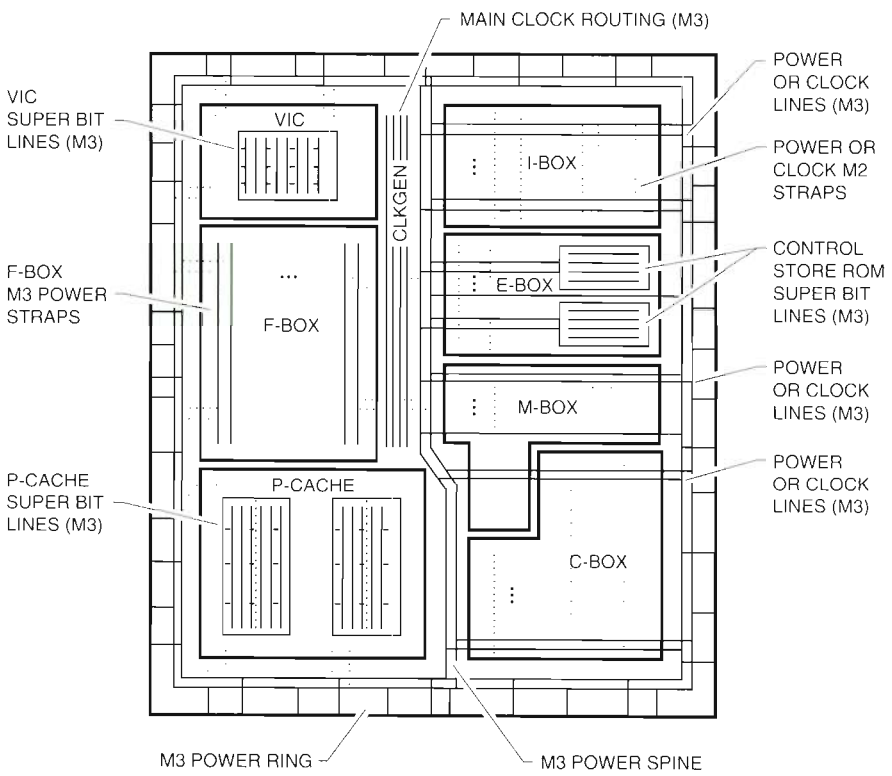


Figure 4 Metal 3 Routing

lines, each 17 micrometers wide. Vertical metal two (M2) lines are used to strap the power lines and form a V_{dd} grid and a V_{ss} grid. The V_{dd} and V_{ss} distribution of the left-hand side of the chip was different from that on the right because of the special layout requirements of the cache arrays and the F-box.

Individual cell layout did not contain M3. The power, ground, and clock connections for a cell were routed by short vertical M2 lines inside each cell. These M2 lines were connected to the M3 grids automatically by a CAD tool.

On-chip Clock Distribution

In order for us to meet the performance goals, it was critical to keep clock skews small and edge rates sharp across the chip. As shown in Figure 5, special attention was given to the clock distribution scheme. Differential outputs from an off-chip oscillator were supplied to a receiver located at the top of the chip. The output of the receiver was routed to the global clock generator (CLKGEN), which was placed at the center of the chip to reduce clock skew. The outputs of the global clock generator were buffered by four inverters to

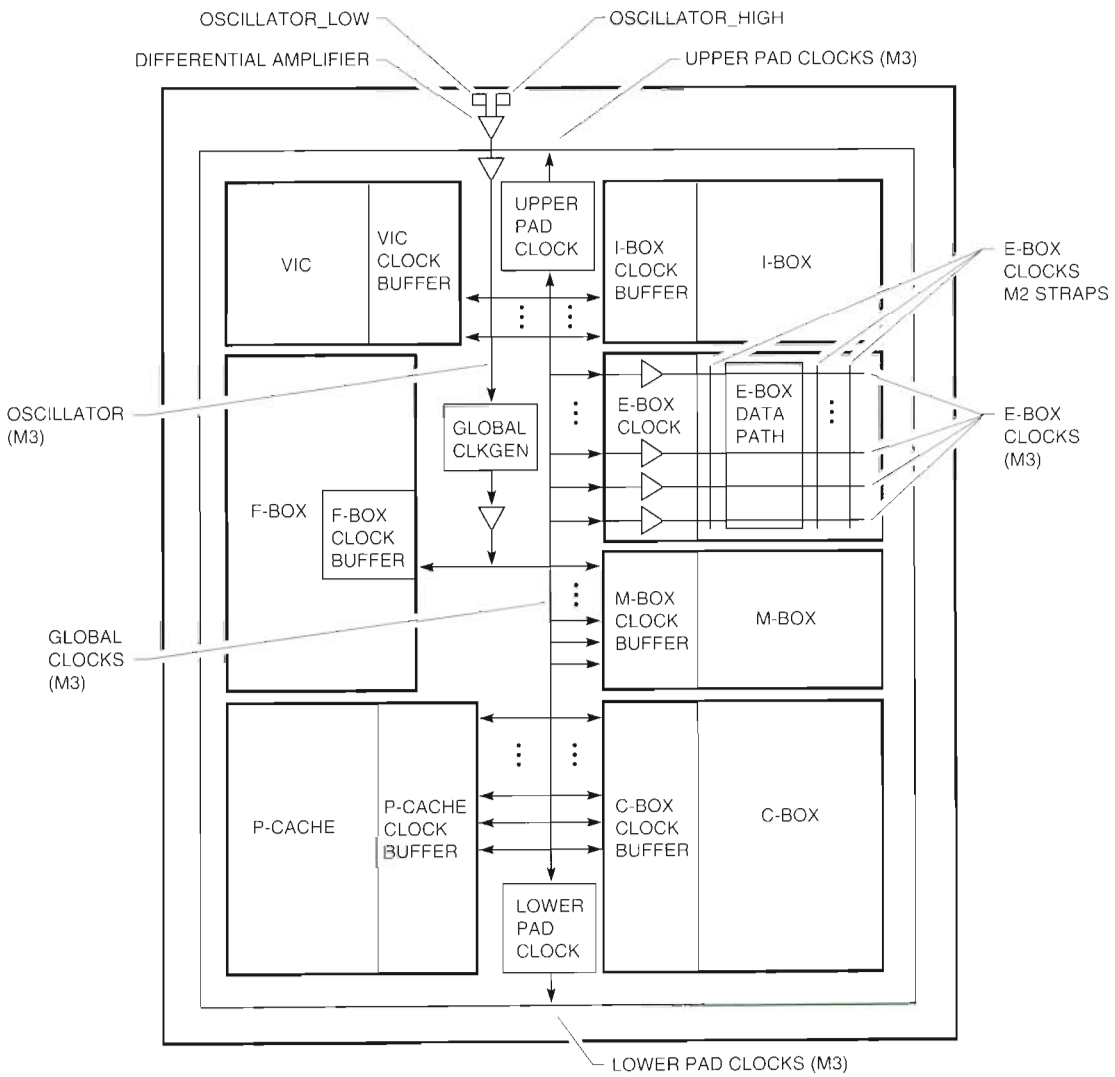


Figure 5 Clock Generation and Distribution

increase their driving capability. The clocks were then distributed, using the low-resistance third metal layer (17 milliohms per square), from the top to the bottom of the central clock routing channel that spans the chip.

Clocks were supplied to the different functional boxes by locally tapping off the central clock routing and buffering each signal with four inverters to further increase the signal's driving capability. This buffering helps to minimize the capacitive loads seen by the clock phases in the central routing channel in which the RC delays are held to 30 picoseconds (ps). To reduce distribution skew between the global clock lines, loading on each global line was balanced by adding dummy loads to the more lightly loaded lines. The buffered clock phases were distributed to the east and west of the central clock routing channel, again using M3 to reduce RC delay. The east-west clock routing was strapped with M2 as shown in Figure 5. These straps were not allowed to cross box boundaries. Box-level clock skew was reduced by using a common section buffer design and layout, and by carefully tuning the buffer drive capability to the clock load in each section.

Finally, before the clocks were used by the logic, the clock signals were locally buffered. These final stages of local buffering served two purposes: they reduced the gate loading on the east-west clock routing, and they helped to sharpen the clock edges seen by the logic.

The global clock routing network was spaced so that the RC delays of local clock branches would never exceed a negligible 125 ps. We calculated the RC delays of local clock branches using the WAWOTH layout interconnect analyzer (described in the section New Proprietary CAD Tools) and, where necessary, rerouted branches to meet the 125-ps design goal. A sample RC plot, generated by WAWOTH for a section of local clock routing, is given in Figure 6. The clock skews and edge rates across this 1.62-centimeter chip are less than 0.5 ns and 0.65 ns, respectively.

Microcode Control Store

The design of the 12KB ROM control store was simplified by dividing it into four subarrays. Each subarray has its own M1 bit lines. The M1 bit lines from the subarrays are cascaded onto low-capacitance M3 super bit lines that extend over all four subarrays. Since the capacitance of the M3 super bit lines is low, the access time is very fast, obviating the need for sense amplifiers and voltage reference generators. This substantially reduced the time required to design and verify the control store ROM.

Primary Cache

A similar technique was used in the 8KB P-cache to ease the timing requirements. The three high-order P-cache address bits must be translated and consequently become valid later than the untranslated lower-order bits. By dividing the P-cache into eight subarrays, each with its own sense amplifiers, the cache subarray access can be started before the three translated address bits are valid. When the last three address bits become valid, the outputs of the subarrays are multiplexed onto the M3 super bit lines, resulting in a faster cache access time.

Layout Verification Tools

Verifying the NVAX chip layout presented several CAD software challenges. Prior to the NVAX design, the existing layout verification tools were able to extract full-chip netlists from layout for all large designs in a single batch process. However, the existing layout netlist extractor could not handle designs such as NVAX with over one million transistors. Also, a more accurate capacitance extraction algorithm was required to calculate side-to-side and fringing capacitance, which came to show significant effects in the small physical dimensions in CMOS-4. Furthermore, accurate interconnect resistance extraction was needed for NVAX. A combination of new CAD tools (see Figure 7) and design methods was employed to meet the NVAX layout verification requirements.

Partitioning Using "Clean Belts"

To address the problem of extracting parasitic capacitance data from such a large design, the NVAX chip layout was constructed so that each chip partition could be independently extracted without introducing inaccuracies in the results. The chip was partitioned into nonoverlapping regions, each of which had a rectilinear annulus or "clean belt" around its periphery. A clean belt is a rectangular region that contains only metal lines and satisfies a number of layout design rules beyond those set by the technology. The clean belt layout rules prevented design rule violations within the clean belt and between adjacent clean belts. The rules also ensured that extracting parasitic capacitance from a region enclosed by a clean belt could be done

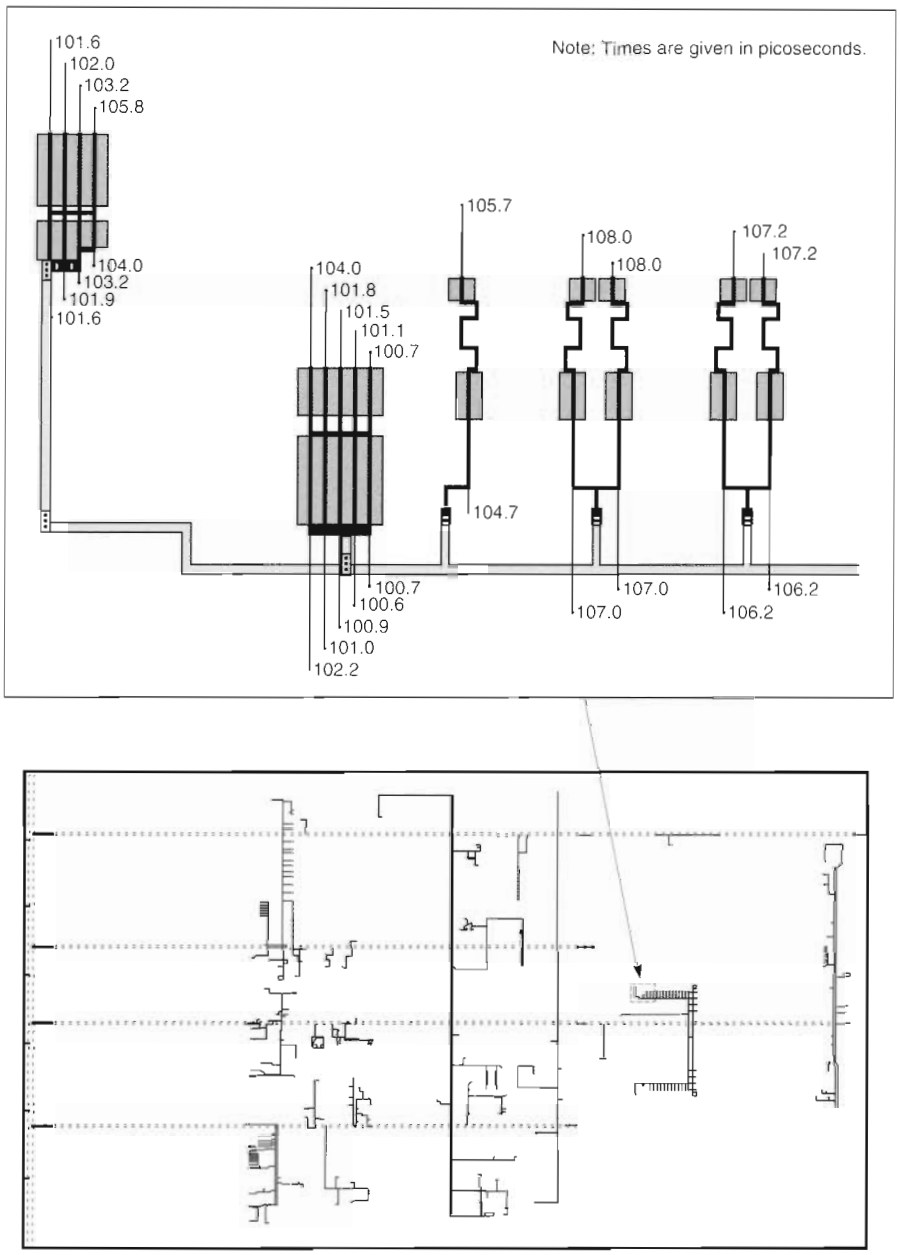


Figure 6 WAWOTH RC Delay Analysis Results for a Clock Node

accurately regardless of the materials that border the region. Partitioning the chip in this manner made it easier to locate global wiring errors.

Hierarchical Netlist Extraction

A new netlist extractor, HILEX, was used to meet the high data capacity requirements of the NVAX microprocessor. HILEX is more efficient than the previous in-house netlist extractor because it recognizes layout cell instances and in many cases needs to extract cell-only definitions. In contrast, the previous netlist extractor "flattened" the layout data into one nonhierarchical cell and, therefore, extracted all data without reusing previously extracted cell definitions. The actual performance improvement realized by HILEX depends on the hierarchy of the chip layout design. If very few cells are replicated, or cells are replicated in a way that requires the

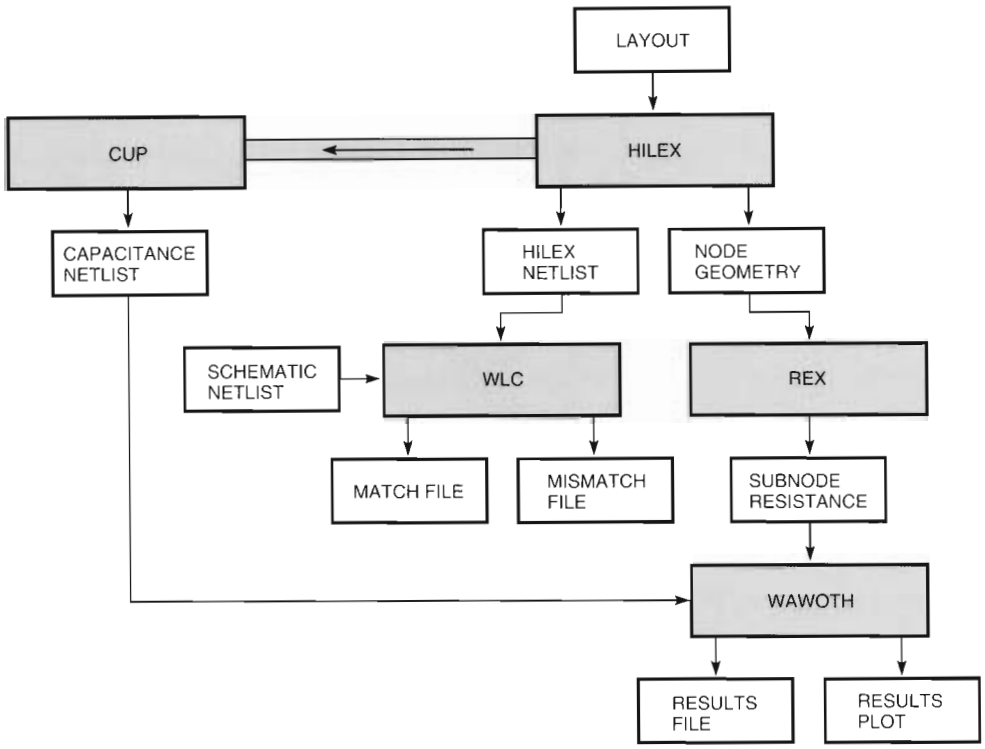


Figure 7 Simplified Layout Verification Tool Flow

extractor to explode the cells (i.e., create more flat data) to extract them properly, then minimal performance improvements are seen. An example of the latter situation is an array of a repeated pair of overlapping cells that forms one or more transistors due to the overlap (one cell contains diffusion areas that become source and drain regions when overlapped with the other cell, which contains polysilicon lines that become transistor gates).

Several layout design guidelines were defined to ensure that performance improvements from using HILEX would be realized. Adherence to the guidelines minimized situations that require HILEX to explode cells and encouraged the use of hierarchy in the layout. However, since it was not always possible to adhere to these design recommendations, HILEX was designed to handle large amounts of flat data.

The chip netlist was extracted from the complete chip layout prior to tape out. This presented quite a challenge since even with the use of HILEX, extracting the chip netlist from the 225MB chip layout file in one pass required more than the maximum of two million virtual pages of memory allowed by the VMS operating system architecture. To go beyond the VMS virtual memory limit, the internal memory management routines within HILEX were modified to allocate additional heap from the process stack (in P1 space) when the VMS memory allocation routines indicated that P0 memory space was exhausted. This technique was used to allocate the 2.5 million virtual pages required for full-chip connectivity extraction.

Netlist Comparison

A utility called WLC was used to verify that netlists extracted from layout by HILEX matched netlists generated from the chip schematics. Since the NVAX schematic hierarchy rigorously matched the layout hierarchy only at certain levels, the connectivity comparison was performed flat. WLC employed a graph-building and graph-traversing algorithm that worked well for comparing less than 500,000 device connections. However, substantial paging occurred when comparing larger netlists. Since the NVAX chip contained 1.3 million devices, the performance of WLC was inadequate for full-chip netlist verification.

To improve the elapsed time of netlist comparison batch jobs, multiprocessing was employed. HILEX was modified to write the extracted netlist of the clean belt partitions. Each partition was then compared by WLC, in parallel on multiple CPUs, to its equivalent schematic-generated netlist. This approach reduced the total elapsed time for the NVAX chip netlist comparison from more than three days to seven hours. Cross-reference files output by WLC and the schematic netlists were processed by a separate program, MATCH_CHECKER, to verify the connectivity of nodes that crossed partition boundaries. This additional step added only eight minutes to the total elapsed time for comparing chip netlists.

Capacitance Extraction

The small spacing dimensions of the submicron CMOS-4 process caused fringing and lateral capacitances to contribute significantly to the total nodal capacitance. The existing layout extraction tool only extracted overlapping parallel plate capacitance. Thus, a new layout capacitance extractor, CUP, was written to accurately extract fringing, lateral, and area capacitances.

CUP extracted interconnect capacitance from layout by decomposing interconnect layout into pieces of uniform layout cross sections. The geometry of each interconnect piece, and its distance from layers above, below, and adjacent to it, are used to calculate its area, fringing, and lateral components of capacitance. The empirical formulae used to calculate the capacitive components were based on curves of two-dimensional electrostatic simulation data of various layout cross sections. This technique produced accurate internodal and total interconnect capacitance data. This accuracy resulted in CUP being very compute intensive.

Multiprocessing was employed again to reduce the elapsed turnaround time for capacitance extraction batch jobs. CUP sectioned the layout database into fixed-size stripes, which were inserted into the batch queues of multiple CPUs. This method reduced the data complexity and allowed as many parallel computations as there were processors. During NVAX chip design, capacitance extraction was partitioned across as many as 20 CPUs. Multiprocessing reduced, for example, the NVAX I-box capacitance extraction from 26 hours to just 8 hours using 4 processors. Extraction of the E-box took 40 hours using one processor, but only 12 hours with 4 CPUs. Table 2 shows the device and node counts of the NVAX boxes (excluding the caches), and the CUP extraction run times on a VAX 6000 Model 500. Each box run resulted in approximately 500,000 extracted parasitic capacitors.

Resistance Extraction

Verifying the NVAX power, ground, and clock networks, and long signal lines required accurate extraction of interconnect resistance from layout. To meet this requirement, the REX resistance extractor was developed.[9] REX processed the output of HILEX to produce a series and parallel combination of resistors that modeled a node's interconnect. The resistor network was generated by fracturing the node layout into polygons based on changes in the layout geometries (width, length, bends) of the node or the occurrence of contacts. The effective resistance of each polygon and contact, or cluster of contacts, was then determined from technology parameters and the polygon geometries.

The power and ground resistor networks were extracted for individual boxes rather than the entire chip. The resulting networks were still quite large due to the fine granularity of the REX extraction process. Table 3 shows the extraction times for a REX job running on a VAX 6000 Model 500 and the total number of resistors extracted from each box.

Table 2 CUP Parasitic Capacitance Extraction Batch Run-time Data for NVAX Boxes

Box	Device Count	Node Count	Single CPU (Hours)	Four CPUs (Hours)
I-box	107,000	36,830	26	8
M-box	102,000	38,770	29	8
E-box	107,600	41,760	40	12
C-box	92,400	45,050	42	12
F-box	129,150	55,550	45	12.5

Table 3 REX Extracted Parasitic Resistance Data and Batch Run-time Data for NVAX Boxes

Box	Resistor Count	Extraction Time (Hours)
M-box	602,000	5
C-box	621,000	5
I-box	522,000	10
E-box	719,000	10
F-box	1,200,000	36

New Proprietary CAD Tools

Several other novel CAD tools were specifically designed for the NVAX chip. These tools provided practical solutions to verification and analysis problems that were previously unmanageable or intractable.

CHANGO Logic Simulator

CHANGO was an important development for NVAX functional verification because it allowed significantly more simulation cycles and functional verification tests to be performed from the NVAX transistor-level description than was previously possible. CHANGO is a two-state gate-level logic simulator designed to maximize performance through compiled, straight-line simulation. Electrical issues such as gate delay and charge sharing were not modeled since CHANGO was used for functional, not timing, verification. CHANGO's parallel simulation capability allowed the simultaneous execution of 13 different NVAX model simulations on one CPU, which resulted in an eightfold increase in simulation performance. Overall, CHANGO has been shown to accelerate simulation one to two orders of magnitude over traditional event-driven gate-level simulators. Its high throughput enabled us to boot the VMS operating system twice (75 million cycles) prior to tape out.

To create a CHANGO model, a transistor-level netlist description of the design was input to a preprocessor called GEN_MODEL. GEN_MODEL transformed the netlist into a logical description of the design, consisting of simple Boolean elements, D-type latches, and SR flops. CHANGO transformed this logical description into a highly optimized simulation stream of VAX assembly code.

CHANGO achieved its high simulation throughput in many ways. Conditional branch latency penalties were largely avoided because CHANGO simulation code is designed to execute in a straight-line fashion. Due to the high switching event densities we observed on NVAX, 18 percent on average, this straight-line compiled approach to simulation was more efficient than event-driven simulators, which typically fail to compete when event densities increase beyond 3 to 5 percent. The CHANGO translation process further optimized the simulation by partitioning the simulation according to signals that should be evaluated during each particular clock phase. This avoided processing signals during clock phases when signal transitions could not occur. Further, evaluation of a switching event was only performed when the signal could affect the evaluation of some other signal. This prevented simulation of unimportant switching events that were ignored by the remaining design. Redundant signals (i.e., nodes with the same logical behavior) were grouped together as a list of synonym signals in order to model multiple nodes by only one simulation event.

NTV Timing Verifier

NTV is a static timing verification tool developed for use on the NVAX microprocessor.[10] NTV processed 350,000 circuit paths and checked 42,000 timing constraints on the NVAX design. NTV eliminated the need for the pattern-dependent dynamic speed verification strategy used by other chip designs and significantly reduced the extensive speed verification work needed for SPICE simulations. It identified critical paths that would have otherwise remained undetected due to the complexity and size of the NVAX design.

NTV read multiple flat transistor netlists with or without parasitics and automatically identified circuit structures such as complementary, dynamic, and cascode gates as well as several types of latches. Based on the classification of these structures, NTV identified timing constraints. For example, NTV checked that the latch storage nodes become valid before the latches closed. NTV also read user-specified timing for primary inputs and propagated node timing throughout the design based on when signals arrived at gate inputs, the drive capability of each gate, and its output loading.

NTV has three delay models that were used for calculating gate delay: (1) unit delay was used for an initial rough timing estimate before real parasitics were known, (2) a SPICE-calibrated lumped RC model was used for delay calculation of complementary gates, and (3) an Elmore-distributed RC

model was used for other structures.[11] NTV flagged circuits that failed to meet the identified timing constraints within a user-specified time tolerance. Like other static timing verifiers, some paths identified by NTV were "don't cares" or were logically impossible. The user eliminated these false paths by deleting timing constraint checks or by specifying mutual exclusivity between specified groups of nodes.

WAWOTH Interconnect Analyzer

Traditional manual techniques for checking RC delay, IR noise, and electromigration (EM) were impractical for NVAX due to the size and complexity of the design. A suite of CAD tools called WAWOTH was developed to perform these checks automatically, more accurately, and in far less time than would otherwise have been possible.

During EM and IR analysis, current sources representing gate-switching events were added to a REX-generated resistor network. The magnitude of each current source was calculated based on the average switching frequency of the gate, the load it drove, and whether average or peak current was desired for the current analysis mode. The network node voltages were then solved through LU decomposition. Peak voltages were flagged for IR analysis, and average and peak current densities were calculated for each resistor element and checked against EM limits.

During RC analysis, node capacitance was proportionately distributed along the resistor network. The resulting RC network was processed by Carnegie-Mellon's AWE algorithm to generate a close approximation of the transfer function for the network.[12] From this, the step response RC delay was calculated and the delay response to any specified edge was found through convolution of the transfer function.

Since it was neither possible nor necessary to perform RC and EM analysis on all signal nodes, WAWOTH contained a number of tools that identified only those nodes that would have some chance of failing these checks. To decrease run time, we reduced the size of the files that were input to WAWOTH by eliminating any devices and parasitics that were not related to the node under examination. Noteworthy were the large data requirements met by WAWOTH. For example, WAWOTH calculated the current through the 719,000 resistive elements that compose the power and ground nodes of the E-box. Current stimulus of the network was derived from average node-switching frequencies calculated from logic simulation data. Over 1,800 signal nodes were also analyzed by WAWOTH.

Conclusions

Our design strategies, methods, and CAD tools allowed us to complete the NVAX CPU chip design in 30 percent less time than our initial schedule had required. Typical parts operate at 83.3 MHz (a 12-ns cycle time) under worst-case conditions for temperature and power supply. This is 2 ns better than our maximum cycle time design constraint. The chip booted the VMS operating system ten days after the first prototype wafers were available, and booted the ULTRIX system a few days later. Multiprocessing was running within weeks. Fifteen obscure logic bugs were found in the first-pass chips, but none of them impeded system debug or prototype development. No circuit design bugs were found. Only one design revision was needed prior to volume chip manufacturing.

Careful design, complexity management, and proprietary CAD tools targeted to large custom CMOS integrated circuits played crucial roles in the successful design of the NVAX microprocessor.

Acknowledgments

We would like to acknowledge the contributions of the following people: W. Anderson, E. Arnold, R. Badeau, I. Bahar, M. Benoit, B. Benschneider, D. Bernstein, D. Bhavsar, L. Biro, M. Blaskovich, P. Boucher, W. Bowhill, L. Briggs, J. Brown, S. Butler, R. Calcagni, S. Carroll, M. Case, R. Castelino, A. Cave, S. Chopra, M. Coiley, E. Cooper, R. Cvijetic, M. Delaney, D. Deverell, A. DiPace, D. DuVarney, R. Dutta, J. Edmondson, J. Ellis, H. Fair, G. Franceschi, N. Geagan, S. Goel, M. Gowan, J. Grodstein, P. Gronowski, W. Grundmann, H. Harkness, W. Herrick, R. Hicks, C. Holub, A. Jain, R. Khanna, R. Kiser, D. Koslow, K. Ladd, S. Levitin, S. Martin, T. McDermott, K. McFadden, B. McGee, J. Meyer, M. Minardi, D. Miner, T. O'Brien, J. Pan, H. Partovi, N. Phillips, J. Pickholtz, R. Razdan, S. Samudrala, J. Siegel, K. Siegel, C. Somanathan, J. St. Laurent, R. Stamm, R. Supnik, M. Tareila, S. Thierauf, M. Uhler, N. Wade, S. Watkins, and Y. Yen.

References and Note

1. G. Uhler et al., "The NVAX and NVAX+ High-performance VAX Microprocessors," *Digital Technical Journal*, vol. 4, no. 3 (Summer 1992, this issue): 11–23.

2. B. Zetterlund, J. Farrell, and T. Fox, "Microprocessor Performance and Process Complexity in CMOS Technologies," *Digital Technical Journal,* vol. 4, no. 2 (Spring 1992): 12–24.

3. J. Crowell, K. Chui, T. Kopec, S. Nadkarni, and D. Sovie, "Design of the VAX 4000 Model 400, 500, and 600 Systems," *Digital Technical Journal,* vol. 4, no. 3 (Summer 1992, this issue): 60–72.

4. L. Chisvin, G. Bouchard, and T. Wenners, "The VAX 6000 Model 600 Processor," *Digital Technical Journal,* vol. 4, no. 3 (Summer 1992, this issue): 47–59.

5. R. Badeau et al., "A 100 MHz Macropipelined VAX CMOS Microprocessor," Accepted for publication in *IEEE Journal of Solid State Circuits,* vol. 27, no. 11 (November 1992).

6. SPICE is a general-purpose circuit simulator program developed by Lawrence Nagel and Ellis Cohen of the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley.

7. T. Fox, P. Gronowski, A. Jain, B. Leary, and D. Miner, "The CVAX 78034 Chip, a 32-bit Second-generation VAX Microprocessor," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 95–108.

8. W. Durdan, W. Bowhill, J. Brown, W. Herrick, R. Marcello, S. Samudrala, G. Uhler, and N. Wade, "An Overview of the VAX 6000 Model 400 Chip Set," *Digital Technical Journal,* vol. 2, no. 2 (Spring 1990): 36-51.

9. J. Hwang, "REX—A VLSI Parasitic Extraction Tool for Electromigration and Signal Analysis," *28th Design Automation Conference* (1991): 717–722.

10. J. Pan, L. Biro, J. Grodstein, B. Grundmann, and Y. Yen, "Timing Verification on a 1.2M-Device Full-Custom CMOS Design," *28th Design Automation Conference* (1991): 551–554.

11. W. Elmer, "The Transient Response of Damped Linear Networks with Particular Regard to Wide-band Amplifiers," *Journal of Applied Physics,* vol. 19 (1948): 55–63.

12. V. Raghavan and R. Rohrer, "A New Nonlinear Driver Model for Interconnect Analysis," *28th Design Automation Conference* (1991): 561–566.

Walker Anderson |

Logical Verification of the NVAX CPU Chip Design

Digital's NVAX high-performance microprocessor has a complex logical design. A rigorous simulation-based verification effort was undertaken to ensure that there were no logical errors. At the core of the effort were implementation-oriented, directed, pseudorandom exercisers. These exercisers were supplemented with implementation-specific focused tests and existing VAX architectural tests. Only 15 logical bugs, all unobtrusive, were detected in the first-pass design, and the operating system booted with first-pass chips in a prototype system.

The NVAX CPU chip is a highly complex VAX microprocessor whose design required a rigorous verification effort to ensure that there were no logical errors. The complexity of the design is a result of the advanced microarchitectural features that make up the NVAX architecture, such as branch prediction, micropipelining and macropipelining techniques, a three-level hierarchy of instruction caching, and a two-level hierarchy of write-through and write-back data caching.[1] Also, the chip was initially intended for two different target system configurations and had to be verified for operation in both. Product time-to-market goals mandated a short development schedule relative to previous projects, and there was a limited number of verification engineers available to perform the tasks.

The verification team set two key goals. The first was to have no "show stopper" logical bugs in the first-pass chips and, consequently, to be able to boot the operating system on prototype systems. Meeting this goal would enable the prototype system hardware and software development teams to meet their schedules and would allow more intensive logical verification of the chip design to continue in prototype systems. The second key team goal was to design second-pass chips with no logical bugs, so that these chips could then be shipped to customers in revenue-producing systems. Meeting this goal was critical to achieving the time-to-market goals for the two planned NVAX-based systems.

Team Organization and Approach

Logical verification was performed by a team of engineers from Digital's Semiconductor Engineering Group (SEG) whose primary responsibility was to detect and eliminate the logical errors in the NVAX design. The detection and elimination of timing, electrical, and physical design errors were left to separate efforts.[2]

Given the design complexity, the critical need for highly functional first-pass chips, and the fact that the designers had other responsibilities related to the circuit and physical implementation of the full-custom chip, special attention to logical verification was considered a requirement. Every verification engineer approached the verification problem with a different focus. Each member of one group of engineers focused on the verification of a single box, while other engineers focused on functions that spanned several boxes. Certain verification engineers were available throughout the project to test the functions of the chip that required extra attention. This variety of perspectives was an important aspect of the verification strategy. Most verification engineers followed the process described below.

1. Plan tests for a function.
2. Review those plans with the design and architecture teams.
3. Implement the tests.
4. Review the actual testing with the design and architecture teams.
5. Implement any additional testing that was deemed necessary.

Simulation and Modeling Methodology

The verification effort employed several models of the full NVAX CPU chip and of the individual design elements. Each model had its strengths and weaknesses, but all models were necessary to ensure a thorough logical verification of the design.

Behavioral Models

The behavioral models of the chip were written by design team members using the DECSIM behavioral modeling language; to achieve optimal simulation performance, they were written in a procedural style. These models are two-state models that are logically accurate at the CPU phase clock boundaries. These fairly detailed behavioral models represent logic at the register transfer level (RTL), with every latch in the design represented and the combinational logic described in a way similar to the ultimate logic design.

Behavioral simulations were performed first on box-level models, where most of the straightforward design and modeling errors were eliminated. A box is a functional unit such as the instruction prefetch/decode and instruction cache control unit, the execution unit, the floating-point unit, the memory management and primary cache control unit, or the bus interface and backup cache control unit.[1]

The box-level models were then integrated into a full-chip behavioral model, which also included a backup cache model, a main memory model, and models to emulate the effects of several system configurations. The pseudosystem models did not model any one specific target system configuration but could be set up to operate effectively like any target system configuration or in a way that exercised the chip more intensely than any target system would. Available early in the project, this model was the primary vehicle for logical verification until the schematics-derived, full-chip, in-house CHANGO model was created. The full-chip behavioral model could simulate approximately 13 cycles per VAX VMS CPU second and was used to simulate about one billion CPU cycles.

The procedural, full-chip behavioral model also ran on a hardware simulation accelerator where it achieved simulation performance of about twice that of the unaccelerated simulation. The simulation accelerator was used primarily for simulating long, automated, noninteractive tests.

In addition, the model was encapsulated in an event-driven shell and incorporated into module (i.e., board) and then system models. The chip verification team performed only a limited amount of simulation using these module and system models. These simulations were used primarily to verify that the chip model functioned correctly in a more accurate model of a target system configuration and to better test the multiprocessor support functions in the design. The system development groups performed more extensive simulations with such models.

Schematics-derived Models

Schematics-derived models were created and simulated at both the box and full-chip level. The CHANGO simulator is a two-state, compilation-driven simulator and, like the behavioral model, is accurate only at the CPU phase clock boundaries.[2] The full-chip CHANGO model linked together the following: the code that was automatically generated from the schematics; C-code models for chip-internal features such as control store and random-access memories (RAMs); C-code models to perform simulation control functions; and the same DECSIM behavioral models for the backup cache, main memory, and system functions that were used in the full-chip behavioral model. The simulation performance of the full-chip CHANGO model was only about one-half that of the unaccelerated, full-chip behavioral model. Although these models were not useful for electrical or timing verification because they did not model timing or electrical characteristics of the design, their simulation performance made them extremely useful for logical verification.

Another full-chip model was created to run on an event-driven, multiple-state simulator. However, only a limited amount of simulation was performed using this model, because its performance was very slow when compared to the CHANGO and behavioral models. Since it was the only model that could run on a multiple-state simulator, this third model was used primarily to verify chip power-up and initialization.

Pseudorandom Exercisers

Early in the project, it became apparent that, given the limited number of engineers, the short schedule, and the complexity of the NVAX chip design, a strategy of developing and simulating all conceivable implementation-specific test cases would be ineffective. This strategy would have required the engineers to implement tedious, handcrafted tests.

Instead, the verification team adopted a strategy that depended heavily on the use of directed, pseudorandom tests referred to as exercisers. This strategy generated and ran many more interesting test cases than would ever have been conceived by the verification and design engineers themselves.

The basic structure of an exerciser consisted of the following five steps, which were repeated until a failure was encountered:

1. Set up the test case.
2. Simulate the test case on either the behavioral or the CHANGO model.
3. Execute the test program on a VAX reference machine.
4. Analyze the simulation and accumulate data.
5. Check the results for failure.

Figure 1 depicts the interoperation of the tools used to construct an exerciser and its basic flow.

Setup

Setting up the test case involved generating a short assembly language test program, activating some demons to emulate various system effects, and selecting a chip/system configuration for simulation.

The assembly language test programs were generated using SEGUE, a text generation/expansion tool developed for this project. This tool processes script files that contain hierarchical text generation templates and implements the basic functions of a programming language.

SEGUE provides a notation that allows the user to specify sets of possible text expansions. Elements of these sets can be selected either pseudorandomly or exhaustively, and the user can specify the weighting desired for the selection process. For example, a hierarchy of SEGUE templates typically comprised three levels. At the lowest level, a SEGUE template was created to select pseudorandomly a VAX opcode, and another template was created to select a specifier, i.e., operand. At an intermediate level, the verification engineers created templates that called the lowest-level templates to generate short sequences of instructions to cause various events to occur, e.g., a cache miss, an invalidate from the system model, or a copy of register file contents to memory. At the highest level, these intermediate-level templates were selected pseudorandomly with varied weighting to generate a complete test program.

Because the SEGUE tool was developed with verification test generation as its primary application, the syntax allows for the easy description of test cases and the ability to combine them in interesting fashions. Using SEGUE, the verification engineers were able to create top-level scripts quickly and easily that could generate a diverse array of test cases. These engineers considered SEGUE to be a significant productivity-enhancing tool and

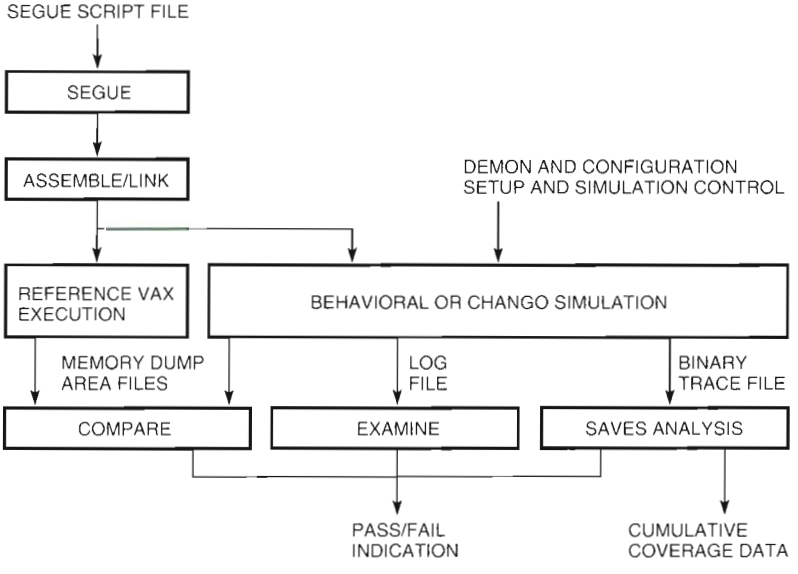


Figure 1 Verification Tool Flow for Exercisers

preferred using SEGUE to hand-coding many focused tests.

Before simulations were performed, model demons were set up. Demons were enabled or disabled, and their operating modes were selected pseudorandomly. Demons may be features of the model environment that cause some external events such as interrupts, single-bit errors, or cache invalidates to occur at pseudorandom, varying intervals. Demons may also be modes of operation for the system model that cause pseudorandom variation in operations such as the chip bus protocol, memory latency, or the order in which data is returned. Some demons were implemented to force chip-internal events, e.g., a primary cache parity error or a pipeline stall. These chip-internal demons had to be carefully implemented, because sometimes they forced an internal state from which the chip was not necessarily designed to operate. In a pseudorandomly generated test, it is frequently difficult or impossible to check for the correct handling of an event caused by a demon, e.g., check that an interrupt is serviced by the proper handler with correct priority. However, simply triggering those events and ensuring that the design did not enter some catastrophic state proved to be a powerful verification technique.

Chip/system configuration options such as cache enables, the floating-point unit enable, and the backup cache size and speed were also preselected pseudorandomly. Aside from testing the chip in all possible configurations, e.g., with a specific cache disabled, varying the configuration in a pseudorandom manner caused instruction sequences to execute in very different ways and evoked many different types of bugs unrelated to the specific configuration. Also, specific configurations and demon setups would significantly slow down the simulated execution of the test program, sometimes to the point where intended testing was not being accomplished. To work around this problem, the verification engineer could force the configuration and demon selection to avoid problematic setups.

Simulation and VAX Reference Execution

After assembling and linking the test program, it was loaded into modeled memory, and its execution was simulated on either the behavioral or the CHANGO model. As the test program simulation took place, a simulation log file and a binary-format file, which contained a trace of the state of the pins and various internal signals, were created. As the exerciser test programs executed, various VAX architectural state information was written periodically to a region of modeled memory referred to as the dump area. When the simulated execution of the test program completed, the contents of the dump area were stored in a file. Also, the test program was executed under the VMS operating system running on a VAX computer used as a reference machine. At the end of execution, the contents of the memory dump area were stored in another file.

Analysis

A tool called SAVES allows users to create C programs in order to analyze the contents of binary trace files. SAVES was used to provide coverage analysis of tests, and to check for correct behavior of chip-internal logic and give a pass/fail indication.

For coverage analysis purposes, information such as the number of times that a certain event occurred during simulation or the interval between occurrences was accumulated across several simulations. This data gave the verification engineer a sense of the overall effectiveness of an exerciser. For example, a verification engineer who wanted to check an exerciser that was intended to test the branch prediction logic was able to use the SAVES tool to measure the number of branch mispredictions.

Frequently, verification engineers used the SAVES tool to perform cross-product analysis and data accumulation. For cross-product analysis, the engineer specified two sets of design events to be analyzed. The analysis determined the number of times that events from the first set occurred simultaneously with (or skewed by some number of cycles from) events in the second set. For example, one verification engineer analyzed the occurrence of different types of primary cache parity errors relative to different types of memory accesses. Analyzing the cross-product of state machine states against one another, skewed by one cycle, allowed state machine transition coverage to be quickly understood.

The verification team used this SAVES information about the exerciser coverage in the following ways:

- To enhance productivity by helping the engineers identify planned tests that no longer needed to be developed and run because the exerciser already covered the test case

- To indicate significant areas of the design where coverage may have been deficient
- To determine how the exercisers might be adjusted to become more effective or thorough, or to focus on a particular low-level function of the chip design

Pass/Fail Checking

Several checking mechanisms were employed to determine whether tests passed or failed. The SAVES tool was used to check for correct behavior of the design, especially where correct behavior was difficult to observe at a VAX architectural level. For example, the verification engineers used SAVES to check the proper functioning of performance-enhancing features such as branch prediction logic, pipelines, and caches.

A VMS command procedure automatically scanned simulation log files for error output from any of several design assertion checkers built into the model. These assertion checkers varied widely in complexity. For example, simple assertion checkers ensured that unused encodings of multiplexers' select lines never occurred. As another example, a more sophisticated and complex assertion checker verified that the CPU had maintained cache coherence and proper subsets among the three caches and the main memory.

The same VMS command procedure checked the simulation log file to verify that the simulation of the execution of the test program reached the proper completion program counter. Finally, a simple program compared the memory dump area files generated by the simulation and the reference machine execution to verify that the memory dump areas were identical. Although the simulated test program may have followed a different execution path from the VAX reference execution because it was simulated in the presence of demons, the completion points of both executions were the same, and the VAX architectural state information that was compared was identical.

If these checks found no errors, the exerciser looped back to generate another test case. Because this whole process was automated, the verification engineer could run this test continuously, on all available computing resources.

Other Aspects of the Exercisers

The exercisers were the core of the NVAX CPU chip verification effort. They were run nearly continuously throughout the project on behavioral and/or CHANGO models, and proved to be very effective at detecting subtle, complex bugs in the design. Each exerciser concentrated on testing a single box, a subsection of a box (e.g., branch prediction logic), or a particular global chip function. By adjusting the SEGUE template weightings, preventing or forcing the use of a particular demon, or forcing a particular configuration parameter, the exercisers could be controlled at a high level to focus on low-level functions. Verification engineers traded interesting SEGUE templates among themselves to provide each exerciser with a rich and diverse set of possibilities for code generation, while still maintaining the intended focus of the exerciser.

Focused Tests

Several focused tests were generated to supplement the exercisers. These were necessary to test implementation-specific aspects of the design that could not be checked by comparing results against a VAX reference machine. In some cases, an exerciser could have been used to test a particular function, but the verification engineer judged it easier to hand-code a focused test program than to control an exerciser in order to accomplish the testing. Focused tests were necessary and particularly challenging to create and maintain when very precise timing of events was required to test a certain scenario of chip operation. This timing could be achieved only by handcrafting an assembly language test and running it under carefully controlled simulation conditions.

Each of the focused tests was run at least once on the full-chip behavioral model and then again on the full-chip CHANGO model.

Other Tests

Several tests that had been used for the verification of previous VAX implementations were also used for verification of the NVAX CPU chip. The use of these tests allowed the NVAX logical verification team to focus on the implementation-specific complexities of the NVAX design and not expend as much effort on implementation-independent, VAX architectural verification.

The HCORE suite of tests can be used to verify several permutations of all VAX instructions, as well as some VAX architectural concepts, e.g., memory management.[3] HCORE was valuable in that it was the first test used to debug both the full-chip behavioral model and the CHANGO model.

Small portions of the HCORE suite were used as a nightly model regression test. In general, very little

regression testing of the NVAX models took place; the team believed that using computing resources to run pseudorandom exercisers and other new tests was of more value to the verification effort than consuming resources with extensive, frequent regression testing. Consequently, the entire HCORE suite was run at only a few key checkpoints during the project.

AXE is a VAX architectural exerciser that pseudorandomly generates single-instruction test cases.[4] MAX is an extension of AXE that generates multiple-instruction test cases with complex data dependencies between the instructions. Both tools set up enough VAX architectural state to prepare for a test case, simulate the test case on a model, execute the test case on a VAX reference machine, compare VAX architectural state information from the simulation and VAX reference execution, and finally, report any discrepancies. Each test case may force some number of exceptions; the AXE and MAX tools ensure that all exceptions are detected and properly handled.

The AXE and MAX tools generate tests with no knowledge of the particular VAX implementation being tested and thus differ from the implementation-specific exercisers. Consequently, AXE and MAX are less effective than the implementation-specific exercisers for intensive exercising of performance-enhancing features that are transparent from a VAX architectural perspective. However, MAX was an effective test for the micropipelining and macropipelining aspects of the NVAX design. Altogether, about 706,000 AXE test cases and 137,000 MAX test cases were run on the behavioral model.

Schematic Verification

An initial goal of the NVAX CPU chip verification team was to perform a more extensive verification of the schematic design than had been accomplished in the past. Because of the development of the CHANGO simulator, with its significant performance advantage over previously used logic simulators, the team met this goal. Approximately 75 million NVAX CPU cycles were simulated on the schematics-derived, full-chip CHANGO model.

Box-level CHANGO Simulation

First, box-level CHANGO models were constructed and tested using a technique called patterns-on-the-fly (POTF). This technique involved simultaneously starting a full-chip behavioral model simulation process and a box-level CHANGO simulation process under the VMS operating system and then communicating between the processes. Stimulus and response data from the behavioral simulation is used to drive the inputs to and check the outputs from the box-level CHANGO model. In addition to comparing primary outputs from the box, this technique was used to compare many chip-internal points. The POTF technique eliminated the need to extract and maintain large pattern files from behavioral simulations and proved to be a straightforward way of comparing the two models. Exercisers and focused tests were run using the POTF method, and several bugs were quickly and easily isolated. Because a close correlation between the behavioral models and the implementation as represented by the schematics had been maintained, few conceptual, logical design errors were found by the box-level, POTF simulations. These simulations were, however, extremely useful for finding simple schematic entry errors.

Full-chip CHANGO Simulation

Next, the team constructed the full-chip CHANGO model. The simulation environment of this model included many features available in the behavioral model environment. After simulating the HCORE suite of tests, all the focused tests were run on the full-chip CHANGO model, and the exercisers were run on this model for several weeks. In addition, 44,000 AXE cases and 33,000 MAX cases were run on the full-chip CHANGO model. All these simulations uncovered only one additional schematic entry error.

Simulation of the VMS Boot Process

To ensure the success of operating system booting, i.e., initial processor loading, on first-pass chips and as a final functional test of the design, members of the architecture team simulated the VMS operating system boot process on the full-chip CHANGO model. The operating system source code was modified to add support for the NVAX-specific features and for the modeled system environment. A VMS system disk that contained the changes was created on an existing VAX system. Each block of the disk was copied to a VMS file, which was then used as the system disk image during simulation.

A disk model with a simple programming interface and a direct memory access (DMA) capability was added to the simulation environment of the full-chip CHANGO model. The disk model read blocks

from the system disk image file, and wrote data to a small cache of internally maintained disk blocks. To accelerate disk transfers, the disk model would examine cache state and use the system bus for the disk transfers only when the data was present in the cache and required a cache invalidate or write back. In other cases, the data was transferred directly into the memory subsystem in zero simulated time.

While tracking the progress of the simulation, the team identified operating system code that executed a time-consuming search algorithm. To limit the amount of time spent in this loop, the code was rewritten to implement a much faster algorithm. However, because the booting simulation effort could not be restarted from the beginning, several utilities were developed that allowed the code to be replaced in the system disk image file and in simulated memory during a pause in the simulation.

To provide the ability to restart the simulation effort and move it to any available computing resource, simulation state was saved after every 50,000 to 100,000 cycles of simulation. In total, approximately 25 million cycles were simulated. The simulation was stopped at the point where multiple processes were created and the main start-up process began executing. Even though this effort identified no bugs in the design, it did provide a high degree of confidence that the design was ready to be released for fabrication of first-pass chips.

Prototype Chip Verification

The prototype NVAX chips were verified in several VAX 6000 Model 600 multiprocessor systems. The CPU module was the only new hardware component in the system; the backplane, memory, and I/O subsystem were known to be robust, because they were used in the VAX 6000 Model 500 system. One logic analyzer was connected to the system bus, and another was connected to the pins of the NVAX chip.

The strategy for the early prototype verification was to boot the low-level console user interface, run the HCORE suite of tests, boot the VMS operating system, and then run the User Environment Test Package (UETP) system exerciser. Within 10 days of receiving the first prototype chips, all these tasks had been accomplished. Later, the AXE and MAX exercisers were run on the prototype systems.

The rigorous testing that continued on prototype systems revealed a few logical bugs which had gone undetected during simulated verification. Typically, information about a bug was collected on the prototype system, and then the failing scenario was reproduced on the behavioral model, where the scenario could be analyzed and better understood. The chip-internal signals were extremely difficult to observe, but a 12-bit, parallel port allowed access to one of eight sets of signals from various sections of the chip. The ability to monitor the control store address bus by means of this parallel port proved to be an essential debugging feature.

The control store patching mechanism that was part of the chip design helped identify some bugs in prototype chips. The debugging engineers successfully used microcode patches to work around several of the hardware and microcode bugs. In cases where a microcode bug was patched, extensive system testing verified that the planned change was correct.

Bug Tracking and Design Release

Bug detection was a key status indicator throughout the NVAX logical verification effort and thus helped to steer the team's work. Bugs were tracked carefully with an on-line system and analyzed each week to consider trends, successful and unsuccessful bug-finding techniques, and bug hot spots, which required additional attention. The bug detection rate was fairly constant throughout the project at about 22 per month, with the exception of the last month in which the rate dropped to nearly zero. An analysis of the bug-detecting effectiveness of each testing technique shows that all test techniques were effective and seemed to complement each other. Table 1 shows the percentage of bugs detected by each technique. This table includes data on the ever-valuable, nonsimulation verification technique of simply reviewing, inspecting, and discussing the design and its many representations.

The decision to release the design for fabrication of first-pass chips was a consensus decision made by the verification, architecture, and design teams. From a verification perspective, the design was ready for release when the bug detection rate remained at zero for several weeks and the majority of the planned tests had been implemented. The verification of some areas of the design was deferred until after the release of the first-pass design. The development team decided that any bugs that might be found in these areas would not have a significant negative impact on the system development schedule, whereas additional delay in releasing the design would.

Table 1 Bug Detection Using Various Techniques

Technique	Percent of Total Bugs Found
Focused tests	28
Directed, pseudorandom exercisers	23
Review, inspection, observation, thought	20
Detection technique unknown	12
AXE	8
MAX	6
HCORE	2
Other	1

Results and Conclusions

Only 15 logical bugs were found in the first-pass NVAX CPU chip design, all of which were either easily worked around or did not impact normal system operation. The nature of the bugs found in the first-pass design ranged from straightforward bugs that escaped detection for clear-cut reasons to extremely complex bugs that required hours or weeks of rigorous prototype system testing to uncover. Some of the bugs escaped detection during simulated verification for classic reasons such as:

- Testing of the function was performed just before release, in a hurried manner.
- Simulation performance prohibited running a certain type of test case.
- A test was not run in a certain mode due to the difficulty of running it in all possible modes.
- It took an exerciser running on a simulator a long time to encounter the conditions that would evoke the bug.
- A test was inadvertently dropped from the set of exercisers that were run continuously.

Details about five of the more interesting bugs found in the first-pass design follow. Included is information about how the bug was detected, a hypothesis on why the bug eluded detection before first-pass chips were fabricated, and lessons learned from the detection and elimination of the bug.

1. One simple bug was detected by running the HCORE test suite on the prototype system with the floating-point unit (F-box) disabled. This bug could have been found in the same way through simulation, but the test suite was not run as a final regression test with the F-box disabled. In general, focused tests like HCORE were not run with varied chip/system configurations. The verification team concluded that all focused tests should be run with different chip/system configurations. At the minimum, a configuration that disables all possible functions should be tested.
2. Another bug was discovered because the CPU chip generated spurious writes to memory in the prototype system. The exercisers probably did generate the conditions necessary to evoke this bug; however, the spurious writes went unnoticed. It is extremely difficult to verify that a machine does everything it is supposed to do and nothing more. Additional assertion checkers or monitors in the models might detect such bugs in the future.
3. A third bug was evoked when a prototype system exerciser executed a translation buffer invalidate all (TBIA) instruction under certain conditions. On a real system, the TBIA instruction is used only by the operating system. In our verification effort, the TBIA instruction was little used by the exercisers that were simulated. Operations that are performed only by the operating system should not be underemphasized in exercisers.
4. One first-pass bug was related to the halt interrupt, which is used only during debugging operations. The halt interrupt received minimal testing and was not tested at all in any type of exerciser. Discovering this bug was especially annoying because a similar bug had escaped detection by the initial logical verification effort for a previous VAX implementation. This turn of events reinforces the belief that there is value in reviewing the escaped bug lists from other projects. Also, during the verification effort, there seemed to be a natural, but erroneous, tendency to undertest functions used infrequently or not at all during normal system operation. Such functions sometimes require extra attention, because they may be quite complex and may have been given less careful thought during the design process.

5. A state bit that needed to be initialized on power-up was not. This problem was noticed during initialization simulation but erroneously rationalized as being acceptable. Design assumptions and assertions about initialization should be verified through simulation or other means.

Overall, the NVAX CPU chip logical verification effort was a success. The pseudorandom testing strategy detected several complex and subtle logical bugs that otherwise probably would not have been detected by simulation. The extensive simulation performed on the schematics-derived model of the chip provided a high degree of confidence in the design.

The goals of producing highly functional first-pass chips and bug-free, second-pass chips were both met. Neither the bugs in first-pass chips nor their work-arounds impeded prototype system debugging in any significant way, and first-pass chips with work-arounds were used in preproduction, field-test systems. The verification team corrected the 15 first-pass design bugs for second-pass chips, which were shipped to customers in revenue-producing systems.

Acknowledgments

The NVAX logical verification effort was performed by a team of engineers from the SEG microprocessor verification group. Members of this team included Walker Anderson, Rick Calcagni, Sanjay Chopra, and John St. Laurent. NVAX architects Mike Uhler and Debra Bernstein provided extensive technical direction and assistance to the verification team. The SEG CAD group helped by its continual development and support of tools. The CHANGO simulator would not have been possible without the significant contributions of Kevin Ladd. Will Sherwood provided high-quality, top-level guidance through all phases of the project. The system development groups performed system-level simulations and rigorous prototype testing. The VAX Architecture Group AXE/MAX team, once again, provided and supported an effective verification tool. Lastly, the success of the project and the final quality of the NVAX chip logical design are as much a tribute to the work of the NVAX architecture and design teams as they are to the work of the verification team.

References

1. G. Uhler et al., "The NVAX and NVAX+ High-performance VAX Microprocessors," *Digital Technical Journal*, vol. 4, no. 3 (Summer 1992, this issue): 11–23.

2. D. Donchin et al., "The NVAX CPU Chip: Design Challenges, Methods, and CAD Tools," *Digital Technical Journal*, vol. 4, no. 3 (Summer 1992, this issue): 24–37.

3. R. Calcagni and W. Sherwood, "VAX 6000 Model 400 CPU Chip Set Functional Design Verification," *Digital Technical Journal*, vol. 2, no. 2 (Spring 1990): 64–72.

4. D. Bhandarkar, "Architecture Management for Ensuring Software Compatibility in the VAX Family of Computers," *IEEE Computer* (February 1982): 87–93.

Lawrence Chisvin
Gregg A. Bouchard
Thomas M. Wenners

The VAX 6000 Model 600 Processor

The Model 600 is the newest member of the VAX 6000 series of XMI2-based, multiprocessing computers. The Model 600 processor integrates easily into existing platforms. Each processor module provides 40.5 SPECmarks of performance made possible by the NVAX CPU chip. The major VLSI interface chip, called NEXMI, was created using Digital's internal CMOS-3 design and layout process. The ability to design and fabricate the interface chip internally was critical to delivering a working CPU prototype module on schedule. The aggressive module timing goals were met by employing previous module experience in combination with extensive SPICE simulation.

The Model 600 system is the latest addition to Digital's VAX 6000 family of midrange symmetric multiprocessing computers.[1] The Model 600 was designed to be integrated cost-effectively into an existing VAX 6000 platform, and to provide a significant performance improvement over the previous generation of systems. Table 1 compares the performance of the Model 600 with the previous generation Model 500 on several important benchmarks. The powerful NVAX single-chip microprocessor enables this level of performance.[2]

Table 1 Comparison of CPU Performance

Benchmark	VAX 6000 Model 600	VAX 6000 Model 500
SPECmark[3]	40.5	15.3
SPECint	30.9	14.2
SPECfp	48.6	16.1
VUPs	30.2	12.4

Notes: SPECmark is a quantitative measure of performance, determined by running a suite of 10 benchmark programs. SPECint defines the performance on the subset of the tests that are integer intensive, and SPECfp defines the floating-point intensive tests. A VAX-11/780 system has a performance of 1.0 SPECmark by definition. The SPEC tests used for the table values are from the System Performance Evaluation Cooperative, Release 1. VUP is a VAX unit of performance. One VUP equals the performance of a VAX-11/780.

Design Goals

The primary goal of the project was to deliver a module that included the appropriate support functions, performance, and VAX 6000 system compatibility. Equally important, the module had to be delivered on schedule to prevent an adverse time-to-market impact on the program. This included the delivery of a working prototype module before the first NVAX microprocessor chips were available, since a VAX 6000-based platform would be used to debug the initial NVAX CPU chips. Furthermore, the prototype module had to allow the VMS operating system to be booted and tested.

Our goals were achieved. When the NVAX CPU chip, the prototype module, and the NEXMI support applications specific integrated circuit (ASIC) were integrated for the first time, the hardware worked almost immediately. The software team was well prepared, and the full VMS system boot took place 11 days after the hardware was put together. Moreover, the first-pass modules were used for all the system debugging, and were of sufficient quality to be used for system field test.

This paper relates the background and design process for the VAX 6000 Model 600 processor module. The module and its system context are described, as well as many of the design decisions that were made during the project. The first section describes the module in general terms, along with some of the trade-offs and choices associated with its development. The second section focuses on the NEXMI support chip, which is the primary interface device positioned between the NVAX CPU data and address lines (NDAL), the XMI2 system bus, and the support peripheral ROMBUS. It discusses the very large-scale integration (VLSI) design process used to create and verify the functions of this interface. The final section details some of the physical aspects of the module design, including the important work performed to ensure good module signal integrity and thermal management.

Description of the Processor Module

Figure 1 shows the VAX 6000 Model 600 CPU module. Figure 2 is a block diagram of the module, showing the major subsections. The CPU module contains two VLSI components: the NVAX microprocessor and the NEXMI ASIC interface chip. The module also holds the backup cache static random-access memory (SRAM) devices, the XMI2 corner bus interface, and the supporting logic necessary to implement a VAX 6000 processor node.

The NVAX CPU directly controls its external backup cache SRAMs. When the data is resident in the cache, the NVAX CPU modifies it there, and implements a write-back scheme.[2] When the data misses in the backup cache, or when an I/O access is necessary, the NVAX CPU places the command on the NDAL, along with any associated control and data information. The NEXMI accepts and processes the command.

The NEXMI VLSI chip provides an interface to the functions necessary to integrate a VAX 6000 processor module into an XMI2-based system. In particular, the NEXMI chip:

- Translates the NDAL bus commands to XMI2 bus commands
- Returns read data from the XMI2 bus to the NVAX CPU (via the NDAL)

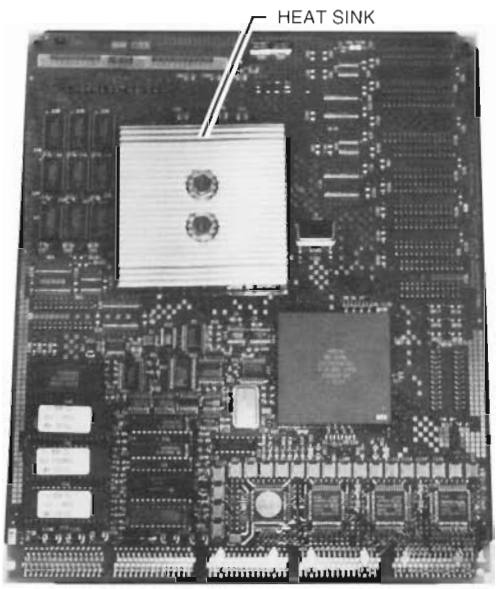


Figure 1 VAX 6000 Model 600 Processor Module

- Forwards invalidate traffic to the NVAX CPU for lookup and potential write back
- Controls NDAL bus arbitration

The NEXMI contains a programmable interval timer, reset logic, halt arbitration logic, and secure

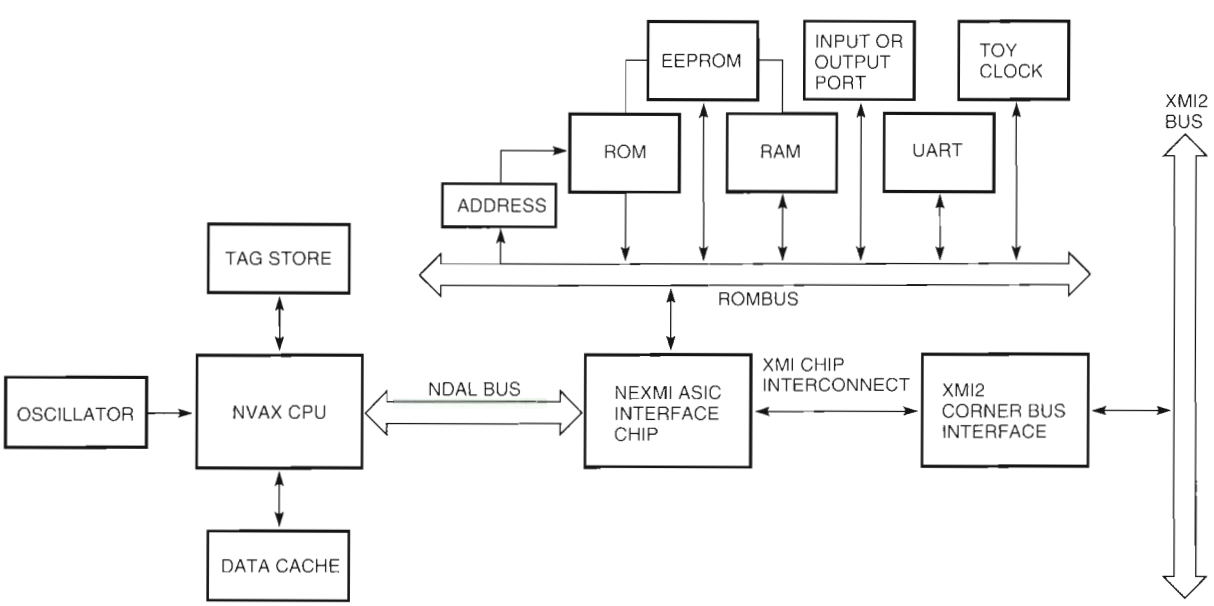


Figure 2 Block Diagram of the Model 600 Module

console logic on chip. It accommodates the rest of the support functions through the ROMBUS.

The ROMBUS is controlled by and interfaces to the NEXMI; it supplies a path for the low-speed devices that are necessary to create a working computer. The devices on the ROMBUS are off-the-shelf parts that contribute specific functions, including ROM for the boot diagnostic and console program, a stack RAM for storage of diagnostic/console dynamic information, an electrically erasable programmable read-only memory (EEPROM) for saved state, a universal asynchronous receiver/transmitter (UART) for console communication, an input and output port, and a time-of-year (TOY) clock.

The bare module itself is a sophisticated printed wiring board (PWB) with the following characteristics:

- 11.024-inch by 9.18-inch module size
- 10-layer module with 0.093-inch thickness
- 4 signal layers, 4 power/ground layers and 2 component/dispersion layers
- 0.010-inch vias
- 5-mil etch/7.5 mil space minimum for signals
- 10-mil etch/40 mil space minimum for clocks

NEXMI Support Chip

The NEXMI chip is the routing and control interface between the three major buses that reside on the VAX 6000 Model 600 processor module. A functional diagram of the NEXMI is provided in Figure 3. The three major buses are the NDAL, XMI2, and ROMBUS. The NEXMI chip contains the following functions, each function is associated with one or more of the three major buses:

- NDAL bus arbitration
- NDAL receive and transmit logic
- NDAL/XMI2 queues
- XMI2 data/invalidate responder queue
- XMI2 bus control logic
- System support control (SSC) logic

The NDAL receive logic latches and decodes commands and data from the NVAX CPU, and routes them to one of two queues. If the command is a write back, consisting of an address and 32 bytes of write data, it is placed in the XMI2 write-back queue. All other commands (reads and 8-byte writes) are placed in the non-write-back queue. The non-write-back queue services both the XMI2 logic and the SSC logic. Depending on the function, the SSC logic might then send the command on to the ROMBUS. Each queue is loaded in the NDAL time domain, and a request signal is sent to the appropriate XMI2 or SSC logic for processing.

On read commands, data is returned from the XMI2 responder queue or SSC control section, and forwarded to the NDAL through the NDAL transmit section. The NDAL is then requested, and when bus access is granted the data is driven onto the NDAL to be accepted by the NVAX CPU.

The XMI2 logic also sends potential invalidate addresses to the NVAX, where the information is compared with the existing tag address in the indexed backup cache block. An invalidate address is nothing more than the address associated with a command initiated on the XMI2 by another processor. If the cache block matches, and if the NVAX must relinquish control of the data, the block is either invalidated or written back. The choice depends upon the type of transaction and the state of the cache block.

In the Model 600, the NDAL arbitration is handled by the NEXMI chip. The NVAX CPU does not implement the NDAL arbitration on chip because the microprocessor must accommodate many different types of system platforms. A method of arbitration that is fair and efficient on one type of system (e.g., a single processor workstation with several potential NDAL master nodes implemented in off-the-shelf programmable devices) might be less than satisfactory for another type of system (such as the NEXMI).

The NEXMI and the NVAX CPU are the only two nodes on the NDAL in this implementation, so a simple priority scheme is used. The NEXMI always has highest priority, since it is either returning data or forwarding XMI2 bus transactions for potential invalidation and write back. In both cases, some other entity on the bus is actively waiting for the information to be returned.

Choice of NEXMI Technology

The NDAL is the NVAX CPU external interface bus. It is a 64-bit, bidirectional, multiplexed address and data bus that runs synchronously with the CPU. Although it is significantly slower than the internal CPU speed (an NVAX with a 12-nanosecond [ns] clock cycle has an NDAL with a 36-ns cycle), it is still aggressive in many respects, and presented a challenge to design using standard parts. The NDAL

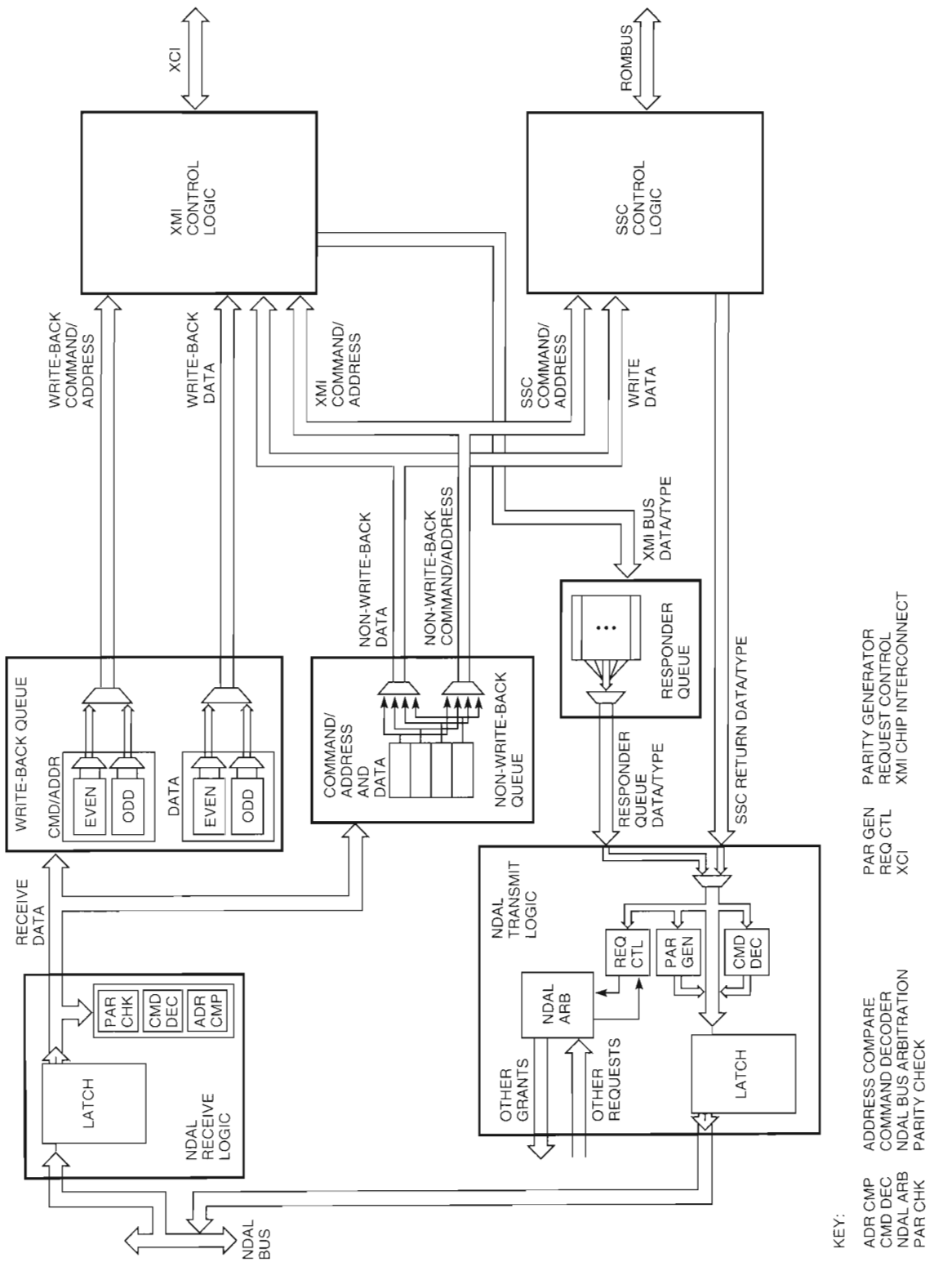


Figure 3 Block Diagram of the NEXMI Chip

arbitration for the next cycle happens in parallel with the data transfer for the current cycle, and the full request and grant loop must be performed in one NDAL cycle. In addition, the NDAL data path is bidirectional. An NDAL master must be able to transmit its data to the receiver within a single cycle, then allow time for the bus to become tristate before the next master can drive the bus.

The original product plan was to use several gate arrays to implement the module control logic. One gate array would have contained the XMI2 interface logic, and the other would have controlled the system support functions and interface. It became clear early in the project that the external I/O cells of the gate array could not meet the timing mandated by the NDAL specification. Furthermore, we were unable to obtain timely and accurate SPICE models of the gate array output stage, which compounded our general difficulty in determining the design trade-offs for the module.[4]

The use of an external commodity gate array implied another design-related drawback. The normal gate array design process included the submission of our design for chip layout after we had completely verified it for both logical correctness and estimated timing constraints. The timing was estimated since the actual timing could not be known until the ASIC was routed. The gate array routing would be performed by the vendor, and actual delay numbers would be used to verify the design again. This sometimes meant changing logic interconnections to fix timing-related violations, especially if the design team had been aggressive in either the chip cycle time or gate usage. The design would then have to be verified again, and sent back to the vendor where the process was repeated until everything worked at speed.

Consequently, we decided to design the NEXMI ASIC chip using Digital's CMOS-3 standard cell process at the Hudson, MA site. Once the decision had been made to use the internal process, we realized other significant benefits. We believed that we could collapse the design into one package, since we could make use of Digital's chip expertise to create full-custom sections where necessary. We would know early in the design cycle if our assumptions were wrong, since the design flow uses a subchip approach. The approximate size of each subchip is known as soon as the first pass of the structural design is finished.

Another major advantage of using Digital's internal CMOS design process was that it afforded us direct contact with the VLSI design, process, and manufacturing groups. As our design progressed, we had constant communication with the people who wrote and supported the computer-aided design (CAD) tools, the people who had designed the circuits and standard cell library elements, and the people who would eventually fabricate the chips. At any stage of the project, we could determine how to obtain a performance advantage without risk to either chip yield or product reliability. When a tool problem surfaced, or when a tool did not do exactly what we needed, the issue would be immediately addressed and resolved.

By having easy access to the individuals who had detailed knowledge of the circuits, we could attain the maximum possible speed and density from the design. We benefited from the experience of the internal full-custom design community, and profited from access to its complete and accurate SPICE libraries. Consequently, we were certain of how we could obtain a design advantage, yet still allow the chip to perform reliably under worst-case conditions.

The Digital semicustom design process (described in more detail in the section NEXMI Design Process) provides constant feedback on circuit layout. Therefore, our functional and logical timing simulations were always up-to-date with accurate gate and wire delays. By the time we were ready to freeze the design (after we had verified it for logic and timing correctness), only one regression run was needed on a minor change from the last iteration. This approach prevented last-minute surprises, and resulted in a smooth transition from design description, through layout, and into fabrication.

NEXMI Design Issues

During the design of the NEXMI chip, several interesting problems were addressed.

Interblock Control Signal Synchronization The clock that drives the NVAX and NEXMI chips runs at 36 ns, and is not synchronized to the 64-ns clock that drives the XMI2 bus interface. Without any special care, the data that is generated in one clock domain can cause the target latching device outputs to enter a state called "metastable." This state is characterized by oscillations, or an output voltage level that is neither high nor low for an extended period of time. This can cause unreliable system operation.

Traditionally, a synchronizer is provided for the control signals between two different clock domains to allow communication without causing metastability. A synchronizer is a cascaded set of latches, allowing the first latch to go metastable, but characterized so that it settles down before the second latching device is sampled. The drawback to a synchronizer is that it increases the latency of communication due to the cascaded latching devices.

There are no inexpensive and easy remedies for latency of communication when the system goes from idle to active. However, we decided to reduce the synchronizer latency on a busy system. Our method optimizes the case where information is in either the NDAL queue or the XMI2 responder queue, waiting for transmission when the current command has been successfully transmitted. For these situations, we created a set of round-robin synchronizers, with a ring of request and done signals in each direction. While one request/done pair is being serviced, the pipeline overhead for the next pair is hidden by the overlapping synchronizers.

NEXMI Queues For the three major queues in the NEXMI chip, we determined reasonable queue sizes based on our chip space constraints and performance simulations. System testing on the real hardware during our debugging and system integration phase confirmed that our decisions were correct. None of the queues cause performance degradation on an actual running system.

We decided to make the non-write-back and the XMI responder queues into integrated queues rather than have separate queues for each function. Queuing theory shows that a shared resource is better utilized when only one queue is served by the next available resource, rather than having a separate queue for each resource.[5]

Visibility Port A problem that always exists with dense, complex integrated circuits is how to diagnose problems that happen in an actual running system that do not show up during simulation. Although no such problems appeared in the NEXMI chip, the designers wanted to provide visibility to as many internal states as possible. To this end, we created a parallel port to provide visibility to some important internal signals.

Given the size of the design, we could provide only the minimum number of signals for visibility. We tried to predict the internal signals that might help diagnosis and adjustment, such as the internal state machines, interblock control signals, and queue head and tail pointers. We then grouped them into similar units so that related signals were visible within one control group. Internally, the signals were segregated within their boxes, and routed so that they did not adversely affect the top-level chip routing.

NEXMI Design Process

The NEXMI chip contains 250,000 transistors in a die that measures 0.595 inches by 0.586 inches. It is housed in a custom-designed, 339-pin, ceramic pin grid array (PGA). This section describes the NEXMI chip design methodology and the unique CAD tools that were used to design Digital's largest CMOS-3 semicustom chip. (The chip is physically larger and has more transistors than any previous standard cell produced using the Digital semicustom process.) This section also covers many of the tradeoffs made during the chip design, and explains the reasoning behind our decisions.

The design of the NEXMI chip can be characterized by three major phases:

- Behavioral modeling
- Structural design
- Physical chip implementation

The three design phases of Digital's internal CMOS design process significantly overlap each other. Each design stage is explained and analyzed in this section.

Behavioral Modeling Phase

The first major design effort focused on describing the chip functions at a high level of abstraction. This is normally referred to as behavioral or functional modeling, and the design team used Digital's internal hardware description language, DECSIM-BDS, for this task.[6] Functional design sections were allocated to different design engineers, and interfunctional block boundary descriptions were specified.

A behavioral modeling strategy was attractive for many reasons. A well-defined hierarchical partition of the design was quickly realized, and smaller subsections (or subchips) within the chip were identified. Behavioral models of each subchip were initially developed to prove functional correctness. As the design progressed, these subchips were replaced by functionally equivalent gate-level structural models. Using this mixed-mode functional simulation strategy, each designer could progress

at his/her own pace, from behavioral definition to structural implementation, independent of the status of other subchips.

The behavioral modeling effort identified many architectural problems early in the design cycle. Details such as queue sizes and structure, flow control mechanisms, and interblock communication protocols were all emphasized, and each decision yielded valuable information about the feasibility of a single-chip implementation.

Behavioral modeling allowed a functional description of the NEXMI chip to be integrated into a system-level model quickly. This enabled the verification team to write and debug tests early in the design cycle. As each structural subchip was finished, it replaced its previous behavioral counterpart in the verification model, and was tested for functional correctness. This step-wise, integrated approach permitted the tests, models, and logic to be verified incrementally.

One major advantage of our behavioral modeling strategy was that it let the design progress without targeting a specific technology. The designers focused their attention on logical implementation rather than on the technology-specific details, such as the timing and loading constraints imposed by a choice of technology. The Choice of NEXMI Technology section described the process of selecting the CMOS-3 implementation path. The initial behavioral phase of the project allowed some of the design to be finished before the final choice of CMOS technology was made.

Structural Design and Chip Implementation Phases

The next major project design phase involved mapping the behavioral subchip models into their equivalent gate-level structural representations. The design methodology chosen followed directly from the decision to implement NEXMI using Digital's CMOS-3, 1-micrometer, semicustom process. The semicustom process includes a fully specified library of primitive elements, called standard cells, similar to the cells included in a gate array library.

The advantage of the semicustom approach is that it gives the designer full control over the placement and routing of the individual primitives or groups of primitives (subchips) within the chip. This allows the engineer to easily take advantage of special placement for speed-critical paths. Because we were using the internal tool suite and fabrication process, we were also able to take advantage of a wealth of full-custom knowledge provided by our support groups. One of the original reasons for using the internal VLSI process was the tight timing on the NDAL. Therefore, the NEXMI pad ring was custom designed for speed, control, and TTL-level compatibility. Other major handcrafted sections were the dense, multiported queue structures.

To coordinate our large, multiple-person chip design, we used the organized chip design (ORCHID) file management system. The ORCHID system manages the files created by each design tool for every hierarchical subchip. It contains translation tools that convert the schematic data into file formats accepted by the simulation, layout, and verification tools. The system allowed individual designers to work independently on subchips at various stages of development (schematic entry, simulation, floor plans, layout, verification) yet still maintained a coherent hierarchical design database that could be shared by all members of the design team. The idea of a shared database facilitated the reuse of common logic (e.g., counters, parity trees, testability devices), and designers often borrowed from each other to avoid primitive design duplication.

Semicustom Design Flow

Figure 4 shows the semicustom design process and the individual tools that were used during the structural and physical implementation phases of the NEXMI chip design.

ALOE, Digital's in-house graphical editor, was our schematic entry vehicle, and was used in conjunction with the primitive symbols and models from the CMOS-3 standard cell library (SCL3). The tools within the ORCHID system were used to translate schematics into generic wirelists with estimated delays. The schematics were then input to other tools for logic and timing verification. Specific design tools included the internal logic simulator, DECSIM, a timing analysis tool, AUTODLY, and the SPICE circuit simulator.

Automated logic synthesis was used to convert some behavioral models into structural entities. OCCAM, an internal CAD tool, was used to synthesize a gate-level representation of the scattered address decode, and was able to minimize the logic to meet the aggressive bus timing.[7] Controllers embedded within the XMI and SSC subchips were synthesized using SMD2SIM, a tool developed at Digital's Boxboro, MA site for another project. This synthesizer allows large programmed logic array

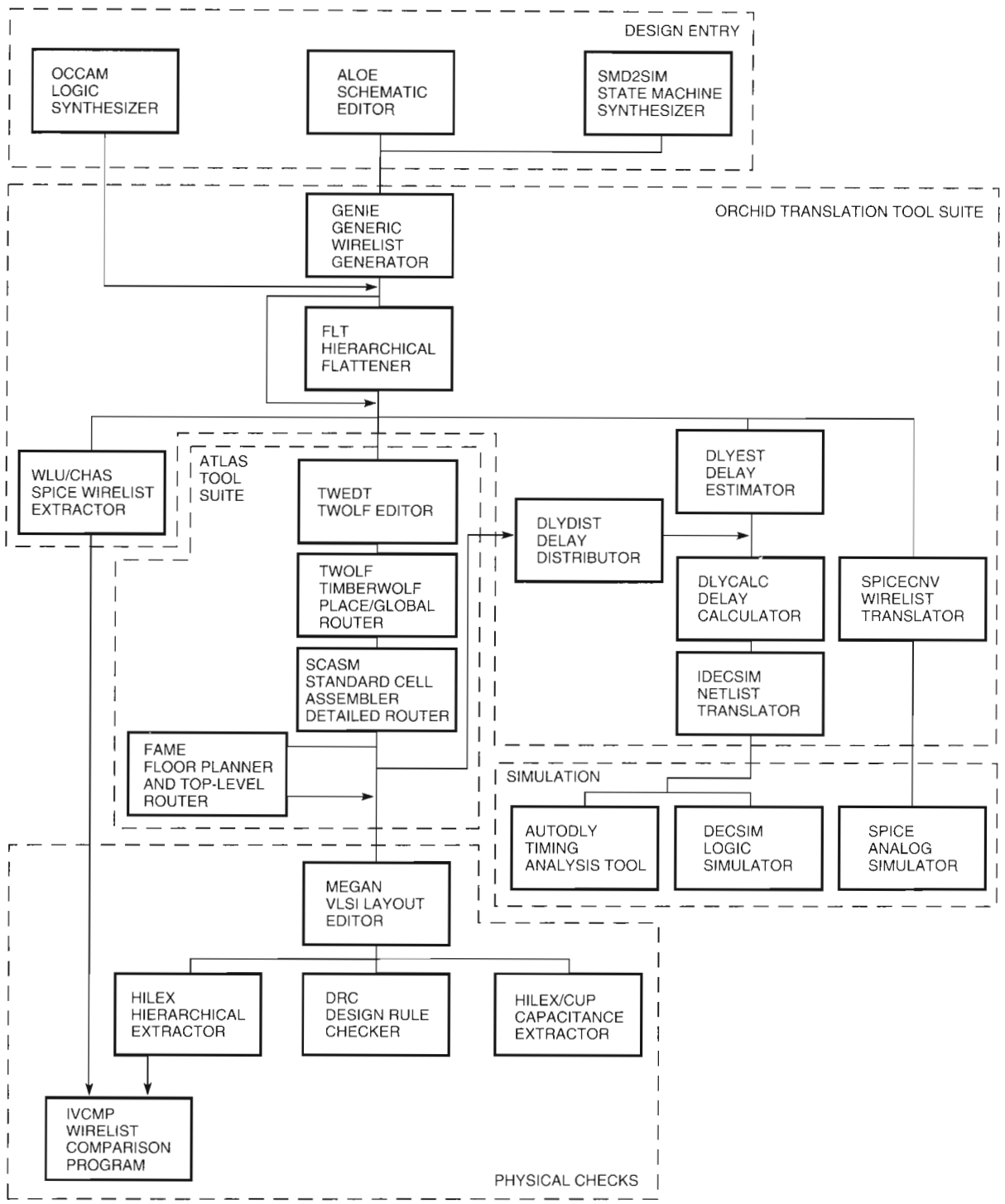


Figure 4 Semicustom Design Flow

(PLA) structures to be realized from a simple LISP-like behavioral description. During the course of the design, controller function changes became easier to maintain using a textual description.

After each gate-level subchip was created, the ATLAS tool suite was used to place and route the standard cells within the subchip. The TWOLF editor (TWEDT) was used to place special cells, such as

clock buffers and timing-critical gates, within the subchips. The rest of the subchip was then placed automatically and globally routed using the TWOLF program, which relies on a simulated annealing placement algorithm.[8] Lastly, the standard cell assembler known as SCASM was used to assign standard cell rows and to complete the detailed routing.[9] SCASM uses both channel routing and "over-the-cell" routing, which reduced the overall size of the subchip by permitting metal routes over the underlying cells. Subchip post-layout timing information was then fed back into the logic and timing verification tools for a more detailed analysis. The ability to analyze post-layout delay information and to iterate through the layout process early in the design phase was crucial to meeting our schedule.

The top-level floor plan was progressing in parallel with the process of subchip placement and routing. FAME, another tool from the ATLAS tool suite, was used for the chip floor plan and top-level routing.[10,11] Information about top-level routing was fed back to the subchip place and route tools to change the basic shapes and subchip boundary pin placement information. This was used to keep inter-subchip routing to a minimum, which in turn reduced overall loading and timing delays. Many successive iterations were necessary to achieve an optimal top-level floor plan that would meet our timing goals and fit onto a single die. Fortunately, much of this work could be done by the designers themselves.

After the final place and route iteration had been done on the the entire chip, an exhaustive set of checks was performed to ensure design integrity and circuit reliability for first-pass silicon. A VLSI design rule checker was run on the individual subchips, then on the entire chip, to verify the layout against the CMOS-3 process design rules. As a precautionary measure, the original chip schematics were extracted to SPICE wirelists using an internal wirelist tool, and another program, called HILEX/CUP, did the same thing using the geometric information in the final layout database. A wirelist comparison program, called IVCMP, compared the two wirelists to ensure that the design we had simulated was the same one we would build.

Finally, a program called XREF used the capacitance from each internal chip node and the topology of the routed interconnection database to predict the cross talk each signal could expect. Changes were made to the signal routing and the sizes of some of the driving transistors, based upon potential problems identified by the XREF program.

Figure 5 is a representation of Digital's CMOS-3 design process. It shows the chip floor plan, along with an example of the schematics used to drive the layout process, and a timing diagram created by the simulation program.

Verification of the VAX 6000 Model 600

Early in the design cycle, our verification team was able to integrate a behavioral chip-level model into a CPU module model and perform system-level testing of the VAX 6000 Model 600. These tests gave the designers timely feedback on the effects of their design decisions on the system as a whole.

An NDAL/XMI2 bus monitor, called the BEMAR, was written to verify the bus activity between the NDAL and XMI2 ports of the chip, and to log valuable cycle information. The NEAT, an NDAL cycle emulator, was also used to create NVAX transactions on the NDAL. It was written to reduce simulation test run times and to ease the test verification process.

Most of the tests written were focused tests, targeting a specific function within the NEXMI chip. However, a random bus exerciser was also written to test unexpected combinations, and was instrumental in catching two design flaws that the focused test cases had missed. In all, over 100 focused tests were written and verified against the simulation model, giving us a high degree of confidence that first-pass silicon would be functional. The quality of the prototype systems is due in large part to this complete verification.

A subset of the functional tests was also used by manufacturing to generate chip test vectors. Test vectors were automatically converted from DECSIM-formatted trace files to Takeda pattern sets through a special program called TEMPEST. Prior to the return of first-pass silicon, the generated pattern sets were converted back into DECSIM format, to verify that the pattern sets would run successfully on the Takeda chip tester. This test pattern generation and simulation process reduced the time needed to debug the test patterns when the NEXMI chips became available.

Module Design Issues

During the design of the VAX 6000 Model 600 processor, many module-related issues were considered. The next section describes some of the more interesting issues that we encountered during the physical module design process. In many cases,

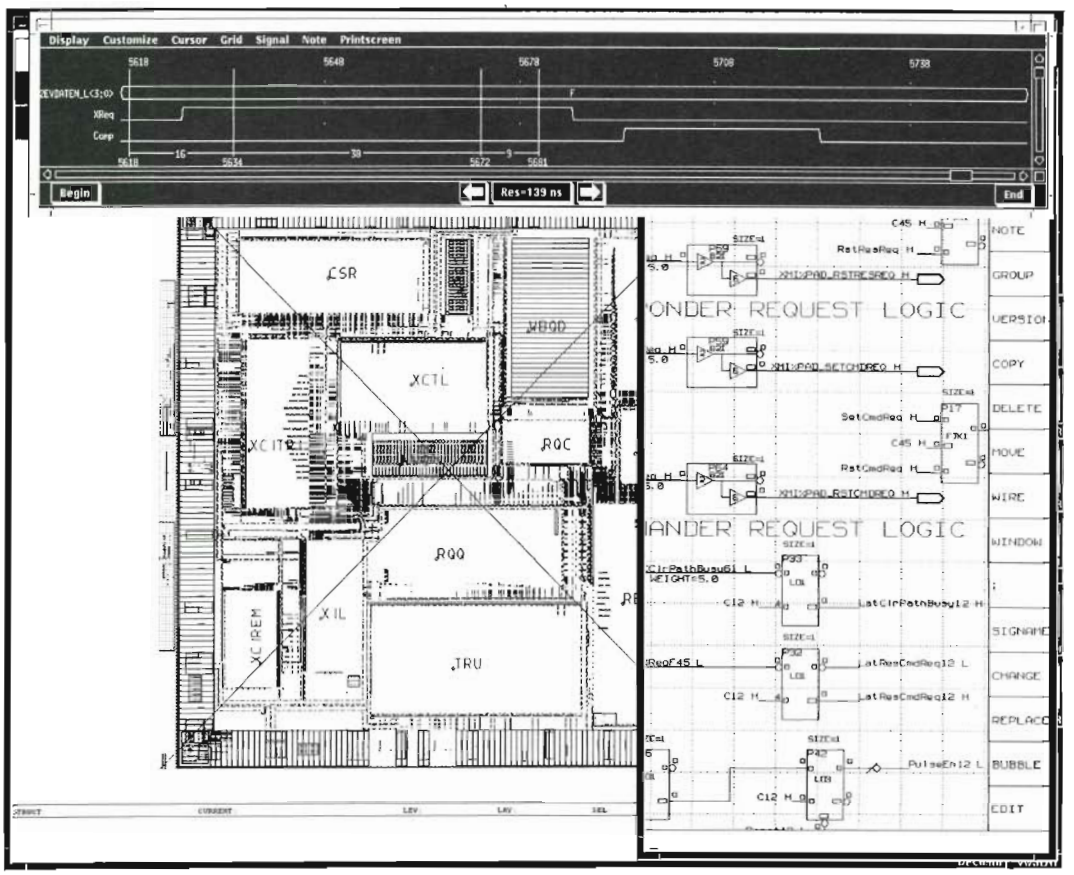


Figure 5 Representation of the CMOS-3 Design Flow

we describe the method that was adopted by the design team to prevent problems through careful planning and analysis.

Backup Cache Size and Speed

We chose to defer the selection of the backup cache size and speed until late in the design cycle. This allowed us to make the decision based upon the pricing and availability of the SRAM devices, and the performance to be gained from combinations of these devices. We also wanted to wait until we knew the final speed of the NVAX CPU. The two most likely cache sizes were 512 kilobytes (KB) and 2 megabytes (MB). We felt that simulation could provide insight into the performance trade-offs, and simulation studies were performed with both sizes to establish approximate performance numbers. We knew, however, that running real modules on a variety of benchmarks under actual workloads was the most accurate way to determine the performance side of the price/performance trade-off.

The footprints for the two SRAM cache chips that represented the cache size trade-offs were different, which made it difficult to create one prototype module to accommodate both sizes. Creating two separate modules to test the different combinations of cache sizes would have burdened the rest of the project, so special surface-mount pads were designed to handle the different SRAM geometries on the same module. Once the cycle time of the NVAX CPU was determined to be 12 ns, we were able to make the final choice on the cache size and SRAM speeds. The final decision was to provide a 2MB cache with 20-ns 256KB by 4 SRAMs for the data and 15-ns 64KB by 4 SRAMs for the tag. This provided the best balance between cost and performance in the Model 600 system.

ROMBUS Decisions

General-purpose computing systems need a core of system functions to supplement the computing

power of the processor. On the VAX 6000 series of computers, this includes:

- ROM to hold the boot, diagnostic, and console code
- EEPROM to hold items such as boot paths and error information
- Console terminal UART
- Time-of-year (TOY) clock
- Battery backed up RAM
- Programmable interval timer
- Time-of-day register
- Input ports to sense external switches and other state
- Output ports to drive module light-emitting diodes (LEDs)
- Reset logic for the module
- Halt detection and arbitration
- Secure console logic

Previous VAX 6000 systems used a system support chip for most of these functions. After we decided to combine as much support logic as possible into the single ASIC device (NEXMI), we had to determine what functions to include inside the chip, and what functions to locate external to the chip. We saw a potential schedule risk if some functions, such as the UART and TOY clock, were placed inside the NEXMI. These functions were relegated to outside the chip since industry-standard, off-the-shelf components were available to perform exactly the functions we needed.

Once we decided to implement the UART and TOY clock outside the NEXMI, and added the normally external ROM, EEPROM, and input/output ports, we found that the NEXMI pin count was higher than we could afford. Since all the support devices were slow, and each one was byte-wide by its nature, we solved the pin problem by creating a single, slow-speed, bidirectional bus, which we called the ROMBUS. Figure 6 is a block diagram of the ROMBUS and its components. The ROMBUS reduced the NEXMI pin count by 40 pins for the same functions.

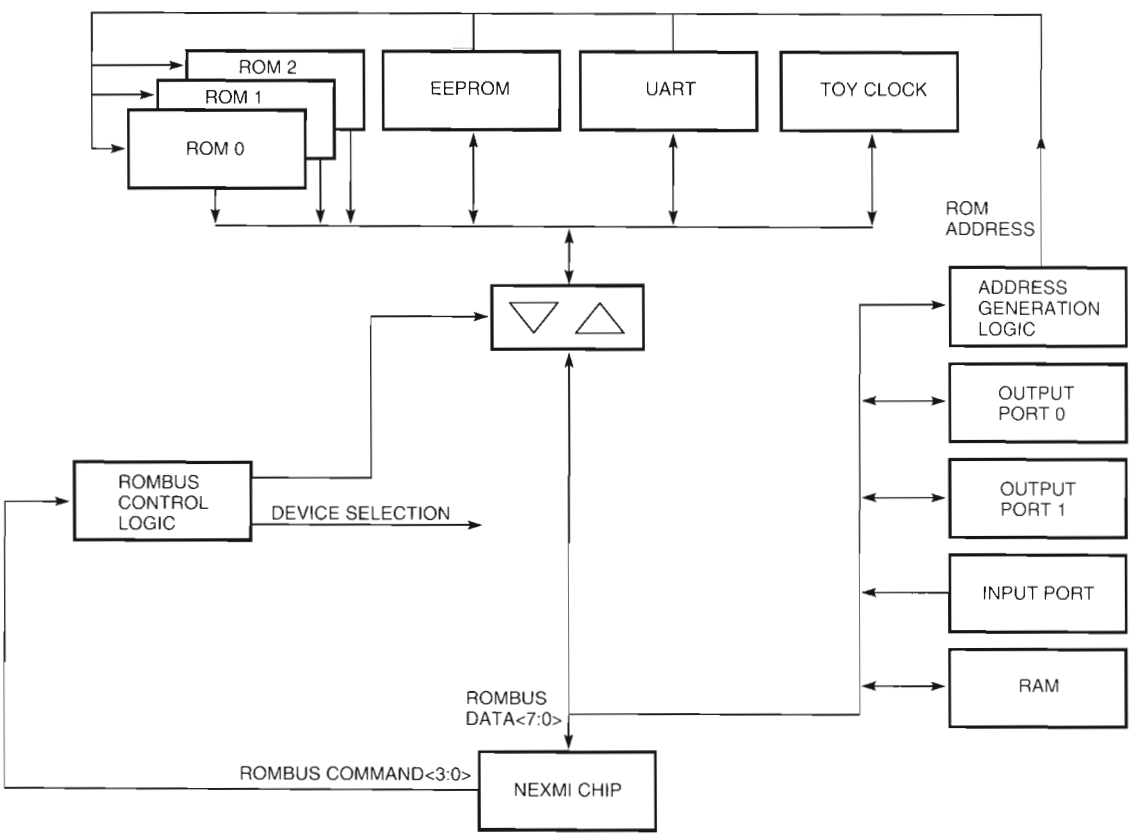


Figure 6 Block Diagram of the ROMBUS

Using the ROMBUS for the UART and TOY clock reduced the risk entailed in designing them inside the semicustom NEXMI ASIC, but the ROMBUS itself was eventually burdened with 12 separate devices. This presented a large capacitive and direct current load to all the components on the bus. The UART and TOY chip in particular were not well suited to this large load. We attempted to find CMOS-equivalent devices for all the TTL components to eliminate the direct current problem, but were unsuccessful. We finally decided to split the bus into two sections, and place all the CMOS low-drive-capability devices on a separate segment. A transceiver connected the two segments.

Given that most of the devices on the ROMBUS were non-Digital components, we had the normal problem of obtaining accurate SPICE models to determine the ROMBUS timing. Extensive lab testing was performed on the devices to characterize their output delays under different capacitive loading conditions. These loading tests helped generate SPICE models for the ROMBUS simulations.

Signal Integrity Considerations

Module signal integrity analysis was one of the most important aspects of the project. A system that includes components running as fast as the NVAX and NEXMI can easily see performance degradation or unreliability if the module signals are not carefully placed, routed, and terminated where necessary. The past experience of the signal integrity team played a major role in the eventual success of the process. Several approaches were taken for this aspect of the design.

Package Selection SPICE simulations were performed on each level of the design. This included the chip, package, module, and backplane. Even though each particular simulation analysis was focused on one aspect of the design, we understood that each individual area affected the entire system. For example, the selection of a package spanned several important levels of design hierarchy. The connections between the die pads and the module signal pins, as well as the connection of the package to the board, needed to be considered; as did the module layout that accompanied each package size and type. One aspect of the signal integrity might show that a particular package type was superior to another, while another aspect might favor a different approach to module component interconnection. Electrical SPICE simulation determined that the performance and reliability of a ceramic through-hole PGA on a 100-mil grid was the best trade-off.

NDAL and Backup Cache Simulation The most important module-level signal integrity analysis was the extensive characterization of the NDAL and backup cache. As the interconnection bus between the NVAX and the NEXMI, the NDAL controls not only the transmission speed between the components, but also the speed at which the NVAX can run (since the NDAL is synchronous to and scales with the NVAX speed). The speed of the backup cache was a performance concern for obvious reasons.

We determined that estimates of the interconnect and routing would be insufficient for our aggressive timing goals. Several trial layouts were performed to provide accurate input for the SPICE simulations. The I/O drivers for both the NVAX and the NEXMI chips were known to be complete and accurate, since they were both designed internally. The timing requirements for reliable operation were also well understood for the same reason. The module-level SPICE simulations provided guidelines about the length and routing rules associated with each high-speed signal trace, such as:

- Daisy-chain routing of all signals
- Clock routing scheme (e.g., matched length and termination)
- Maximum length requirements for each type of network
- Treeing, or ordering, requirements for networks
- Impedance of different networks

These requirements were used as design guidelines. A cross-talk prediction program within the layout tool verified that the coupling between signals was within an acceptable range. After the prototype modules were delivered, measurements were taken of all the critical signals on the module, showing excellent correlation with the SPICE results.

Thermal Management

During the early phases of the module design process, the power dissipation of the NVAX chip was estimated to be as high as 20 watts, though the final figure for the 12-ns component that we shipped with the product was 14 watts. Cooling such a part presented a challenge to the module designers. Since the Model 600 was an upgrade option in the VAX 6000 family, there was no possibility of system modifications to improve the thermal design.

Figure 1, which is a picture of the Model 600 module, shows that the heat sink for the NVAX is larger than the package. The final dimensions of the heat sink are 3.285 inches by 3.285 inches by 0.325 inch, while the PGA package is only 2.2 inches by 2.2 inches. One of our limitations was the maximum component height of 0.420 inch on side 1 for any module in a VAX 6000 backplane. This restriction forced the heat sink to grow wider rather than higher. The NVAX chip was also placed closer to the edge of the board, where the airflow provides maximum cooling. The final size of the NVAX heat sink was a compromise between system requirements, board area, and thermal performance.

Summary

The decision to use Digital's proprietary tool suite and fabrication process was proved correct by the quality of the VAX 6000 Model 600 and its delivery, on schedule, for NVAX CPU debugging and product shipment. Access to accurate information about the components allowed decisions to be made early, and entailed less risk. This advantage, coupled with a seasoned module development team and extensive functional, timing, and circuit simulation resulted in a successful project.

Acknowledgments

The authors would like to acknowledge the following people for their contributions to the VAX 6000 Model 600 Processor: Chuck Benz, Mike Gowan, Chris Houghton, Dave Ives, Keith Johnston, Mike Kagen, Diane Kirkman, Doug Koslow, Bill LaPrade, Don MacKinnon, Lisa Noack, Del Ramey, Sharad Shah, Steve Thierauf, Mike Warren, and Beth Zeranski.

References and Note

1. B. Allison, "An Overview of the VAX 6200 Family of Systems," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 10–18.

2. G. Uhler et al., "The NVAX and NVAX+ High-performance VAX Microprocessors," *Digital Technical Journal,* vol. 4, no. 3 (Summer 1992, this issue): 11–23.

3. *SPEC Newsletter,* vol. 3, no. 4 (December 1991).

4. SPICE is a general-purpose circuit simulator program developed by Lawrence Nagel and Ellis Cohen of the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley.

5. S. Ross, *Introduction to Probability Models* (Orlando, Florida: Academic Press, Inc., 1985).

6. M. Kearney, "DECSIM: A Multi-Level Simulation System for Digital Design," *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers* (1984): 206–209.

7. R. Brayton, ASV, and A. Wang, "MIS: A Multiple-Level Logic Optimization System," *IEEE Transactions on Computer-Aided Design,* vol. CAD-6, no. 6 (November 1987).

8. C. Sechen and A. Sangiovanni-Vincentelli, "Timberwolf 3.2: A New Standard Cell Placement and Global Routing Package," *Proceedings of the 23rd Design Automation Conference* (1986): 432.

9. J. Reed, A. Sangiovanni-Vincentelli, and M. Santamauro, "A New Symbolic Channel Router: YACR2," *IEEE Transactions on Computer-Aided Design,* vol. 4 (July 1985): 208.

10. N. Chen, C. Hsu, and E. Kuh, "The Berkeley Building-Block (BBL) Layout System for VLSI Design," *Digest of Technical Papers, IEEE International Conference on Computer-Aided Design* (1983): 40.

11. M. Marek-Sadowska, "Two-Dimensional Router for Double Layer Layout," *Proceedings of the 22nd Design Automation Conference* (1985): 117.

Jonathan C. Crowell
Kwong-Tak A. Chui
Thomas E. Kopec
Samyojita A. Nadkarni
Dean A. Sovie

Design of the VAX 4000 Model 400, 500, and 600 Systems

The design of Digital's NVAX CPU chip provided the opportunity to bring RISC-class performance to deskside CISC VAX computer systems. The new VAX 4000 Model 400, 500, and 600 low-end systems take full advantage of the performance capabilities of the NVAX microprocessor. The three systems offer from two to four times the performance of the previous top-of-the-line VAX 4000 Model 300 system in the same deskside enclosure. To achieve this increased performance, Digital's systems engineers designed a new high-performance memory controller chip as part of the CPU module, whose basic design is shared by the three systems. In addition, a high-performance memory module and a VLSI bus adapter chip were designed.

The design of Digital's NVAX high-performance microprocessor offered systems engineers the opportunity to design computer systems with significantly improved performance.[1] The project structured to use the NVAX CPU chip to upgrade the VAX 4000 line of low-end deskside computers resulted in a family of three new systems and the associated CPU modules. These systems share a basic CPU module design but offer a range of performance capabilities. This paper first presents the goals of the development project and then describes the architecture, design, and implementation of the resulting new VAX 4000 Model 400, 500, and 600 systems.

Goals of the VAX 4000 Project

Schedule and performance goals were of prime concern to the engineers committed to upgrading the VAX 4000 family of computers. Time-to-market was a key goal of the development project. Consequently, fully qualified systems were ready to be shipped to customers when the NVAX CPU chip was released and available in volume. The initial goals for performance specified that the new systems provide three times the performance of the VAX 4000 Model 300 system. Ultimately, the performance of the NVAX CPU chip exceeded its design goals. As a result, the new top-of-the-line VAX 4000 Model 600 system performance is four times that of the Model 300.

Achieving the performance goals required designing a new, high-performance memory controller chip called the NVAX data and address lines (NDAL) pin bus memory controller (NMC). The objectives were to double the memory bandwidth of the VAX 4000 Model 300 system and to provide a total system memory capacity of 512 megabytes (MB). To support the NMC specifications, a high-performance memory module called the MS690 was designed.[2]

To reduce hardware and software development cost and the risk of failing to meet project schedules, the system design incorporated all existing high-performance I/O adapter chips. These devices include the second-generation Ethernet controller chip (SGEC), the Digital Storage Systems Interconnect (DSSI) shared-host adapter chip (SHAC), the CVAX Q22-bus interface chip (CQBIC), and the system support chip (SSC).[2,3,4] A very large-scale integration (VLSI) bus adapter chip was required to provide CVAX pin (CP) buses to connect these I/O devices.[5] The NDAL-to-CP bus adapter chip (NCA) was designed to meet this need.

The three CPU modules designed to upgrade existing VAX 4000 Model 300 systems retain the

same BA440 system enclosure used for these systems. The upgrade requires that the older MS670 memory module used in the Model 300 be replaced with the new, higher-performance MS690 memory module. The new systems had to support all of the Q-bus option modules that were supported on the VAX 4000 Model 300.

System Overview of the VAX 4000 Models 400, 500, and 600

The BA440 system enclosure shown in Figure 1 supports the VAX 4000 Models 300, 400, 500, and 600. This pedestal enclosure was designed to operate in an open office environment. To allow the systems to operate quietly, the cooling fans are speed controlled, based on the ambient temperature. The enclosure power supply provides 644 watts of direct current from a standard 15-ampere wall circuit. The system was designed and qualified to operate in an environment with a temperature range of from 10 to 40 degrees Celsius.

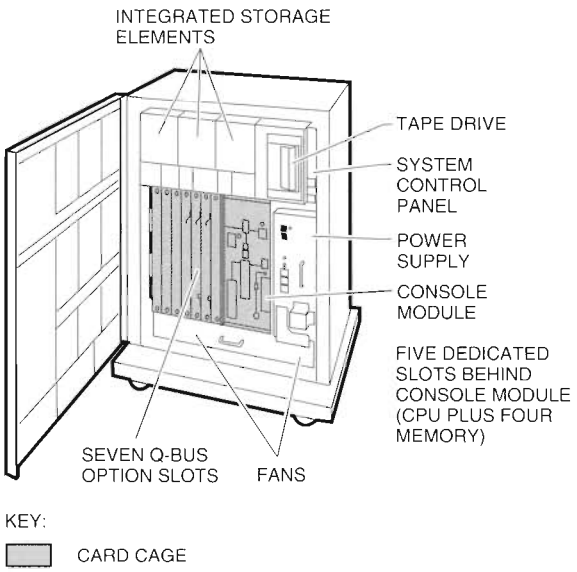


Figure 1 BA440 System Enclosure

The new CPU modules, differentiated only by the part numbers KA675, KA680 and KA690, are utilized as the engines for the VAX 4000 Models 400, 500, and 600, respectively. All three CPU modules, henceforth referred to as the CPU module, provide the same I/O functionality, including two DSSI buses, a thick-wire and ThinWire Ethernet adapter, a Q-bus adapter, and the console serial line. The CPU module performance is 16, 24, and 32 times the performance of the well-known VAX-11/780 system, for the three new VAX 4000 systems, respectively. The CPU cycle clock speed and cache sizes determine the product performance, as discussed in the section CPU-cache Subsystem.

The backplane in the system enclosure provides the signal interconnection and power distribution between system components. There are connectors and slots for the CPU module, four slots for MS690 memory modules, and seven Q-bus slots. The CPU module has a 270-pin connector that receives the module power and connects the CPU to the NVAX memory interconnect (NMI) bus, the system DSSI bus, and the Q-bus. The backplane was modified to support the wider 72-bit data path of the new MS690 memory modules. This new backplane was phased into the BA440, enabling most VAX 4000 Model 300 systems to be upgraded without requiring a backplane change.

The system enclosure supports up to four DSSI or small computer system interface (SCSI) tape integrated storage elements (ISEs). These cableless bricks support either one 5.25-inch, full-height drive or two 3.5-inch drives. The ISEs are available in variants that support the 2-gigabyte (GB), RF73 DSSI disk drive and dual 85MB, RF35 DSSI disk drives. The single-system pedestal can support six RF35 devices and a tape drive, providing 4.8GB of storage for applications that require high I/O rates. This RF35 configuration can provide over 360 queued I/Os per second for random I/Os. If RF73 drives are used, the single-system box can provide 8GB of storage.

There are several ways to expand the base VAX 4000 system; the most common way is to expand to another DSSI-based system and create a two- or three-node DSSI VAXcluster. The Q-bus in the VAX 4000 system can be expanded to provide 10 additional Q-bus slots to each system using the B213A Q-bus expansion enclosure. The DSSI expansion enclosures together with the Q-bus DSSI adapter (KFQSA) can expand the total available disk storage to 28 DSSI disks. Using the RF73 disk allows up to 56GB of disk storage.

CPU Module

The CPU module common to the three new VAX 4000 systems is based on a highly integrated CPU and I/O system built on the single 21.6-by-26.7-centimeter (8.5-by-10.5-inch) module shown in

Figures 2 and 3. The CPU module printed wiring board (PWB) consists of the following subsystems: a central processor and its associated three-level cache; a pin bus, bus adapter, and memory controller; and an I/O system with integrated controllers for DSSI and Ethernet buses. The CPU module also contains a CQBIC and 512KB of field erasable programmable read-only memory (FEPROM) for console code.

CPU-cache Subsystem

The CPU-cache subsystem is built around the single-chip NVAX CPU, which provides a three-level cache architecture. The first two levels of cache, which are contained on the chip, include a 2KB virtually addressed instruction cache and an 8KB physically addressed instruction and data cache. The third level of cache, the backup cache, is constructed using static random-access memories (SRAMs) on the module and is completely controlled by the NVAX CPU chip. The backup cache was designed to support a CPU cycle time as low as 10 nanoseconds (ns) with a slip cycle, i.e., a two-cycle read (20 ns) using 8-ns SRAMs. This write-back caching architecture significantly reduces the demands on main memory by caching both reads and writes without the need for a memory access. On all previous VAX 4000 systems, the caches required that all write operations continue through to main memory, i.e., write through.

The NVAX CPU is clocked by a differential emitter-coupled logic (ECL) surface acoustic wave oscillator. This oscillator runs at 250 megahertz (MHz) (16-ns cycle time), 286 MHz (14-ns cycle time), or 333 MHz (12-ns cycle time) on the KA675, KA680, and KA690 CPU modules, respectively. The NVAX chip produces a four-phase internal clock directly from this input and generates system clocks at one-third the internal clock rate.

The new CPU module design supports either a 128-kilobyte (KB) or a 512KB backup cache. (512KB for the KA690 module and 128KB for the KA680 and KA675.) The tag store for the two cache sizes can be

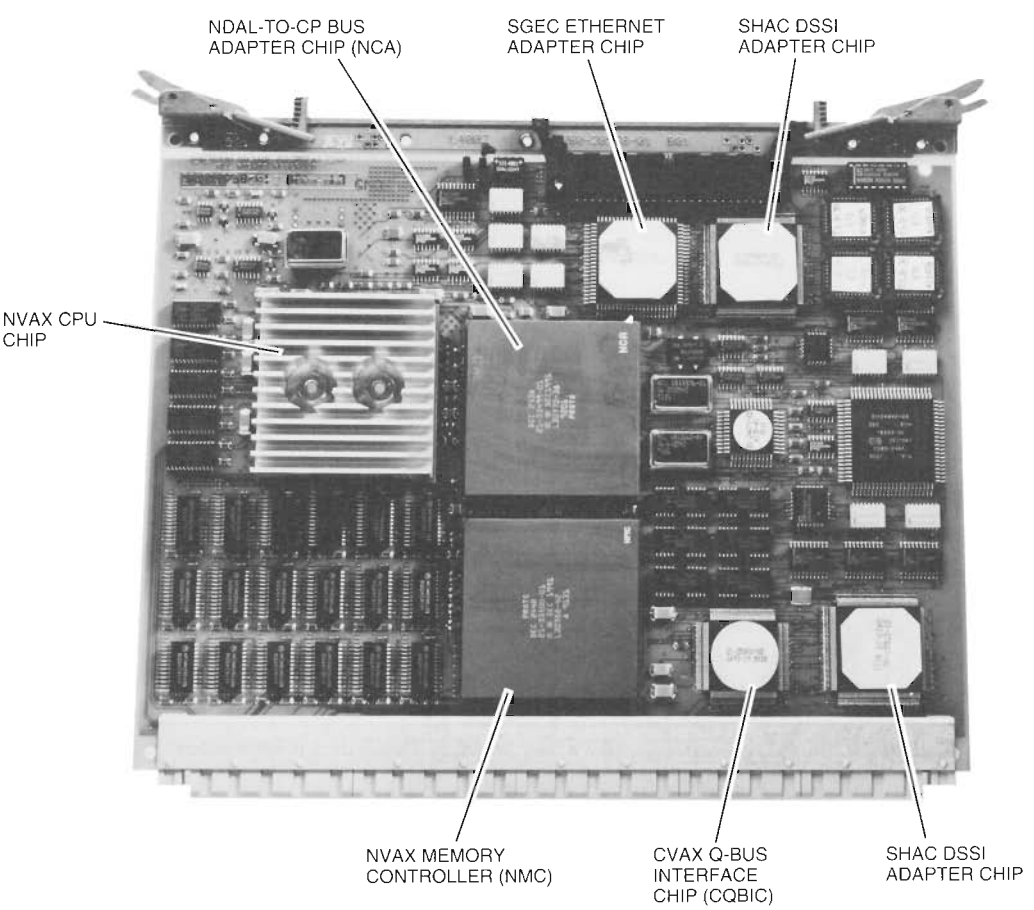


Figure 2 CPU Module

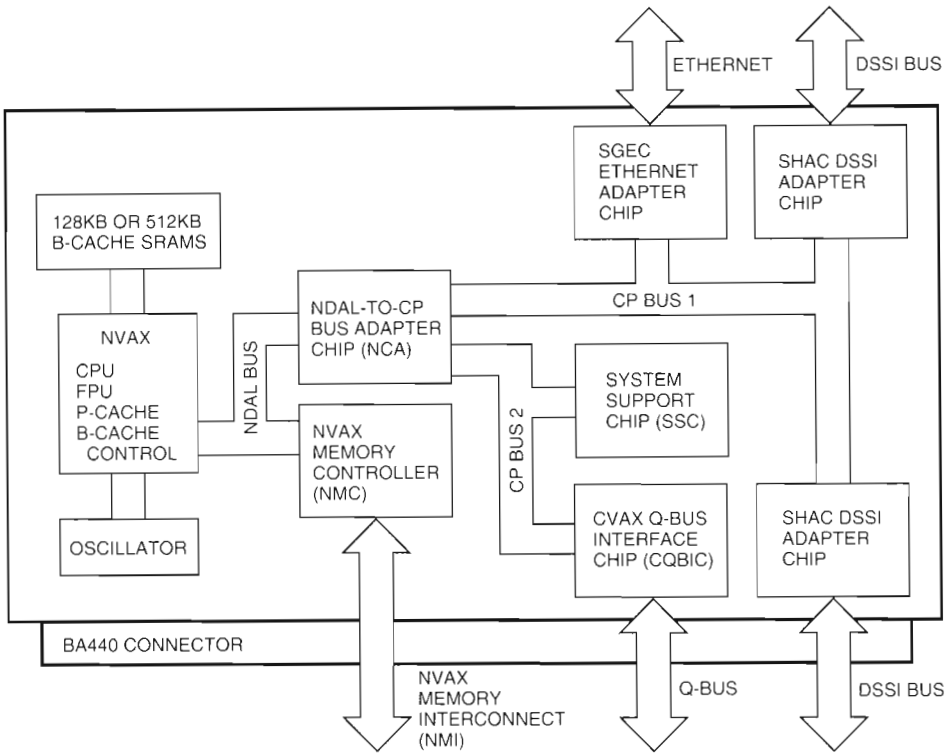


Figure 3 Block Diagram of the CPU Module

constructed from SRAMs that have the same 24-pin package with compatible pinouts. This packaging makes it easy to design the PWB to support either cache. However, the data store, which is constructed from parts whose packages have incompatible pinouts, required a special dual footprint design to accommodate either 16K-by-4K or 64K-by-4K SRAMs. This footprint is designed with the decoupling capacitors and series edge-limiting resistors carefully placed in the outline of the footprint to allow a dense, geometric packing of the 18 RAMs necessary for the data store.

The backup cache is a write-back cache with memory coherence maintained through a directory-based broadcast coherence protocol.[1] When the NVAX CPU needs to write data to memory, the data is first transferred into the backup cache with a request for write privilege command. Once a cache row is stored in the cache as "written," any direct memory access (DMA) read to that memory address of displacement of that cache row will result in a release write privilege transaction, even if the data was never actually written. The cache controller inside the NVAX CPU is responsible for all activities related to cache maintenance.

NDAL Interconnect, Memory Controller, and Adapter

The NDAL bus is a synchronous, multiplexed address and data, pended interconnect. Each device on the NDAL bus may be a commander (request data transfer), a responder (respond to commander requests), or both. In the new VAX 4000 systems, the NVAX CPU chip is only a commander, the NMC is only a responder, and the NCA I/O adapter is both a commander and a responder.

Arbitration of the NDAL interconnect is performed by the NMC, which is also the default master responsible for driving valid no-operation bus cycles when there is no master activity. The NVAX CPU is responsible for watching all NDAL traffic and performing any invalidates or write backs of primary and backup cache data required to maintain cache coherence with memory.

I/O Buses

The CPU module uses a custom third-generation complementary metal-oxide semiconductor (CMOS-3) process I/O adapter (the NCA chip) to interface between the NDAL bus and a pair of 32-bit

CP buses. Two CP buses are used to prevent long response latencies on the Q-bus from interfering with buffer management on the Ethernet interface. One CP bus, CP1, operates under the synchronous CP bus protocol, and the other, CP2, uses the asynchronous protocol. The faster peripherals, the Ethernet and the two DSSI adapter chips, reside on CP1 and can take advantage of optimizations in the NCA to reach a peak bus bandwidth of 33MB per second (MB/s). CP2 has only one peripheral DMA master, i.e., the CQBIC, which also connects the SSC and the console FEPROM to the system. The NCA acts as a master on both CP1 and CP2. Since only one of the buses is asynchronous, the system uses only one CP bus clock (CCLK) chip for signal synchronization and CP clock distribution.

The two CP buses share the same clocks. Consequently, arbitration is performed by a single programmable sequencer, which serves both buses. The sequencer is clocked at 35 ns (KA680 and KA690) or 40 ns (KA675); this is one-half the CP cycle time. The arbitration for CP2 is a simple two-priority scheme with the CQBIC at the higher priority. When more than one master is requesting the bus, the minimum DMA request deassertion times on both the CQBIC and the NCA effectively make this scheme behave like a round-robin arbiter. The internal state does not have to keep track of the previous master. No special treatment is required for lock cycles because the CQBIC will never perform a lock on behalf of the Q-bus.

Signal Integrity

Signal integrity work began very early in the project, and the effort was a close collaboration between the CPU module design team, the design teams for the three VLSI devices, and the VAX 6000 Model 600 CPU module design team. Because some CPU module team members had experience with designing CP bus modules, the signal integrity issues on the CP buses were generally well understood. The CP bus data lines were routed with only a length constraint. The control signals on CP1 required significant analysis and SPICE modeling to meet both settling time and waveform requirements.

The most critical signals in the backup cache are the output enable and write enable signals. The output enable deasserting edge must be transitioned quickly to avoid tri-state contention on the cache data lines. The write enable signal must be perfectly monotonic through the threshold region, because it is an edge-sensitive signal. Both of these requirements were met by using a strong driver in the NVAX chip and a parallel R-C termination at the far end of two of the three stubs. The R-C terminations absorb some of the incident energy, reducing the reflections to an acceptable amount and allowing incident-wave switching without the reflected wave reentering the threshold region.

The backup cache data store was designed with strong drivers and incident-wave switching on the address lines. The routing of a representative cache address signal is shown in Figure 4. The stubs were arranged in such a way that the unavoidable reflection from the far end of the lines was reduced by a partial reflection from the center junction. Thus, signals traverse the threshold region fairly cleanly and settle rapidly outside the threshold region, i.e., approximately 3 ns elapse from the beginning of the transition at the driver until the time when a valid signal arrives at the receiver.

Printed Wiring Board Physical Design

The NVAX CPU chip draws several amperes of current from the 3.3-volt power supply. This current draw has significant high-frequency components. The integrated decoupling capacitor on the NVAX die helps eliminate some of the high-frequency current pulses, but much of this current must be supplied by the module-level decoupling capacitors.[1]

Charge stored on the module in these decoupling capacitors supplies this current. Any inductance in the path of the current reduces the effectiveness of the capacitors by limiting the rate-of-change of the current. In addition, the larger the physical area enclosed by the current path, the more radio frequency (RF) energy will be radiated into space that must be contained by the enclosure in order to meet regulatory radiation requirements. These two issues led to the exploration of how to minimize both the inductance of the decoupling path and the physical area of the RF current spread.

The internal PWB standard, as it existed when the new CPU module was being designed, required a minimum of 25 mils (0.025 inch) of surface etch on any device before a via could be dropped into an inner layer. Traditionally, the inductance of this connection was reduced by using a wide (i.e., 25-mil) etch for this connection. The inductance of surface etch on the module lay-up used on the new CPU module (calculated with two-dimensional transmission line [TDTL]) is shown in Table 1.

Table 1 provides the data to calculate the total inductance of a pair of 25-by-25-mil etch segments,

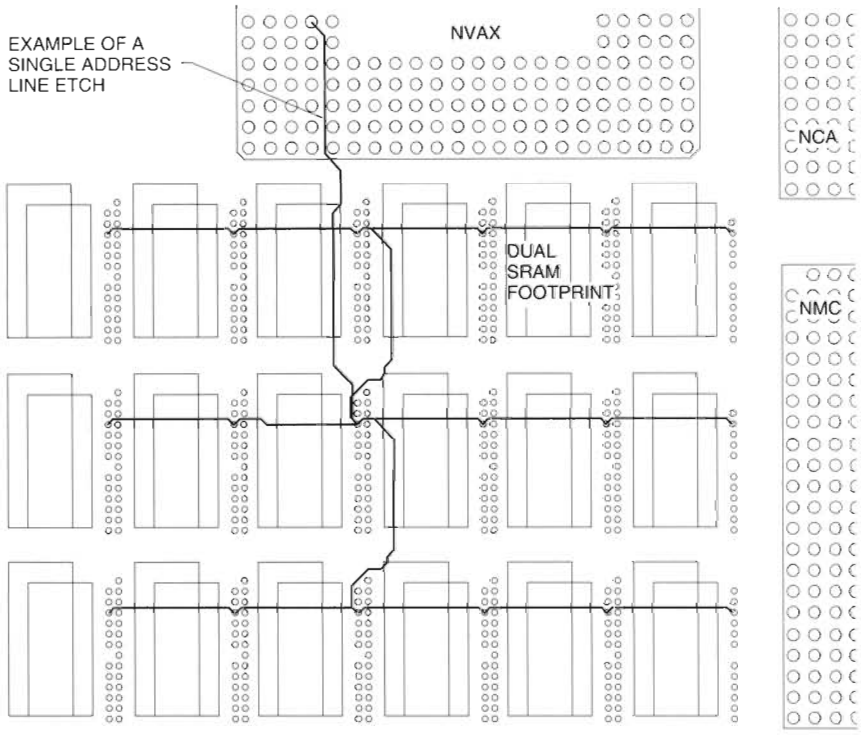


Figure 4 Backup Cache Treeing

i.e., approximately 0.4 nanohenrys (nH). (This measurement is approximate, due to the short dimensions of the segments.) The effective series inductance of the high-quality, 1,000-picofarad (pF) RF capacitor used on the CPU module is approximately 1 nH, including the inductance of the vias connecting the capacitors to the power planes. The reactance as a function of frequency for the inductive component of this decoupling system is shown in Table 2.

Because the resulting impedance is still quite high at upper frequencies, multiple capacitors are used in parallel. The 1,000-pF capacitors are chosen from two different case styles to ensure that the parasitic inductance of the capacitors is not identical for all the high-frequency decoupling

Table 1 Inductance of a Surface Etch

Etch Width	Inductance
10 mils	12.2 nH/in
25 mils	8.1 nH/in

Note: This table represents the results of the inductance of a 0.5-ounce copper strip on a 14-mil FR4 epoxy glass laminate dielectric over a ground plane.

Table 2 Reactance as a Function of Frequency for Decoupling with and without Dispersion Etch

Frequency (MHz)	Reactance	
	With Dispersion (Ohms)	Without Dispersion (Ohms)
100	0.89	0.63
200	1.76	1.26
400	3.52	2.51
600	5.28	3.77

capacitors. This method of selection staggers the frequency of the parasitic resonances.

Cycle Design Goal and Testing

Although the original design goal for the CPU module was a 12-ns CPU cycle time, as knowledge about Digital's fourth-generation complementary metal-oxide semiconductor (CMOS-4) process increased, the design teams investigated the critical paths for a 10-ns operation. At this time, the CPU module had not been routed, so a 10-ns cycle time was set as the layout goal. The cache loop layout resulted in a measured requirement that RAMs have an access time of approximately 8.5 ns to meet worst-case timing with no added slip cycles.

RAMs meeting this specification were not readily available when the CPU module was designed. However, several vendors are beginning to ship devices at this speed today. The NDAL data lines were capable of running at the 30-ns NDAL cycle that is generated with a 10-ns CPU clock. The point-to-point NDAL arbitration signals are the tightest timing path on the NDAL. The combination of analysis by the NMC team and a careful hand-routing of these signals by the module team allowed appropriate NMC speed binning (i.e., sorting the chips based on correct operation at the fastest possible speed) to meet the timing requirements for a 30-ns NDAL cycle goal.

Very few NVAX CPU chips were available that would function at 10 ns over the full range of voltage and temperature. However, empirical signal-delay measurements and limited-range module testing have proven that the NMC and the NCA are ready to operate on the CPU module at this speed. An NVAX CPU that functions at this 10-ns speed can operate the cache with no slip cycles using the faster SRAMs. Future products may be based on an NVAX running at a 10-ns cycle, as sufficient yields at this speed bin are reached.

Module Testing

The module and chip teams considered more than one approach when determining what module-level testability features to implement in the VLSI devices Digital was building for VAX 4000 Model 500 computers. The scan-based Test Access and Boundary Scan Architecture (JTAG) proposal was coming into its own, and the teams desired to follow that specification, if scan-based test features were to be used.[6] However, no other devices on the module would have scan capabilities. Thus, the overall module test strategy could not be based entirely on the JTAG specification.

As the teams reviewed the overall module design, certain issues appeared to show promise for the application of a scan-based test:

1. The automatic test equipment (ATE) pin density was likely to be very high in the areas of the three 339-pin pin grid arrays (PGAs). Reducing this high density would improve the reliability of the test fixturing.
2. Previous experience with module manufacture in Digital's plants showed that the risk for solder defects would be high in the cache area, because of the J-lead SRAMs. A fast way to isolate these problems would help reduce debug time.
3. Providing at least a driver or a receiver for use by the JTAG scan ring would eliminate the need to use continuity structures to verify bonding and solder integrity on the VLSI parts.

Eventually, the module and chip teams settled on a subset implementation of the JTAG scan-based test. The NVAX CPU chip implements scan latches on all data and control pads (input and output, in the case of bidirectional pads). Thus, the cache SRAMs can be tested using only the JTAG port, and the NVAX CPU can act as the driver for scan testing of the NDAL interface. The NMC and the NCA implement receive-only scan, so that the NDAL interface could be tested with no ATE pins required. The external tester was able to test the remaining pins solely by driving signals that could be scanned out of the pad latches. This subset implementation provided the same module-level coverage as would have been possible using a full JTAG implementation. In addition, the implementation removed some design obstacles that were causing implementation problems in the chips.

The ability to use scan-based testing is advantageous to the manufacturing process in the following two areas:

1. The scan tester can find open circuit defects in the cache area where the bed-of-nails tester could not resolve whether the problem was a fixture contact problem or an actual open circuit.
2. The ability to create "virtual test points" on scanned nets has allowed the test coverage of the bed-of-nails tester to be expanded without having to purchase an expensive tester upgrade.

Unfortunately, the module and chip teams' previous experience with scan-based testing at the module level had been spotty at best. The ability of this test to reduce the pin density, therefore, was not used to full advantage in the problem areas under the large PGA devices. Based on the experience testing the CPU module for the three new VAX 4000 systems, follow-on products have been able to use this testing feature to good advantage. The teams now have a firm base of experience on which to base future test strategies.

The NVAX Memory Controller

The NMC is a 520-by-500-mil custom chip fabricated using Digital's 1-micrometer CMOS-3 process and contains 148,000 transistors packaged in a 339-pin PGA. The NMC provides the interface between the NDAL bus and up to 512MB of main memory by means of the NMI. The NDAL bus supports three other nodes on the NDAL—the NVAX CPU and up to two I/O adapters (IO1 and IO2). In the new VAX 4000 systems, the NCA serves as both I/O nodes on the bus. The NMC contains the arbiter for the NDAL and also helps the NVAX CPU maintain cache-memory coherency in the system by interfacing with a separate O-bit memory.

The NMI can operate with either a 32- or 64-bit-wide data path and supports single error correction, double error detection, and nibble error detection, and runs synchronous with the NDAL clock. The NMI timing scales with the NDAL clock cycle time.

In this section, we describe the architecture of the NMC, the objectives of the NMC project, and the results of the effort.

NMC Architecture

As shown in Figure 5, the NMC is partitioned into six major sections: the NDAL arbiter, the NDAL interface, the transaction handler, control and status registers (CSRs), the memory interface, and the O-bit interface. The NMC responds to all memory space addresses when NDAL address bit 29 is equal to 0 and responds to I/O space addresses in its allocated range, i.e., 2101 0000 .. 2101 FFFF (hexadecimal).

The NDAL arbiter gives highest priority to the NMC for returning read data. The two I/O nodes have second priority; their requests are handled in a round-robin fashion. The CPU has lowest priority.

The NDAL interface consists of an input section and an output section. The input section monitors the NDAL for a new transaction every cycle. A valid transaction that has been decoded by the NMC is put into one of four transaction queues (INQUEUEs). There is one queue for each of the NDAL nodes: CPU, IO1, IO2, and the fourth, which stores release write privilege transactions. Commander nodes on the NDAL initiate release write privilege transactions to release the write privilege of blocks in memory. The NMC must

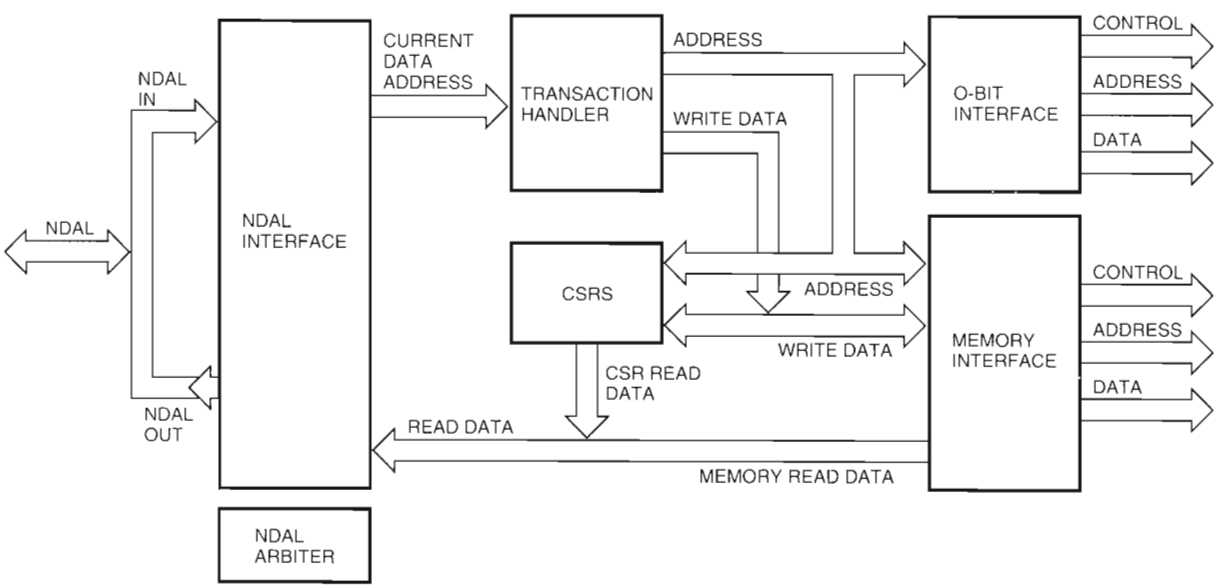


Figure 5 NVAX Memory Controller Block Diagram

accept a release transaction from a node, irrespective of the state of its INQUEUE; otherwise, there is a potential for deadlock. Consequently, the NMC has a separate queue for release write privilege transactions. The output section of the NDAL interface buffers up to six quadwords (i.e., six groups of four contiguous 16-bit words for a total of 384 bits) of read data that must be returned to the NDAL. In a normal functioning system, a buffer depth of six quadwords together with an arbitration scheme that gives the NMC the highest priority will never result in a full output queue. Therefore, there was no need to check for a full queue and to stall while loading the queue.

The transaction handler arbitrates between the four INQUEUEs and stores selected transactions in a current transaction buffer. The current transaction buffer serves as a pipeline stage; this buffer allows the corresponding INQUEUE to be loaded with the next transaction while the current transaction is being serviced. The NMC can service back-to-back transactions with no stall cycles on the NMI.

The CSR section of the NMC contains memory configuration registers, error status registers, and mode and diagnostic registers.

The memory interface contains the data path, address path, and control for up to four memory modules on the NMI. The data path contains all the error correction and detection logic. The address path contains the row and column address multiplexers and a refresh address counter. The control is provided by a state machine that can perform multitransfer read operations, multitransfer write operations, and read-modify-write operations.

The O-bit interface directly controls the O-bit dynamic random-access memories (DRAMs), which are housed on the CPU module. For every memory transaction, the NMC reads the corresponding O-bit in parallel with the memory access. If the block of memory is written, the memory transaction is aborted until the corresponding release transaction is received by the NMC. If the block is unwritten, the memory transaction is allowed to complete. Initiating the memory transaction in parallel with the O-bit access reduces the transaction latency on transactions that are not written. Since most memory accesses are to unwritten locations, using this scheme improves memory performance considerably. In addition, the system design engineers were able to use inexpensive DRAMs to implement the O-bit memory instead of faster, more expensive SRAMs.

NMC Project Objective and Results

The NMC project objective was to create a high-performance memory design that would be compatible with the VAX 4000 Model 300 memory subsystem and could provide two to three times the performance of that subsystem. (The VAX 4000 Model 300 has a bandwidth of 47.4MB/s, at a cycle time of 28 ns, using 100-ns DRAMs with a 32-bit memory data bus.) This goal was achieved by using a 64-bit memory data bus and an interconnect that operates at a cycle time as low as 36 ns, using 100-ns DRAMs, and at an NDAL cycle as low as 30 ns, using 80-ns DRAMs. The asymptotic bandwidth on the NMI using the 100-ns DRAM technology and a 64-bit data path is 111.11MB/s, i.e., 2.3 times the bandwidth of the VAX 4000 Model 300. Using faster 80-ns DRAMs, the bandwidth is 133.33MB/s, i.e., 2.8 times the bandwidth of the VAX 4000 Model 300.

The NMC interface is efficient from the moment it receives a transaction on the NDAL until it starts a transaction on the NMI. This timing path was extremely tight and results in real memory read bandwidth of 63.6MB/s and 76.32MB/s at 36-ns and 30-ns cycle times, respectively.

The NMC chip was designed to meet an NDAL cycle time of 36 ns, which made the timing very critical. Most of the NMC chips produced can exceed this goal and will run at an NDAL cycle time of 30 ns. Future designs based on the 10-ns NVAX chips will require this cycle time.

To meet the performance goal, we chose to use a 64-bit memory interface. However, achieving compatibility with the Model 300 memory modules presented a challenge with respect to the ECC generation and checking mechanism. A simple approach would have been to include two separate ECC trees, one for 64-bit operation and the other for 32-bit operation. This design would have been very area-intensive, so we chose 64-bit ECC code such that 32-bit ECC was a subset. 64-bit ECC requires eight check bits, and 32-bit ECC requires seven check bits. In our 64-bit code, the eighth check bit depends solely on the upper 32 bits. In 32-bit mode, we force the upper 32-bits to a known value; therefore, that check bit is always a fixed value in 32-bit mode.

The VAX 4000 systems do not allow the use of 32-bit memory modules, because it is difficult to meet Q-bus latency requirements with the slower memory. This system constraint indirectly affected the NMC. The CQBIC and SGEC devices, for example, had stringent low latency requirements. The Q-bus

latency problem had to be solved without causing SGEC latency problems. Thus, the latency issue was addressed in the following way:

- The NMC has a mode that indicates whether or not the Q-bus is present in the system. During this mode, the transaction handler gives the Q-bus node (IO2) the highest priority. However, to keep the SGEC latency within required limits, the transaction handler must service transactions from the IO1 node at strategic times.
- To minimize the latency seen by any one node, the three NDAL nodes require separate queues. The simplest implementation of the NDAL input interface would have been to have two queues, one for release write privilege transactions and one for all other transactions. Thus, preserving the order of NDAL transactions would have been very easy. With three queues, it is necessary to compare queue addresses to preserve transaction ordering.
- The NMC combines the CPU request for write privilege, the DMA read, and the DMA write into a single transaction before retiring the data to memory. This optimization reduces the latency of written transactions.

Although the features that were added to the NMC chip to reduce Q-bus and SGEC latency increased the complexity of the chip, these features successfully keep the Q-bus latency below 8 microseconds.

NDAL–to–CP Bus Adapter Chip

NCA is a full-custom, high-performance I/O controller chip that provides the electrical and functional interface between the 64-bit NDAL bus and the 32-bit CP bus. In the new VAX 4000 systems, the NDAL supports three chips: the NVAX CPU, the NCA, and the NMC. On the CP bus, the NCA supports the SHAC, SGEC, CQBIC, and SSC chips. The NCA is fabricated in Digital's CMOS-3 process, contains 155,000 transistors, and is packaged in a 339-pin PGA. The design goals for the NCA project were high quality, improved performance, optimized time-to-market, and leveraged use of existing CP bus chips. The NCA team achieved all design goals and completed the project by the scheduled manufacture release date.

NCA Architecture Overview and Partitioning

Although the original concept of the NCA was motivated by a memory and I/O controller chip called the G chip (used in the VAX 4000 Model 300 systems), the NCA chip was a new design. To optimize the DMA to memory bandwidth and the bus access latency, the NCA provides the two CP bus interface ports (CP1 and CP2, mentioned previously), which operate independently. This strategy has three advantages. First, the Q-bus adapter and the system read-only memory (ROM)/console are connected to the CP bus separately from the Ethernet and the mass storage devices. This arrangement allows the system to tune and optimize throughput on the Ethernet and mass storage devices without degrading the bus access latency seen on the Q-bus. Second, the loading on the two CP buses is reduced. Therefore, each bus can operate at a higher frequency and without external buffers; this also saves module area. Third, the dual-bus structure allows the use of a simpler bus arbitration scheme.

In addition to using the dual-bus strategy, the NCA uses write buffer and read prefetch transactions to allow DMA devices, particularly the SHAC and SGEC chips, to operate efficiently in double octaword mode, where a two-octaword (32-byte) burst of data is transferred within a single bus grant. For write transactions, the NCA buffers up to two octawords of data. Thus, the bus can operate without stalling, while the NCA arbitrates for the NDAL bus for the buffered write transactions. For read transactions, the NCA contains a hexword-size (32-byte) prefetch buffer. Whereas the maximum burst length is only an octaword on the CP bus, the NCA requests up to a hexword of data during DMA memory read operations. The extra data is stored in the prefetch buffer and is immediately available if the subsequent CP bus read transaction targets the same address as the prefetched data. For NVAX initiated I/O, up to four operations can be buffered simultaneously.

The NCA is partitioned into four major sections: the NDAL, CP1, CP2, and registers. The NDAL interface and each CP bus interface contain the master and slave sequencer and controls for the corresponding bus. The NCA chip also has I/O queues and an internal arbiter to select operations from CP1, CP2, and NCA register read transactions to the NDAL bus, based on a predetermined priority.

The register section contains the control and status registers and the interval clock timer registers. The interval clock is a software-programmable timer used by the operating system to account for time-dependent events.

The NCA supports parity check and detection on both the NDAL and the CP buses. The NCA also

supports all interrupts defined by the NDAL and CP bus protocols for other types of error conditions. When an error occurs, an error status bit is set in the NCA error status register. Depending on the type of error, an address may be available for diagnostics. The NCA also provides a mechanism to force a parity error condition on any of the buses to help debug the interrupt routines of the operating system software.

Q-bus Latency Support

To reduce latency seen by the Q-bus devices, the NCA provides special logic to gain priority from the NDAL arbiter. The NCA informs the arbiter of the imminent Q-bus operation, for which latency is a concern. When a Q-bus is present in the systems, the NCA is programmed to use the two IDs mode on the NDAL and to enable the Q-bus present bit of the control register. Upon detection of the Q-bus "map" read transaction on the CP bus, the NCA immediately asserts a signal to the NDAL arbiter. The arbiter will not grant the bus to other devices until after the Q-bus read transaction is accepted by the NMC or until the signal is deasserted. Requests from the buffered write at the same interface are masked off until the signal is deasserted. Using this scheme, the Q-bus latency in the new VAX 4000 systems was never more than 8 microseconds.

Enhanced CVAX Pin Bus

The NCA supports the standard bus protocol in both synchronous and asynchronous modes. The existing CP bus protocol does not utilize the maximum bus bandwidth possible with the standard CP bus protocol. The fastest data transfer rate is two cycles per four-byte (i.e., 32-bit) longword, because the two primary signals for the handshake use the same clock phase for the assertion. When the sent signal is detected, it is already too late to generate the received signal within the same cycle. To achieve the one-cycle transfer rate, a modified protocol is used. The received signal is changed to represent a ready-to-receive signal. The received signal is asserted regardless of whether or not the sent signal is asserted. When the sent and received signals are asserted at the same time, both the master and slave devices know that the data was successfully transferred.

This protocol works with the existing CP bus chips and has increased the theoretical bandwidth by up to 66 percent. For an 80-ns CP bus cycle time, the maximum bandwidth is 33.33MB/s.

Testability

To assist module testing, the NCA contains features that comply with the IEEE Standard P1149.1 JTAG testability.[6] At the pin level, five special pins are provided and work in combination with the internal test access port controller inside the NCA and a bed-of-nails tester to perform short- and open-circuit interconnection tests.

MS690 Memory Module

The MS690 family of CMOS memory modules was designed to support the memory requirements set forth by the NVAX memory controller.[2] The NMC requires the MS690 memory module to provide a two-way, bank-interleaved, 72-bit data path. In addition, a self-test feature is provided that was used on the VAX 4000 Model 300 memory subsystem. The MS690 module returns a unique board identification signature when polled by the NMC. The module used existing qualified parts and fits on a quad-sized PWB.

A common goal of Digital's Electronic Storage Development (ESD) teams is to utilize a single PWB design to accommodate as many memory sizes as possible. The ESD teams routinely stretch the boundaries of Digital's manufacturing processes to provide world-class memory subsystems. Because memory subsystems form the core of the ESD charter, the ESD teams are uniquely tuned into, and actively shaping, present and future device specifications for all types of random-access devices. This advance and intimate knowledge allows us to build current technology products with the hooks necessary to capitalize on the next generation of storage devices.

The MS690 options are available in 32MB, 64MB, and 128MB sizes and are self-configuring. The MS690 memories communicate with the NMC by way of the private NMI. All control and clocks signals originate off-board via the NMI from the NMC. Up to four memory modules of any density mix may coexist on the NMI with a maximum memory size of 512MB.

The MS690 is an extension of the existing 39-bit MS670 memory product designed for the VAX 4000 Model 300 product line. The DC562 GMX was designed and produced in Digital's Hudson, Massachusetts, plant for the MS670 32MB memory. This

GMX is a semi-intelligent, 20-bit-wide, 4-to-1 and 1-to-4 transceiver, with internal test/compare/error logging capabilities for its five I/O ports. The MS670 required eight banks of 39 bits of data, hence the requirement of two GMX chips per module.

The KA670 CPU module used in the VAX 4000 Model 300 transfers 32-bit longwords of data. For every longword, 7 bits of ECC must be allocated, i.e., 8 x (32 + 7) = 312 DRAMs. The CPU module used in the VAX 4000 Model 500 transfers 64-bit quadwords of data. For every quadword, 8 bits of ECC must be allocated, i.e., 4 x (64 + 8) = 288 DRAMs. The MS690 memory is configured as two interleaved bank pairs, each 72 bits wide (64 bits of data plus 8 bits of ECC); all transactions are 72 bits. The memory module supports quadword, octaword, and hexword read/write/read-modify-write transactions. Transactions less than 72 bits, i.e., bytes, words, and longwords, are not supported.

Doubling the data word length is advantageous in two ways: the I/O bandwidth effectively doubles, and 24 fewer DRAMs are required. This last benefit results from the fact that only one additional bit is required to protect 64 bits of data as compared to protecting 32-bit data. The available PWB space allowed room for two additional GMXs to handle the 33 additional data bits. The ability to use the existing GMX integrated circuit eliminated the need for a new, 40-bit-wide, GMX-type VLSI development.

Because DRAMs are edge-sensitive devices, module layout, balanced etch transmission lines, and signal conditioning are extremely important to a quality product. The MS690 design team used a combined total of 18 years of memory design experience along with extensive use of SPICE modeling to determine the optimal PWB layout. The result was a double-sided, surface-mount PWB panel that can accommodate all density variations of the MS690 memory option and thus help control costs by reducing product-unique inventory. All parts, except the bare PCB, are used on products already produced in volume at Digital's Singapore and Galway, Ireland, manufacturing plants.

The MS690-BA memory module, which uses 100-ns 1M-by-1M DRAMs, can support NMC cycle times of 36 ns and 42 ns, respectively, for the VAX 4000 Model 400 and 500 systems. The MS690-CA/DA modules use 80-ns 4M-by-1M DRAMs and can accommodate 30-ns, 36-ns, and 42-ns NMC cycle times.

Performance

The CPU I/O subsystems on all three products provide exceptional performance, as shown in Table 3. The pair of DSSI buses on the CPU modules for the VAX 4000 Models 500 and 600 were tested under the VMS operating system performing single-block (512-byte) reads from RF73 disk drives. The read rate was measured at over 2,600 I/Os per second with both buses running. The Ethernet subsystem, based on the SGEC adapter chip, is also very efficient. It has been measured transmitting and receiving 192-byte-long packets at a rate of 5,882 packets per second. Packets 1,581 bytes long can be transmitted at a rate of 9.9 megabits per second.

The performance of the CPU subsystem has traditionally been measured using a suite of 99 benchmarks.[7] Scaling the results against the performance of the VAX-11/780 processor and taking the geometric mean yields the VAX unit of performance (VUP) rating. The processor VUP rating for the new VAX 4000 system with the lowest performance, the Model 400, is twice the VUP rating of the system it is replacing, the Model 300. The two new high-end systems provide three and four times the performance of the Model 300—an impressive performance increase.

Table 3 Summary of Performance Results for the VAX 4000 Models 400, 500, and 600[7]

Metric	Unit	Model 400	Model 500	Model 600
SPEC Release 1.0	SPECmark	22.3	30.7	41.1
	SPECint	17.1	24.9	31.8
	SPECfp	26.6	35.4	48.7
Single User 99	VUPs	16.9	23.8	31.4
TPC-A	tpsA-local	51.0	62.4	103.0
Dhrystone Integer	MIPS	34.2	43.4	64.4
Whetstone Single	MIPS	47.6	71.4	83.3
Whetstone Double	MIPS	32.3	45.5	52.6
LINPACKD (100 by 100)	MFLOPS	4.8	6.9	9.5
LINPACKS	MFLOPS	7.5	10.5	14.7

The system performance in multistream and transaction-oriented environments was measured with TPC Benchmark A.[8] This benchmark, which simulates a banking system, generally indicates performance in environments that are characterized by concurrent CPU and I/O activity and that have more than one program active at any given time. The performance metric is transactions per second (TPS). The measured performance of the VAX 4000 Model 600 system was more than 100 TPS, tpsA-local. As shown in Table 3, the performance of the new VAX 4000 Model 400, 500, and 600 systems is impressive, even compared to RISC-based systems.

Acknowledgments

The authors would like to acknowledge the contributions of the following members of the design teams: Matt Baddeley, Roy Batchelder, Abed Chebbani, Harold Cullison, Todd Davis, Joe Ervin, Nannette Fitzgerald, Avanindra Godbole, Judy Gold, Bryce Griswold, Thomas Harvey, Jim Kearns, John Kumpf, Steve Lang, Dave Lauerhass, Bill Munroe, Chun Ning, Jim Pazik, Cheryl Preston, Matt Rearwin, Dave Rouille, Prasad Sabada, Stephen Shirron, Adam Siegel, Jaidev Singh, Emily Slaughter, Michael Smith, Steve Thierauf, Bob Weisenbach, and Pei Wong.

References

1. G. Uhler et al., "The NVAX and NVAX+ High-performance VAX Microprocessors," *Digital Technical Journal,* vol. 4, no. 3 (Summer 1992, this issue): 11–23.
2. *KA680 CPU Module Technical Manual* (Maynard, MA: Digital Equipment Corporation, Order No. EK-KA680-TM-001, 1992).
3. B. Maskas, "Development of the CVAX Q22-bus Interface Chip," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 129–138.
4. J. Winston, "The System Support Chip, a Multifunction Chip for CVAX Systems," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 121–128.
5. P. Rubinfeld et al., "The CVAX CPU, a CMOS VAX Microprocessor Chip," *ICCD Proceedings,* (October 1987): 148–152.
6. *IEEE Standard Test Access Port and Boundary Scan Architecture,* IEEE Standard 1149.1-1990 (New York, NY: The Institute of Electrical and Electronics Engineers, Inc., 1990).
7. *VAX Systems Performance Summary* (Maynard, MA: Digital Equipment Corporation, Order No. EE-C0376-46, 1991).
8. *Transaction Processing Performance Council, TPC Benchmark A Standard Specification* (Menlo Park, CA: Waterside Associates, November, 1989).

General Reference

DEC STD 032-0 VAX Architecture Standard (Maynard, MA: Digital Equipment Corporation, Order No. EL-00032-00, 1989).

Jonathan C. Crowell
David W. Maruska

The Design of the VAX 4000 Model 100 and MicroVAX 3100 Model 90 Desktop Systems

The MicroVAX 3100 Model 90 and VAX 4000 Model 100 systems were designed to meet the growing demand for low-cost, high-performance desktop servers and timesharing systems. Both systems are based on the NVAX CPU chip and a set of VLSI support chips, which provide outstanding CPU and I/O performance. Housed in like desktop enclosures, the two systems provide 24 times the CPU performance of the original VAX-11/780 computer. With over 2.5 gigabytes of disk storage and 128 megabytes of main memory, the complete base system fits in less than one cubic foot of space. The system design was highly leveraged from existing designs to help meet an aggressive schedule.

The demand for low-cost, high-performance desktop servers and timesharing systems is increasing rapidly. The MicroVAX 3100 Model 90 and VAX 4000 Model 100 systems were designed to meet this demand. Both systems are based on the NVAX CPU chip and a set of very large-scale integration (VLSI) support chips, which provide outstanding CPU and I/O performance.[1]

Each member of the MicroVAX 3100 family of systems constitutes a low-cost, general-purpose, multiuser VAX system in an enclosure that fits on the desktop. This enclosure supports all the required components of a typical system, including the main memory, synchronous and asynchronous communication lines, thick-wire and ThinWire Ethernet, and up to five small computer system interface (SCSI)-based storage devices.

The MicroVAX 3100 Model 90 system replaces the Model 80 as the top performer in the line; the new model has considerably more than twice the CPU power of the previous model.[2] The Model 90 system also includes performance enhancements to the Ethernet and SCSI adapters, as well as an increased maximum system memory of 128 megabytes (MB). The CPU mother board for the MicroVAX 3100 Model 90 system is called the KA50.

The VAX 4000 Model 100 system is housed in the same desktop packaging as the MicroVAX 3100 Model 90 and provides the same base functionality. The VAX 4000 Model 100 adds two key features found in all previous VAX 4000 systems, i.e., Digital Storage Systems Interconnect (DSSI) storage and Q-bus expansion. The CPU mother board for the VAX 4000 Model 100 system is called the KA52.

The KA50 and KA52 CPUs are built from a common CPU mother board design; the base CPU mother board is configured to create either the KA50 or the KA52 product module. The DSSI and Q-bus optional hardware is added to the CPU mother board to convert a KA50 to a KA52. Also, to provide the additional superset functionality found on the KA52 CPU, the different system read-only memories (ROMs) are added during the manufacturing process. In this paper, the KA50 and KA52 CPUs are referred to as the CPU mother board or module, except where differences exist.

The system design was highly leveraged from existing designs to help meet an aggressive schedule. This paper describes the design and implementation of these systems.

Design Goals

The design team's primary goal was to develop a CPU mother board that would provide at least twice the CPU performance of the MicroVAX 3100 Model 80, while supporting all of the same I/O functionality of the previous systems. This new system would leverage the core CPU design from the VAX 4000 Model 500 system, thus delivering the high performance of the NVAX CPU chip to the desktop.[3]

The team set additional goals to increase system capability and performance. These goals were to

1. Increase the maximum system memory from 72MB to 128MB
2. Provide error correction code (ECC) protection to memory using memory arrays that previously supported only parity
3. Increase the performance of the Ethernet adapter
4. Increase the performance of the SCSI adapter

Early in the project, the team proposed creating a second CPU design that would have the features of the larger VAX 4000 systems. This proposal resulted in the design of a DSSI adapter option for the CPU mother board, as well as a Q-bus adapter to provide a means to upgrade the CPU power of older Q-bus–based MicroVAX systems.

The project to design, implement, and field-test these systems was accomplished under an aggressive schedule. Both designs were ready to ship to customers in just over nine months from the official start of the project.

System Overview

The MicroVAX 3100 Model 90 system supports the same I/O functionality as the previous generation of systems, the MicroVAX 3100 Models 40 and 80. The features include a SCSI storage adapter, 20 asynchronous communication ports, two synchronous communication ports, and an Ethernet adapter.

The VAX 4000 Model 100 includes the same I/O functionality as the MicroVAX 3100 Model 90. In addition, the system provides the I/O functionality of the larger VAX 4000 systems, that is, a high-performance DSSI storage adapter and a Q-bus adapter port that connects to an external Q-bus enclosure.

Both systems provide 24 times the CPU performance of a VAX-11/780 system. The memory subsystem uses Digital's MS44 single in-line memory modules (SIMMs) and thus provides 16MB, 32MB, 64MB, 80MB, or 128MB of main memory.

As shown in Figure 1, the system enclosure used to house both systems, namely the BA42B, provides mounting for the CPU mother board, up to five locations for disk and tape devices, a 166-watt power supply, and fans for cooling the system elements. In addition, the enclosure shields the system from radiated emissions. All I/O connections are filtered and exit the enclosure through cutouts in the rear panel. The system enclosure is compact and measures 14.99 centimeters (5.9 inches) high by 46.38 centimeters (18.26 inches) wide by 40.00 centimeters (15.75 inches) deep.

The system enclosure contains two shelves that support the mass storage devices. In the MicroVAX 3100 Model 90, these storage locations are cabled to support SCSI disks and tapes. The upper shelf supports three SCSI disks, whereas the lower shelf supports two SCSI devices (any combination of removable or 3 1/2-inch disks) with access through a door in the front of the enclosure. In the VAX 4000 Model 100, the top shelf is configured to support three 3 1/2-inch DSSI disks; the bottom shelf supports two SCSI devices, as in the MicroVAX 3100 Model 90.

The VAX 4000 Model 100 DSSI support is provided by a high-performance DSSI adapter card based on the shared-host adapter chip (SHAC), i.e., a custom VLSI design with an integrated reduced instruction set computer (RISC) processor.[3] The system is configured with DSSI as the primary disk storage. The DSSI bus exits the enclosure by means of a connector on the back panel. This expansion port can be used to connect the system to additional DSSI devices, or to form a DSSI-based VAXcluster with a second VAX 4000 Model 100 or any other DSSI-based system.

The Q-bus support on the VAX 4000 Model 100 is provided by the VLSI adapter chip, i.e., the CVAX Q22-bus interface chip (CQBIC).[4] There are no Q-bus option slots in the system enclosure. The Q-bus connects to an expansion enclosure through a pair of connectors at the rear of the system enclosure. Two shielded cables and the H9405 expansion module are used to connect the Q-bus to the expansion enclosure. The near end of the Q-bus is terminated in the system enclosure.

CPU Mother Board Design

The design goals presented engineering with constraints that forced design trade-offs. Some key constraints were (1) fitting the required functionality on a single 10-by-14-inch module; (2) designing the system to adhere to the system power and cooling budget; and (3) minimizing changes to the functional view of the module over previous designs, to decrease the number of software modifications required for operating system support.

The primary way the design team minimized system development was to leverage as much as practical from existing designs. The CPU mother board design used components from the VAX 4000

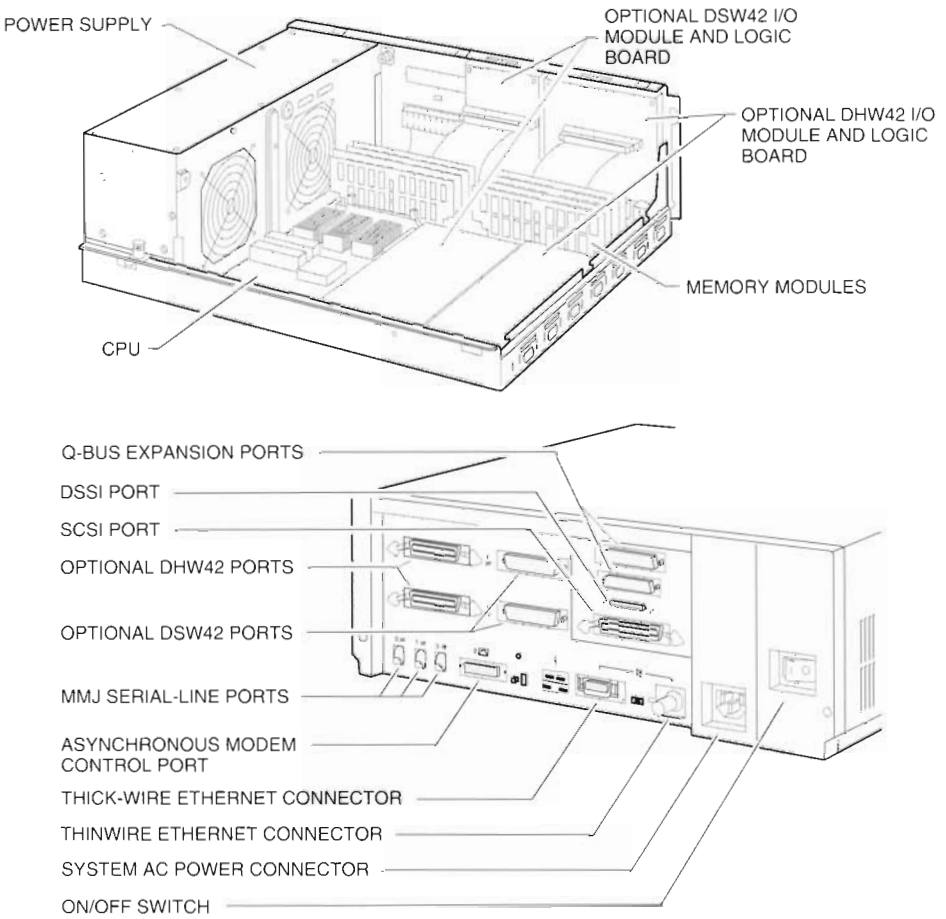


Figure 1 VAX 4000 Model 100 System Enclosure Showing the CPU and Connectors

Model 500, MicroVAX 3100 Model 80, and VAXstation 4000 Model 90 systems. Using proven design components allowed for a shorter development cycle, smaller design teams, and consequently, a higher-quality design, while meeting an aggressive schedule.

The design is structured so that both CPU mother boards can be built using the same printed wiring board (PWB). The added functionality for the KA52 is provided by a daughter card, additional hardware and cabling, and different system ROMs. The shared design helped reduce the complexity in testing and qualifying the system design.

The CPU module contains three major sections: the CPU core, the memory subsystem, and the I/O subsystem. Figure 2 is a block diagram of the basic CPU module for the VAX 4000 Model 100 and MicroVAX 3100 Model 90 systems. Figure 3 is a photograph of the module, including the DSSI daughter card option.

The CPU mother board includes a linear regulator that generates local 3.3-volt (V) current for the CPU core chip set. The voltage is stepped down from the 5-V supply. The regulator is necessary because the 3.3-V direct current (DC) of the system is not sufficient to meet the ± 3 percent tolerance regulation or to supply the required maximum current.

CPU Core

The CPU core consists of three chips: the NVAX CPU chip, the NVAX data and address lines (NDAL) memory controller (NMC) chip, and the NDAL-to-CVAX pin (CP) bus adapter (NCA) chip. The NVAX chip directly controls the 128-kilobyte (KB) backup cache. The core chip set is interconnected by means of the NDAL pin bus, as shown in Figure 2. The NDAL bus is 64 bits wide, has a 42-nanosecond (ns) cycle time, and supports pended transactions.[1] The peak bandwidth of the

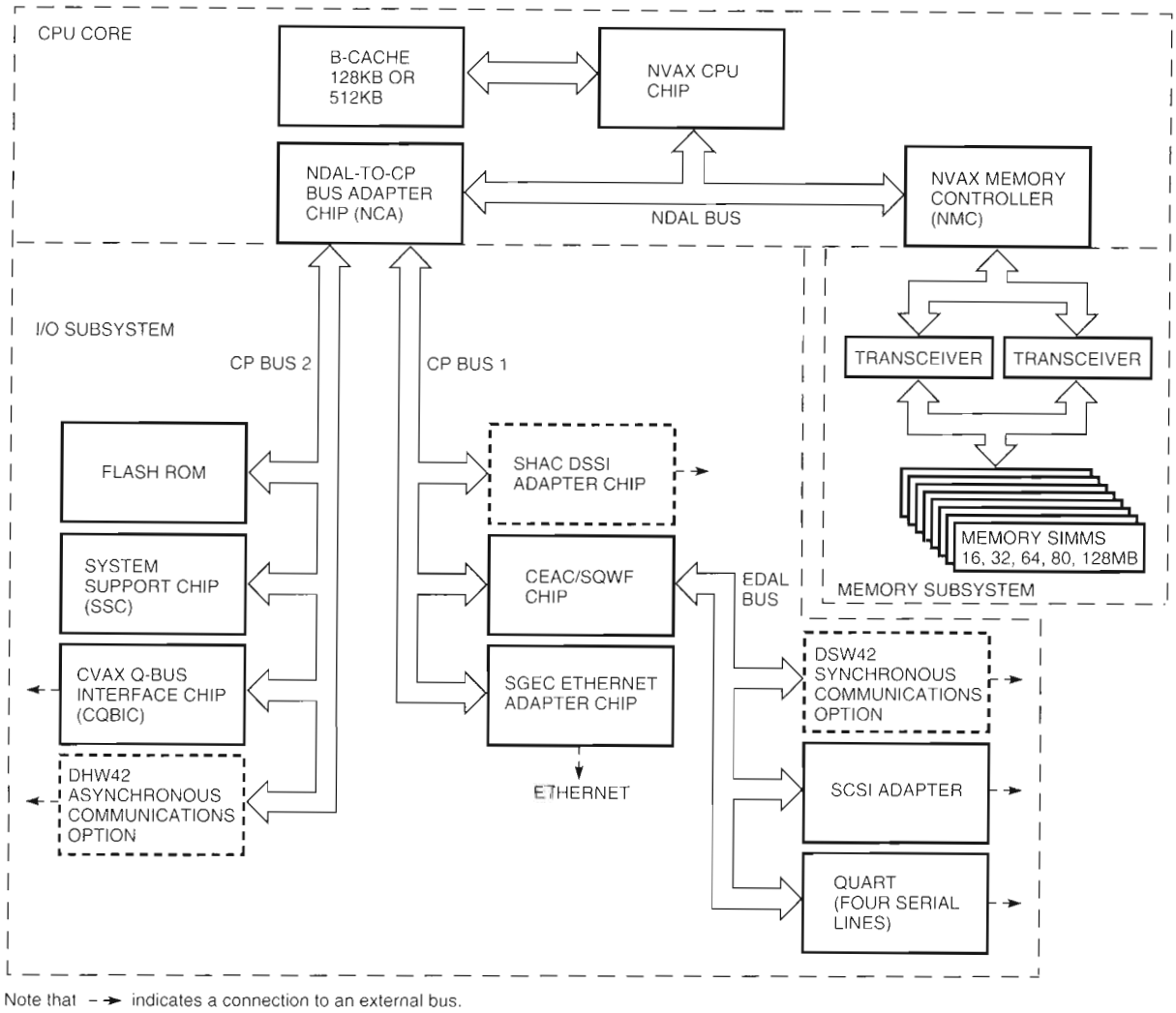


Figure 2 CPU Module Block Diagram

NDAL bus performing 32-byte operations is 152 megabytes per second (MB/s).

NVAX CPU Chip The NVAX CPU chip is an advanced implementation of the VAX architecture in Digital's fourth-generation complementary metal-oxide semiconductor (CMOS-4) technology. The NVAX device consists of 1.3 million transistors on a die approximately 0.6 inch on a side.

The NVAX CPU chip contains the VAX CPU, a floating-point unit, and backup cache controller logic. Some NVAX features that enable it to increase performance are the use of a pipelined architecture, a 2KB virtual instruction cache (VIC), a 96-entry translation buffer, an on-chip 8KB primary cache, and an on-chip backup cache controller. The CPU cycle clock and NDAL bus clocks are generated with an on-chip clock generator supplied by a 286-megahertz (MHz) oscillator.

The NVAX CPU is based on a high-performance macropipelined architecture similar to that of the VAX 9000 CPU.[1,5] The VIC allows the caching of instructions that have already been translated to virtual addresses. Having the backup cache controller on the chip decreases backup cache access time because no external logic, with the resulting delays, is required.

NVAX Memory Controller Chip The NMC is the NVAX memory controller implemented in Digital's third-generation complementary metal-oxide semiconductor (CMOS-3) technology.[6] The NMC consists

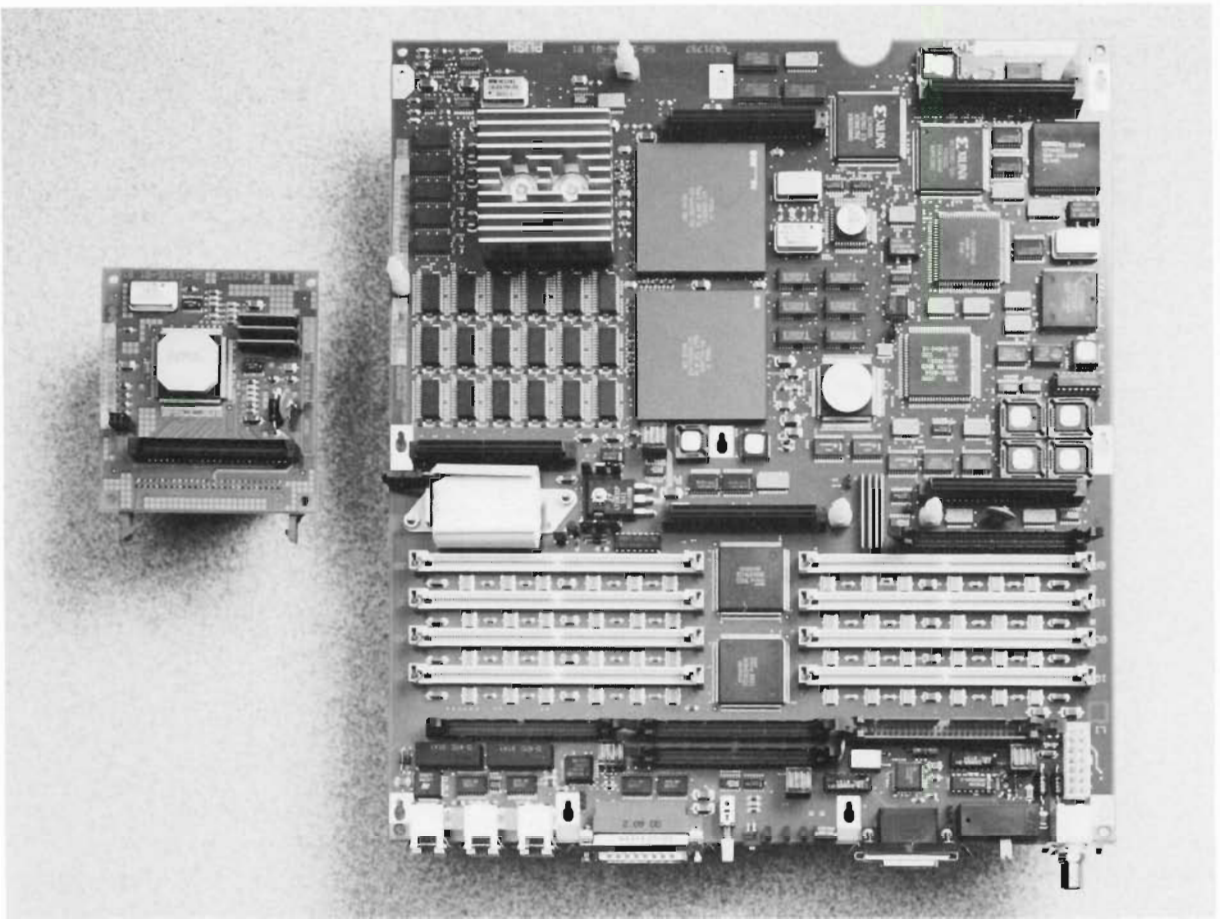


Figure 3 CPU Mother Board

of 148,000 transistors and is the high-speed interface to the system main memory. The NMC is the arbiter for the three chips on the NDAL bus, namely, the NVAX, the NCA, and the NMC. The NMC chip manages the array of ownership bits that correspond to each 32-byte segment of memory. Each of these segments corresponds to a cache line. The ownership bit indicates whether the valid copy of the data is in memory, in the CPU write-back cache, or in an I/O devices buffer.

The NMC has four command queues that accept read, write, and remove write privilege transactions from the NDAL bus. Buffers hold the read data to be returned to the node that requested the data. The NMC and the memory subsystem provide the 95MB/s of bandwidth shared by the NVAX and the I/O devices.

NDAL-to-CP Bus Adapter Chip The NCA chip, also implemented in Digital's CMOS-3 technology, is the interface from the NDAL to the CP bus.[6] The NCA consists of 155,000 transistors and supports two CP buses. The CP bus used on the CVAX microprocessor family is also used on many of Digital's custom I/O adapter chips, such as the CQBIC, the SHAC, the second-generation Ethernet controller (SGEC), and the system support chip (SSC).[3,4,7,8] Thus, the hardware and software designs for these I/O functions could be leveraged from previous efforts. The NCA performs direct memory access (DMA) from the I/O devices and supports the cache consistency protocol required for the NDAL bus.

The NCA was designed to optimize DMA traffic from CP bus devices. In the KA50 CPU, the CP bus devices include the SGEC Ethernet adapter, the SSC, the field erasable programmable read-only memory (FEPROM) subsystem, the CP-to-EDAL adapter chip (CEAC), and the SCSI quadword first in, first out (SQWF) chip. In addition, the asynchronous communication option is attached to the CP bus. The

KA52 CPU also attaches the CQBIC Q-bus adapter chip and the SHAC DSSI host adapter chip.

Memory Subsystem

The memory subsystem is controlled by the NMC chip. The main memory is implemented using MS44 SIMMs and low-cost gate array (LCGA) chips to provide an interface between the NMC and the SIMMs.[9] The SIMMs are used in groups of four to provide two interleaved banks, each with a 64-bit data path and eight bits of ECC. This interleaving scheme increases the bandwidth of main memory by alternating data between both banks of memory. The ECC provides single-bit error correction and double-bit error detection.

The individual SIMMs are available in either 4MB or 16MB variants. Since four SIMMs form a complete functional set, sets can be 16MB or 64MB in size. Therefore, because the system supports up to two sets of SIMMs, the total system memory size can be either 16MB, 32MB, 64MB, 80MB, or 128MB, depending on the combination of SIMM size and the number of sets.

To coincide with the cache coherency scheme used in the NVAX CPU chip, the NMC keeps track of the cache lines that have write privilege reserved by the CPU or I/O devices. This state is stored in separate dynamic random-access memories (DRAMs). These DRAMs interface directly to the NMC by means of a private bus. The ownership bits are protected by ECC.

I/O Subsystem

Because the MicroVAX 3100 Model 90 was intended as an upgrade for the Model 40 and 80 systems, the I/O subsystem of the earlier systems dictated the design of the new Model 90. In addition, the I/O subsystem of the KA52 CPU module for the VAX 4000 Model 100 supports two functions found in the other VAX 4000 systems, the DSSI adapter and the Q-bus adapter.[9] The I/O subsystem includes a ThinWire and thick-wire Ethernet adapter, four built-in asynchronous terminal lines, a connector for the asynchronous option, and the CEAC and SQWF chips.

A bus interface was incorporated in the I/O subsystem to support the DSW42 synchronous communication option, the SCSI adapter chip, and the QUART four-port asynchronous controller chip. The CEAC and SQWF chips, which are gate arrays designed for the VAXstation 4000 Model 90, are used to create the EDAL bus.

Support for the SCSI bus is provided by the 53C94 SCSI adapter chip.[10] The 53C94 chip is interfaced to the system on the EDAL bus and uses the SQWF chip to increase its DMA performance. The SQWF chip makes it possible to buffer data moving to the CP bus. The SCSI bus operates in synchronous mode for high-performance storage access of 5MB/s.

The QUART gate array supplies the logic for four built-in serial ports. The QUART, originally used on the DZQ11 Q-bus device, provides the same software interface as that device. The third port provides modem control functions by means of additional logic; the first, second, and fourth ports are data leads only.

The SGEC Ethernet adapter chip was chosen because it provides higher performance than the Ethernet adapter used on the MicroVAX 3100 Model 80. The SGEC is the adapter chip used on all VAX 4000 systems. In addition, this chip directly interfaces with the CP bus.

The limited size of the CPU mother board required the DSSI adapter to be added by means of a daughter card. The Q-bus adapter chip and bus termination are provided directly on the mother board.

Console and Diagnostics

The MicroVAX 3100 Models 80 and 90 differ in their console designs and diagnostics. Because the basic CPU core of the MicroVAX 3100 Model 90 and the VAX 4000 Model 100 systems is very similar to that of the VAX 4000 Model 500 system, the design team decided to adopt the console of the Model 500 and add the required commands and functionality. Borrowing proven designs, such as the console of another NVAX-based system, significantly shortened the product development schedule.

One enhancement to the CPU mother board was the addition of a FEPROM subsystem. If an update is required, the console and diagnostic code on the CPU can be reprogrammed in the field. In contrast, previous systems required the memories to be in sockets and the parts to be replaced in the field. With FEPROMs, a program is loaded from any bootable device. This program erases the FEPROMs and reprograms them with the new ROM image. This enhancement serves as an easy mechanism for updating the ROMs in the field to provide new features or to fix bugs that may be discovered.

On power-up, the CPU starts executing from the FEPROM memory and runs the power-up self-test to help verify that the system is fully operational.

Upon completion of the execution of this test, the system transfers control to the console program. Depending on the values configured in nonvolatile memory, the console program either boots the system with the correct parameters or stops for console input.

In earlier systems, the speed of executing from ROM could be more than an order of magnitude slower than running from cached main memory. The NVAX CPU chip added the virtual instruction cache, which allows the caching of instruction stream references from I/O space. This feature greatly increases the performance of the ROM code.

Console

The console program gains control of the CPU whenever the processor halts or performs a restart operation such as power-up. The console provides the following services:

1. Interface to the diagnostics that test components of the CPU and system
2. Automatic/manual bootstrap of an operating system following processor halts
3. An interactive command language that allows the user to examine and alter the state of the processor

There are minor differences between the KA50 and KA52 consoles. Largely, these differences relate to the KA52 CPU mother board support for the DSSI bus and the Q-bus. Although the console is similar to that found on the VAX 4000 Model 500, some new commands were implemented to provide functionality that exists on previous MicroVAX 3100 systems. These commands include LOGIN and SET PSWD (set password), which give support for a secure console; SET/SHOW SCSI_ID; SHOW CONFIGURATION; SHOW ERROR; and various commands to support a system exerciser.

On the KA52 CPU, the console supports the DSSI bus and the Q-bus with a set of commands. These commands allow polling of the DSSI bus to determine what devices are present and to configure the internal parameters of each device. The system can be booted from devices on the SCSI, DSSI, or Q-bus chips, as well as over the Ethernet port.

Diagnostics

Diagnostics help isolate faults in the system down to the level of the field-replaceable units. Significant effort was expended on the development of onboard diagnostics. However, as for the console, the philosophy used in developing these diagnostics was to leverage as much of the software design as possible from existing designs.

With the advent of larger boot and diagnostic ROMs, the diagnostic coverage of the power-up self-tests greatly increased, including extensive testing of the cache, the memory, and the CPU core. These tests help assure the customer that a failed component will be detected and reported upon power-up. In many cases, the new tests can help isolate failures in the individual SIMM or cache chip. This feature is used extensively in manufacturing, as well as by field service.

During the power-up sequence, an instruction exerciser (HCORE) is run to test the floating-point hardware. This test provides very good coverage of the floating-point unit. In the past, HCORE has been run as a standalone diagnostic in manufacturing before a system is shipped. The design team for the two new desktop systems believed that this test should run on every system power-up self-test.

The CPU core is designed to function over a wide range of environmental conditions. Some variables of the environment are temperature, voltage, and minimum/maximum component parameters at a given clock frequency. Exceeding the worst-case design envelope can cause unpredictable results. For example, to avoid problems caused by a defective main clock oscillator that may be running too fast, the diagnostics measure the speed of the CPU cycle clock to determine if it is within the accepted tolerance. If the cycle is faster than the design margin dictates, an error is reported.

Design Tools

The design of the CPU mother board uniquely merged components from several designs. The success of this approach relied on the use of design tools to perform the merge and to verify the correctness of the merge.

The normal design process is to create a set of design schematics and to verify these through simulation. Once the design is logically verified, the layout process begins. The layout process includes the use of the SPICE simulator to give direction to the physical layout structure and to check the integrity of the layout.[11] After the layout is complete, the database is fed back into logic simulation, which again verifies the correctness of the design database.

The CPU mother board designers took a different approach. Since the physical placement of the connector portion of the module was the same as for the MicroVAX 3100 Model 80 module, this design element was used as the starting point for the overall design. The database was edited using the VAX layout system (VLS), and the only components saved were those that were to be used in the new CPU module. This procedure provided the correct placement for all I/O connectors that exited the system enclosure.

In addition, the VAX 4000 Model 500 CPU core was used as the basis for the CPU mother board etch layout. The Model 500 design has proven to have very good signal integrity due to its well-thought-out circuit board layout. To leverage Model 500 work in the layout of the CPU mother board, the designers extracted the printed circuit board signal routing from the Model 500. This signal routing included the CPU core and cache treeing, the most critical areas. This approach eliminated the need to model critical signal interconnect in the design and guaranteed that the signal integrity and connector layout would be identical to that of the proven Model 500.

Database comparison tools were used to guarantee that the schematics matched the physical layout database. As a final step, the physical layout database netlist was used to create a simulation model. DECSIM, Digital's simulation tool, was used to verify the final correctness of the design database.

Performance

The CPU I/O subsystems on both the MicroVAX 3100 Model 90 and the VAX 4000 Model 100 provide exceptional performance. The DSSI bus on the KA52 CPU was tested under the VMS operating system performing single-block (512-byte) read operations from RF35 disk drives. The read rate was measured at more than 1,200 I/Os per second. The SCSI adapter on both CPUs was measured at more than 500 I/Os per second for single-block reads.

The Ethernet subsystem used on both the KA50 and KA52 modules is very efficient and has been measured transmitting 64-byte packets at a rate of 14,789 packets per second. The measured receive rate for 64-byte packets was 14,785 packets per second.

The performance of the CPU subsystem has traditionally been measured using a suite of 99 benchmarks.[12] The results are scaled against the performance of the VAX-11/780 processor, and the geometric mean is taken. This calculation yields the VAX unit of performance (VUP) rating. The processor VUP rating for both the KA50 and KA52 CPUs is 24 VUPs—more than twice the performance of the MicroVAX 3100 Model 80. Table 1 presents a summary of the performance results for the VAX 4000 Model 100 and the MicroVAX 3100 Model 90 systems.

The performance of the system in multistream and transaction-oriented environments was measured with TPC Benchmark A.[13] This benchmark, which simulates a banking system, generally indicates performance in environments characterized by concurrent CPU and I/O activity and in which more than one program is active at any given time. The performance metric is transactions per second (TPS). The measured performance of the VAX 4000 Model 100 is 50 TPS tpsA-local; that of the MicroVAX 3100 Model 90 is 34 TPS tpsA-local. The difference in performance between the VAX 4000 Model 100 and the MicroVAX 3100 Model 90 is a result of their different disk subsystems, i.e., the DSSI and SCSI adapter support.

Table 1 Summary of Performance Results for the VAX 4000 Model 100 and the MicroVAX 3100 Model 90 Systems[13]

Metric	Unit	VAX 4000 Model 100	MicroVAX 3100 Model 90
SPEC Release 1.0	SPECmark	30.5	30.5
	SPECint	24.3	24.3
	SPECfp	35.5	35.5
Single User 99	VUPs	23.8	23.8
Integer	VUPs	19.6	19.6
Single	VUPs	26.0	26.0
Double	VUPs	31.3	31.3
TPC-A	tpsA-local	51	34

Acknowledgments

The authors would like to thank all the designers of other products whose logic components were used in the VAX 4000 Model 100 and MicroVAX 3100 Model 90 designs, namely, the VAXstation 4000 Model 90, VAX 4000 Model 500, and MicroVAX 3100 Model 80 teams. The authors would also like to acknowledge the many engineers who helped design and qualify the VAX 4000 Model 100 and MicroVAX 3100 Model 90 systems on a very aggressive schedule. Special thanks to Harold Cullison and Judy Gold (diagnostics and console), Matt Benson and Joe Ervin (module design and debug), Cheryl Preston (signal integrity), Dave Rouille (design verification), Colin Brench (EMC consultant), Larry Mazzone (mechanical design), Billy Cassidy and Walt Supinski (PWB layout), and Rita Bureau (schematics).

References and Note

1. G. Uhler et al., "The NVAX and NVAX+ High-performance VAX Microprocessors," *Digital Technical Journal,* vol. 4, no. 3 (Summer 1992, this issue): 11–23.

2. *MicroVAX 3100 Models 40 and 80 Technical Information Manual,* rev. 1 (Maynard, MA: Digital Equipment Corporation, Order No. EK-A0525-TD, 1991).

3. *KA680 CPU Module Technical Manual* (Maynard, MA: Digital Equipment Corporation, Order No. EK-KA680-TM-001, 1992).

4. B. Maskas, "Development of the CVAX Q22-bus Interface Chip," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 129–138.

5. J. Murray, R. Hetherington, and R. Salett, "VAX Instructions That Illustrate the Architectural Features of the VAX 9000 CPU," *Digital Technical Journal,* vol. 2, no. 4 (Fall 1990): 25–42.

6. J. Crowell et al., "Design of the VAX 4000 Model 400, 500, and 600 Systems," *Digital Technical Journal,* vol. 4, no. 3 (Summer 1992, this issue): 60–72.

7. P. Rubinfeld et al., "The CVAX CPU, a CMOS VAX Microprocessor Chip," *ICCD Proceedings* (October 1987): 148–152.

8. G. Lidington, "Overview of the MicroVAX 3500/3600 Processor Module," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 79–86.

9. M. Callander, Sr., L. Carlson, A. Ladd, and M. Norcross, "The VAXstation 4000 Model 90," *Digital Technical Journal,* vol. 4, no. 3 (Summer 1992, this issue): 82–91.

10. NCR Microelectronics Products Division, *NCR 53C94, 53C95, 53C96 Advanced Controller Data Sheet* (Santa Clara, CA: NCR Corporation, 1990).

11. SPICE is a general-purpose circuit simulator program developed by Lawrence Nagel and Ellis Cohen of the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley.

12. *VAX Systems Performance Summary* (Maynard, MA: Digital Equipment Corporation, Order No. EE-C0376-46, 1991).

13. *SPEC: A New Perspective on Comparing System Performance,* rev. 10 (Maynard, MA: Digital Equipment Corporation, Order No. EE-EA379-46, 1990).

14. *Transaction Processing Performance Council, TPC Benchmark A Standard Specification* (Menlo Park, CA: Waterside Associates, November 1989).

Michael A. Callander, Sr.
Lauren M. Carlson
Andrew R. Ladd
Mitchell O. Norcross

The VAXstation 4000 Model 90

The VAXstation 4000 Model 90 is the latest member of the VAXstation product line. Based on the NVAX CPU, the Model 90 was designed as a module upgrade to the VAXstation 4000 Model 60 system. The Model 90 has 2.7 times the CPU performance of the Model 60 and provides base-level, two-dimensional graphics performance of 266,000 vectors per second. It supports up to 128MB of memory, an SCSI-1 bus interface, a TURBOchannel option, a synchronous communication option, and several graphics options. The design team used only programmable devices to implement the new logic designed into the system. In addition, a breadboard provided the basis for logic and software verification.

During the summer of 1991, Digital's Semiconductor Engineering Group began planning a new VAX workstation based on the NVAX CPU chip.[1] The development process had three main goals: to increase CPU performance, to maintain an aggressive time-to-market schedule, and to provide upgrade compatibility with the VAXstation Model 60.

The primary goal of the VAXstation 4000 Model 90 design was to implement a workstation with well over twice the CPU performance of its predecessor, the Model 60. The advent of high-performance workstations based on reduced instruction set computers (RISC) required any new VAX workstation to provide a significant performance increase over previous VAX workstations to be competitive in the marketplace. The Model 90 met this goal by achieving 2.7 times the performance of the VAXstation Model 60.

The second major goal of the project was to develop and ship the system as quickly as possible. This was mandated by competitive pressures in the workstation market. We proposed an aggressive best-case schedule which forecast a breadboard running within three months of the project proposal, prototypes running the VMS system within five months, and a customer ship date within eleven months. The development teams achieved almost every project milestone within a few weeks of the proposed schedule.

The final major goal of the project was to design the system such that it could be offered as a simple module upgrade to the VAXstation Model 60. There were two main reasons for designing the system as an upgrade. First, it protected the customer's investment in the Model 60 components. Second, by using as many components as possible from the Model 60 design, we could reduce the hardware and software engineering effort required to produce the new system. The Model 90 system module provides a direct upgrade from the Model 60. The only system component or option on the Model 60 that is not supported by the Model 90 is the entry-level graphics option.

This paper presents the design methodology we followed to meet our project goals. It discusses the four major components in the Model 90 system. It describes the physical design of the system board and the breadboard system we used for logic verification and debugging of software. The paper ends with a comparison of performance data for Digital's workstations.

Design Methodology

The design methodology used during the Model 90 project consisted of the following approaches:

- Complex logic, software, and firmware from existing designs would be used whenever possible.
- All new logic would be implemented using programmable technology.
- A breadboard would be built as early as possible.
- Logic would be simulated only if it could not be verified with the breadboard.

These approaches were influenced and shaped by our aggressive schedule, by the emergence of new programmable technologies, and by the

availability of certain VAX system designs that included some of the subsystems that we planned to use. These influences are discussed in this section.

The strategy of using existing hardware and software components stemmed from our goal to deliver the Model 90 as quickly as possible. The project schedule did not allow time for the development of any major new pieces of hardware or software. Consequently, we used as much hardware and software from other VAX products as possible.

Our aggressive schedule also prompted us to explore different technologies and verification methods. On our previous projects, we used conventional gate array or standard cell technology, and typically we strove for exhaustive logic simulation and timing verification prior to releasing chip designs. Our goal was to have fully functional first-pass silicon. Unfortunately, the first pass of a gate array was rarely fully functional. This approach had two consequences on the project schedule: (1) First-pass hardware was usually delayed as much as possible to allow for more thorough logic simulation and timing verification, and (2) a second pass was needed if first-pass silicon was not fully functional, adding several months to the overall project schedule.

At the time of our design, several new programmable silicon technologies were emerging that promised performance, densities, and package sizes comparable to gate array technology. As the logical design of the system progressed through its early stages, we evaluated these new technologies and found that they had matured enough to be used in the design of the Model 90. We chose Xilinx field programmable gate arrays (FPGAs) and AMD MACH PALs to implement large-scale integration, and standard PALs to implement smaller logic functions. These programmable technologies allowed us to build first-pass hardware with the full expectation that we would need to make inevitable changes in response to logic bugs and timing problems. Fortunately, with the new technologies, bug fixes were made in a matter of minutes or days, instead of the weeks or months it would have taken using conventional gate arrays.

During the Model 90 project, we examined our previous notions about the roles of prototyping and simulation in product development. Because the core of the Model 90 was borrowed from the VAX 4000 Model 500, an opportunity arose for us to build a breadboard system consisting of the programmable I/O and graphics interface designs attached to a VAX 4000 Model 500.[2] Unlike a conventional prototype, the breadboard logic was expected to change; therefore we included reconfigurable connections to the FPGAs. Also, the breadboard system did not need to meet any of the physical constraints, such as size and layout, that are normally required of a conventional prototype. An early breadboard system provided the clear benefits of rapid testing and change of hardware and software.

Because we could change logic quickly and easily on the breadboard, the role of simulation on this project focused on verifying module interconnect, and not on exhaustive logic verification. We maintained a working system-simulation model, with a basic set of regression tests, as a reference for logic changes and as a tool for debugging. Logic verification was performed on the breadboard to an extent not possible using simulation.

Major System Components

Figure 1 is a block diagram showing the primary components in the Model 90. In this paper we focus on four distinct components in the system: the core, the memory subsystem, the I/O subsystem, and the graphics subsystem.

The core chip set is composed of a 74.4-megahertz (MHz) NVAX CPU, the NVAX memory controller (NMC), and the NVAX I/O adapter (NCA). The NVAX CPU also controls a 256-kilobyte (KB) write-back secondary cache that reduces memory read latency and decreases memory write traffic.

The memory subsystem supports a 64-bit data path to main memory that is composed totally of single in-line memory modules (SIMMs). Main memory sizes of up to 128 megabytes (MB) are supported by the Model 90.

The I/O subsystem comprises two independent 32-bit buses that communicate with the various I/O and graphics options of the Model 90. One bus interfaces to the optional TURBOchannel adapter, the firmware read-only memory (ROM) chips used for console and diagnostics, and the various graphics options available with the Model 90.[3] The other bus interfaces to the Ethernet and EDAL controllers. The EDAL is a general-purpose 16-bit I/O bus. The EDAL controller consists of a CDAL-to-EDAL chip (CEAC) and a small computer system interface (SCSI) quadword first-in first-out (FIFO) chip, known as SQWF. These two chips communicate over the EDAL bus with the system's remaining I/O devices.

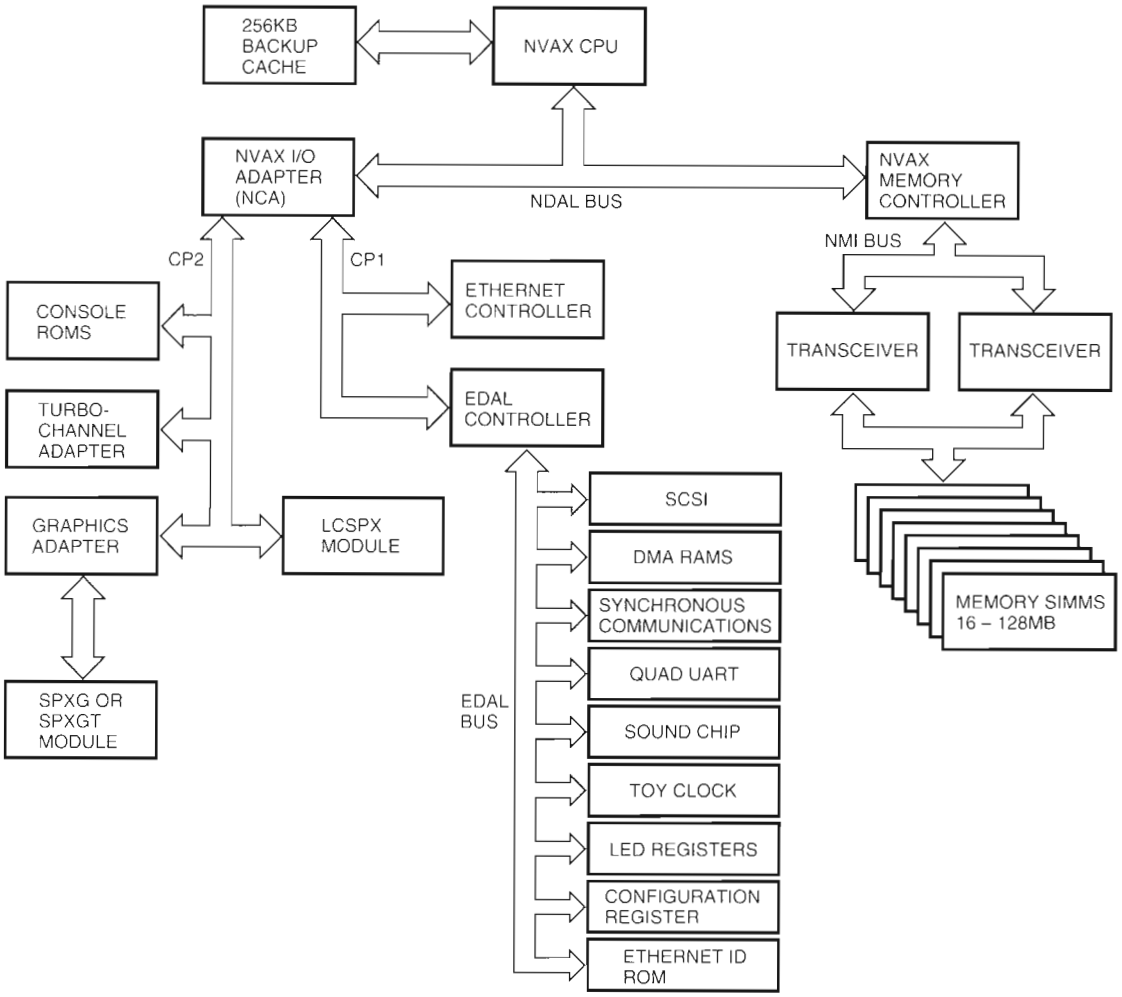


Figure 1 Block Diagram of the VAXstation 4000 Model 90

Finally, the graphics subsystem provides support for three different graphics options. These options include one low-cost graphics option and two high-performance three-dimensional accelerators.

The majority of the components used in the Model 90 had been used in previous VAX systems. Table 1 lists the major Model 90 components and indicates the source of these components.

VAXstation 4000 Model 90 Core

The NVAX CPU, the NMC, the NCA, and the backup cache compose the core of the system module. This core architecture was taken directly from the VAX 4000 Model 500 system. This architecture was chosen because it would meet our performance goals; it provided simple interfaces to our memory, I/O, and graphics subsystems; and because the design was completed and stable. The NVAX CPU is a fully custom complementary metal-oxide semiconductor (CMOS) CPU fabricated in Digital's 0.75-micrometer CMOS-4 process. The NCA and NMC are also fully custom CMOS chips, but are fabricated using Digital's 1.0-micrometer CMOS-3 process. The three custom chips communicate with each other over a 64-bit bidirectional bus named the NDAL.

The NVAX CPU contains a 2KB virtual instruction cache, an 8KB write-through instruction/data primary cache, and, on the Model 90, interfaces to a 256KB write-back instruction/data secondary cache. It contains an on-chip floating-point unit and branch prediction logic. The NVAX CPU pipelines instruction execution at the macroinstruction level as well as the traditional microinstruction level.

Table 1 Model 90 Component Source

Component	Source
Core chip set	VAX 4000 Model 500
Ethernet controller	VAX 4000 Model 500
Memory SIMMs	VAXstation 4000 Model 60
SPXg and SPXgt graphics options	VAXstation 4000 Model 60
TURBOchannel option	VAXstation 4000 Model 60
Synchronous communications option	VAXstation 4000 Model 60
SCSI controller	VAXstation 4000 Model 60
Enclosure, power supply, cables, brackets	VAXstation 4000 Model 60
Memory transceivers	VAXstation 4000 VLC
LCSPX graphics option	Modified VXT 2000 module
EDAL controller	New design
SPXg and SPXgt interface	New design

The NCA provides direct memory access (DMA) and programmed I/O (PIO) support between the 64-bit NDAL bus and two 32-bit bidirectional CDAL buses named CP1 and CP2. In the Model 90 system, these buses run at an 80-ns cycle time and interface to all the graphics and I/O devices in the system. The NCA also contains the VAX standard interval timer register as well as many of the I/O control and status registers.

The NMC services NDAL memory requests using a 64-bit dynamic random-access memory (DRAM) bus called the NMI, which is protected with an error correction code. The NMC, as configured in the Model 90, supports up to 128MB of main memory. It also supports a directory-based broadcast coherence protocol to maintain coherency between the write-back cache of the NVAX CPU and the system's DMA devices.

Memory Subsystem

The memory subsystem of the Model 90 is based on the design of the VAX 4000 Model 500 system. In the memory subsystem, the NMC handles all NDAL memory references by transferring them over the 64-bit NMI. The NMC supports data transfer rates up to 58.5MB per second over the NMI when used with a 74.4-MHz NVAX CPU. Memory is configured in sets; each set contains two banks of interleaved 64-bit wide memory. External multiplexers and transceivers are required to perform interleaving. The NMC provides most of the memory control signals, and only simple bank selection logic is required externally.

The Model 90 memory subsystem implements two sets of memory using the same 36-bit wide SIMMs that are used in the Model 60 system. Four SIMMs are required for each set. By using either the 4MB or 16MB SIMMs, the Model 90 allows memory configuration sizes of 16MB, 32MB, 64MB, 80MB, or 128MB.

Due to module space constraints and cost concerns, we investigated alternatives to the four GMX memory data path chips used on the VAX 4000 Model 500 memory modules. We determined that two low-cost gate arrays designed for the VAXstation VLC could be used instead. These gate arrays provided the same multiplexer and transceiver functions found in the GMX chips. Because the NMI on the Model 90 consists of only two loads, the high-drive capability of the GMX chips was not required. We used a simple PAL to decode the bank selection signals from the NMC and to generate the control signals required for the gate arrays.

Because the Model 90 design uses the NMC, we received an additional benefit of having error correction code protection at no additional cost to the system. The NMC implements a single-bit error correct, double-bit error detect code (SEC/DED) across every 64-bit word of memory data. The eight bits of error correction code replaced the eight bits of parity used on the Model 60.

I/O Subsystem

Given that the Model 90 system was an upgrade to the Model 60, a requirement of the I/O subsystem design was to provide support for all I/O devices/options found on the Model 60. The Model 60 I/O design consisted of an interface to a 16-bit bus known as the EDAL where most of the system I/O devices resided. The Model 60 also supported a TURBOchannel adapter that connected to a 32-bit CDAL bus.

The main task of the Model 90 I/O subsystem design was to provide an interface between the two 32-bit CP buses provided by the NCA and the 16-bit EDAL bus and the 32-bit TURBOchannel adapter option offered on the Model 60. The design work necessary included a small PAL design for the TURBOchannel interface on the CP2 bus and the design of two programmable gate arrays for the interface between the 32-bit CP1 bus and the 16-bit

EDAL. The following list describes the Model 90 I/O devices and options and explains why each was chosen.

- Ethernet—The Model 90 Ethernet interface is implemented with the second-generation Ethernet controller (SGEC), which provides an Ethernet connection through a ThinWire or thick-wire cable, selectable by a switch on the rear of the system box. The SGEC, which connects to the CP1 bus and was used on the VAX 4000 Model 500 system, facilitates scatter/gather mapping and dual internal FIFO buffering. We chose the VAX 4000 Model 500 design to implement an Ethernet controller because it required no new logic design.

- Small Computer System Interface—The SCSI bus interface is implemented using the NCR 53C94 SCSI controller chip that was used on the Model 60.[4] The NCR 53C94 device connects to the EDAL bus and performs DMA operations to and from main memory in concert with the two programmable gate arrays known as the CEAC and SQWF. DMA virtual-to-physical address translation is performed by the SQWF chip based on 8,192 mapping registers implemented in external static RAMs.

- Serial Lines—The DC7085 quad universal asynchronous receiver/transmitter (UART) chip was chosen to provide the Model 90 with four serial lines for the keyboard, mouse, modem, and printer/console ports. The DC7085 provides a 64-entry FIFO queue that is shared by all four receive lines and is implemented in a small external SRAM.

- Sound—The Model 90 sound functionality is implemented using the AMD 79C30 sound chip just as it was in the Model 60. The programmed I/O interface to this device allows both record and playback functions through a jack on the front panel, and provides voice-quality sound.

- TURBOchannel—The Model 90 provides a single slot into which any TURBOchannel option that is supported by the VMS operating system may be installed. On the Model 60, the TURBOchannel adapter was designed to interface to a CDAL that was not a complete implementation of the general-purpose CDAL bus. For the new design, a small amount of interface logic was necessary to adapt the TURBOchannel option to the CP2 bus.

- Synchronous Communications Option—The Model 90 supports the same multiple protocol-communications option that is offered by the Model 60. This interface was implemented on the EDAL bus and allows use of synchronous wide-area network communication through protocols such as high-level data link control (HDLC) and synchronous data link control (SDLC).

- Miscellaneous EDAL Devices—The other devices and registers on the Model 90 16-bit EDAL are a 16-bit system configuration register, an 8-bit light-emitting diode register, an Ethernet identification ROM, and a watch chip. All of these devices also existed on the Model 60 EDAL bus and were accessed in a similar manner.

CEAC and SQWF Chip Designs

One of the major pieces of design work required for the Model 90 I/O subsystem was to interface the 32-bit CP1 bus to all the I/O devices that reside on the 16-bit EDAL bus. This interface was partitioned into two tightly coupled designs called the CEAC and SQWF. The CEAC chip is primarily responsible for handling control of I/O register read and write requests from the NCA to the various devices on the EDAL. The SQWF chip handles DMA transfers and buffering of data between the SCSI controller chip and the NCA.

The CEAC chip, which was first implemented in a Xilinx 3090 FPGA and later converted to a conventional gate array, is a 3,400-gate design and uses 119 I/O pins of a 160-pin plastic quad flat package (PQFP). It performs the CP1 bus arbitration between the SQWF for SCSI DMA, the SGEC for Ethernet DMA traffic, and the NCA for I/O register access. The CEAC responds to NCA I/O accesses that are directed at internal CEAC/SQWF registers and EDAL device registers. Its slave sequencer controls read, write, and chip-select signals that control EDAL devices. The CEAC has CP1 and EDAL multiplexing logic which selects between addresses and data and is controlled by the slave sequencer. The CEAC chip contains an interrupt controller which consists of interrupt request and mask registers, priority decoding logic, and interrupt vector generation logic. The CEAC also has a master sequencer that supports the SQWF during transfers of DMA data on the CP1 bus.

The SQWF chip, which was first implemented in a Xilinx 4005 FPGA and later converted to a conventional gate array, is a 3,900-gate design and uses

110 I/O pins of a 160-pin PQFP. The SQWF responds to requests from the NCR 53C94 SCSI controller chip to do DMA transfers. During SCSI DMA, the SQWF chip helps to optimize utilization of the CP1/NDAL/NMI buses by buffering up to eight bytes of data in either direction. The SQWF performs byte swapping to map the NCR 53C94's 16-bit transfers to arbitrary main memory byte boundaries. The SQWF contains a 22-bit main memory address byte counter and a direction bit which are accessible as registers in I/O space. The SQWF chip also performs DMA virtual-to-physical address translation by referencing an 8,192-page address map store based in external SRAM.

Graphics Subsystem

One of the keys to producing a workstation around the VAX 4000 Model 500 core was the ability to integrate graphics support into the system successfully. In addition, maintaining the high level of graphics performance found in the Model 60 was viewed as an important goal. The Model 60 offered three very good graphics options. The Model 60 low-cost graphics (LCG) option features an inexpensive frame buffer module and two-dimensional graphics acceleration logic contained within a large gate array on the system module. The other Model 60 graphics options, SPXg and SPXgt, are three-dimensional graphics accelerators. The SPXg is an 8-plane option, and the SPXgt is a 24-plane option. The three-dimensional graphics options simply replace the LCG frame buffer in the Model 60. We realized that the Model 90 system had to support a high-performance, entry-level, two-dimensional graphics option and the three-dimensional SPXg and SPXgt options. The first major task in the design of the Model 90 was defining the entry-level graphics option.

LCSPX Graphics Option

From the start of the Model 90 project, we knew we could not support LCG. The LCG control logic on the Model 60 was embedded within a very large gate array that also served as a memory and I/O controller. This part was not compatible with our core architecture. Redesigning the Model 60 LCG logic to fit our system would have been a major design task requiring a midsize gate array. This was well beyond our engineering schedule and resources.

To find a graphics option that would provide the desired performance and have a low hardware and software development cost, we met with a number of graphics hardware and software engineers. We found that a new X terminal, the VXT 2000, was being developed with graphics based on a cost-reduced version of the SPX graphics module originally used in the VAXstation 3100 system. This module was close to the Model 60 LCG in both cost and performance. In addition, it was designed to interface directly to a CDAL bus and was software compatible with the VAXstation 3100 SPX. As a result, the module could interface directly to our CP2 bus with a minimal number of changes to its supporting software.

To use the VXT 2000 SPX module in the Model 90, we needed to lay out the module again to fit the physical constraints of our system. This new module was named LCSPX (low-cost SPX). No logic design work was required on the LCSPX or on the system module to support it. A connector on the CP2 bus provides the interface to the LCSPX module.

Although the performance of the VXT 2000 SPX module was close to that of the LCG on the Model 60, we wanted to extract as much performance out of the LCSPX module as possible. To improve the performance of the LCSPX, we increased the clock speed of the module. A speed analysis of the module was performed to determine how much margin existed in the design. The original VXT 2000 SPX module ran at 20 MHz, and we determined that by upgrading a number of components, the LCSPX could run at 25 MHz. As a result of this 25 percent increase in speed, the performance of the LCSPX module exceeds the performance of the Model 60 LCG for almost all operations.

SPXg and SPXgt Graphics Options

On the Model 60, the SPXg and SPXgt graphics options plug into the LCG frame buffer port, and a subset of the LCG control logic provides access to these options. To support SPXg and SPXgt on the Model 90, a port that emulated the LCG frame buffer port was required. The Model 60 supports both a PIO and a DMA interface to the SPXg and SPXgt, but the Model 90 supports only a PIO interface.

We considered a DMA interface for the Model 90, but discarded the idea for several reasons. A DMA interface similar to the Model 60, which supports virtual DMA, requires more logic than would fit in the programmable technologies we were considering for the Model 90. A simpler DMA interface would not have been compatible with VMS graphics system software and would have required a large

number of changes to the software. Finally, it appeared that processing on the SPXg and SPXgt modules, and not data bandwidth, was the limiting factor in performance in the Model 60 system. Based on this analysis, a high-speed PIO interface was built.

The SPXg/SPXgt interface on the Model 90 simply translates CP2 bus read and write commands into frame buffer port transactions. The interface is pipelined such that it can keep up with the peak transfer rate of the CP2 bus. We implemented the majority of SPXg/SPXgt interface logic using two AMD MACH PALs. One of these large PALs contains the control sequencer and generates all CP2, frame buffer port, and data path control signals. The other MACH PAL contains an address decoder and address data path. A few miscellaneous medium-scale integration components make up the remainder of the interface. Performance analysis of the SPXg and SPXgt modules shows that performance on the Model 90 is virtually the same as on the Model 60.

Physical Design

The physical design of the Model 90 system board presented many challenges. Being a module upgrade from the Model 60, the Model 90 used a system board that had many fixed-position obstacles for placement and routing, such as connectors and stand-off post holes. In addition to the seven connectors and the single switch along the back of the unit, seven more connectors scattered about the module had to retain their positions. Also, the Model 90 had to fit eight SIMMs in the same area that the Model 60 had six SIMMs. Furthermore, programmable technologies generally provide logic of less density than conventional gate arrays, and therefore require more module space. To meet these challenges, we eliminated on-board main memory (8MB were present on the Model 60) and reduced the size of the secondary cache from the originally planned 512KB to 256KB.

The Model 90 system board measures 16 inches by 10.5 inches, and has 8 layers of etch, approximately 100 surface-mount and through-hole components, 23 connectors, 5 oscillators, and over 300 discrete resistors and capacitors. All components are mounted on a single side. Figure 2 shows the Model 90 system module.

Model 90 Breadboard System

Our logic verification strategy depended on building a breadboard early in the design cycle. This breadboard allowed quicker and more accurate hardware verification than logic simulation. In addition, the breadboard allowed debugging of console and VMS software earlier than a conventional prototype.

The breadboard system was based on a VAX 4000 Model 500. Logically, the breadboard simply extended the CP1 and CP2 buses of a VAX 4000 Model 500 system to include the complete I/O and graphics subsystems of the Model 90. The Model 90 breadboard was an eight-layer etch module and included all the devices on the Model 90 CP1, CP2, and EDAL buses. The breadboard system used a VAX 4000 Model 500 test backplane that allowed complete physical access to both sides of the VAX 4000 Model 500 CPU module. A socket with pins that extended 1 inch through the back of the module was used on the NCA chip of the VAX 4000 Model 500 CPU module. The breadboard, which contained the holes for the NCA, was then attached to the VAX 4000 Model 500 CPU module by soldering it to the extended socket pins. CP bus clocks were not directly routed to the breadboard logic. To control clock skew, a phase lock loop (PLL) was used on the breadboard to regenerate the CP bus clocks. With this configuration, the breadboard system was able to run at full speed.

Once the breadboard system was assembled, we were able to execute console commands after a quick debugging of the system. At this time, very little of the breadboard logic was being used because the console program was using the VAX 4000 Model 500 I/O devices and not the Model 90 devices. The hardware team began debugging the breadboard logic piece by piece. Debugging was quick because a completely functional console and I/O system already existed. Simple functions, such as register reads and writes, were debugged using the console examine and deposit commands. More complex functions, such as reading and writing to an SCSI disk, were tested by writing test programs in VAX MACRO, downloading them into memory, and executing them using the console.

After some of the major pieces of functionality were verified by the hardware group, members of the VMS group began to use the breadboard. A modified version of the VMS operating system was used to debug VMS device drivers. Drivers for the serial lines, LCSPX, SPXg, SPXgt, and the SCSI port were debugged. In addition to software debugging, this effort provided the software to perform

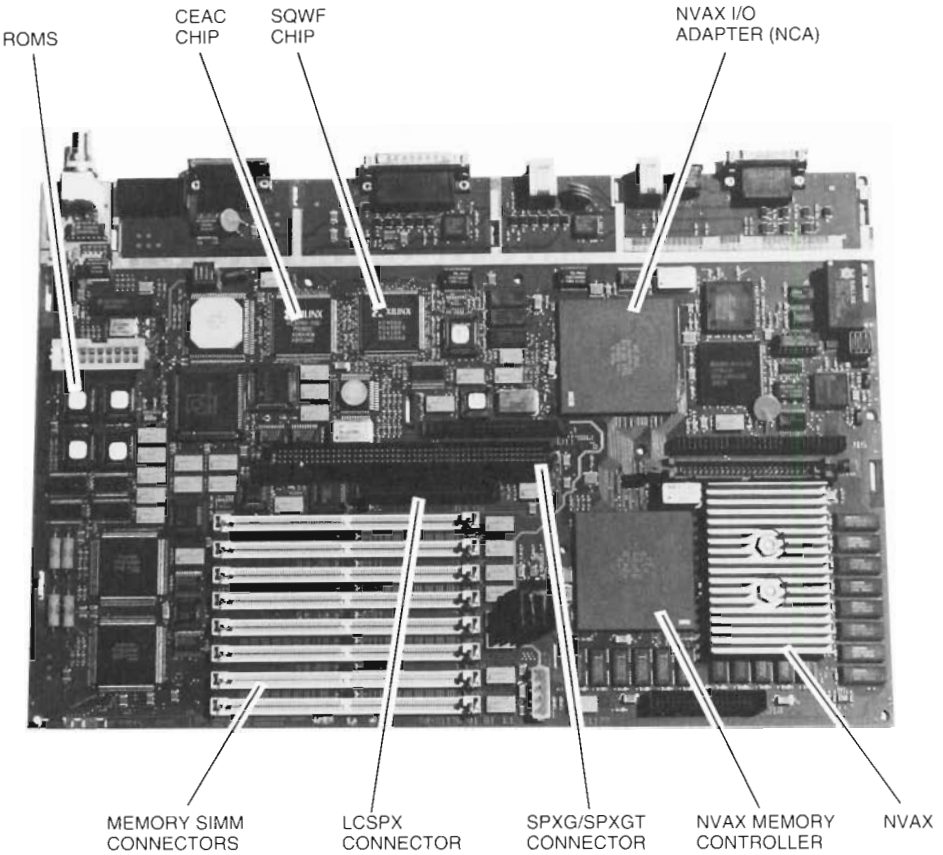


Figure 2 VAXstation 4000 Model 90 System Module

extended verification. The hardware group was able to use graphics test packages running under DECwindows software, disk exercisers, system exercisers, and other tools supported by the VMS operating system. This provided a verification environment we could never achieve with traditional simulation methods.

At this point, we were still using the VAX 4000 Model 500 console. The breadboard was then used to debug the Model 90 console code. We disabled the system support chip, which controls much of the console support hardware in the VAX 4000 Model 500, and began using the Model 90 console support hardware. A base console that included minimal power-up self-test, basic command support, and SCSI boot support was debugged by the Model 90 console team. Once the console was functional, the VMS group returned and debugged boot support for the Model 90 using the breadboard. When this was finished, the software was completely debugged and ready to be loaded onto the first Model 90 prototype.

As soon as we assembled the first Model 90 prototype system, we realized the benefits of all the work performed using the breadboard system. During the first day of debugging, we ran the console program and booted the VMS system with minimal effort. We also ran DECwindows software using the LCSPX and the SPXg and SPXgt graphics options. This quick debugging allowed additional prototype systems to be built immediately and shipped to various development and verification groups throughout the company.

Performance

The VAXstation 4000 Model 90 represents the fastest VAX workstation ever produced. Its CPU performance surpasses previous VAX workstations and is comparable to Digital's RISC-based workstations. By utilizing the NVAX CPU chip, the Model 90 achieves 2.7 times the performance of the Model 60 when measured against the SPECmark benchmarks.[5] Table 2 gives the CPU performance of the

Table 2 CPU Performance Comparison

Workstation	SPECmark* Rating
VAXstation 4000 Model 90	32.7
VAXstation 4000 Model 60	12.0
VAXstation 3100 Model 76	6.8
DECstation 5000 Model 240	32.4

Note:
*SPECmark is a quantitative measurement of performance, determined by running a suite of ten benchmark programs.

VAXstation Model 90 compared to other Digital workstations.

LCSPX is the entry-level, two-dimensional graphics option offered on the Model 90. The performance of this option is better than the LCG option offered on the Model 60 for most graphics operations. Table 3 compares the LCSPX graphics performance to Digital workstations using standard two-dimensional metrics.

SPXg and SPXgt are high-performance, three-dimensional graphics accelerators offered on both the Model 60 and the Model 90. Table 4 compares the three-dimensional graphics performance of several of Digital's workstations using standard three-dimensional metrics. In addition, Table 5 gives three-dimensional performance using the picture-level benchmark (PLB) suite.

Summary

The NVAX CPU chip provides the high performance that makes the VAXstation 4000 Model 90 competitive in today's market. The design methodology used during the project allowed us to develop and ship the Model 90 quickly and to provide a simple upgrade path for existing VAXstation customers.

Acknowledgments

The authors would like to acknowledge the following people for their contributions to the VAXstation Model 90 development: Gregg Bouchard, Paul Campos, Jon Crowell, Terry Furman, Dave Ives, Bill Laprade, Jim Lundberg, Curt Miller, Jan Nordh, Jim Reilley, Brian Rost, Mike Sullivan, Pat Sullivan, Mike Warren, and Tom Wenners.

Table 3 Two-dimensional Graphics Performance Comparison

Workstation	Two-dimensional Area Fill (Mpixels per second)	Two-dimensional Vectors (Kvectors per second)
VAXstation 4000 Model 90 LCSPX	18.2	266.0
VAXstation 4000 Model 60 LCG	14.6	216.0
VAXstation 3100 Model 76 SPX	14.2	183.0
DECstation 5000 Model 240 PXG	13.9	263.0

Table 4 Three-dimensional Graphics Performance Comparison

Workstation	Three-dimensional Polygons (Kpolygons per second)	Three-dimensional Vectors (Kvectors per second)
VAXstation 4000 Model 90 SPXgt	33	295
VAXstation 4000 Model 60 SPXgt	33	300
VAXstation 4000 Model 90 SPXg	30	295
VAXstation 4000 Model 60 SPXg	30	295
VAXstation 3100 Model 76 SPX	6	57
DECstation 5000 Model 240 PXG	52	302

Table 5 PLB Graphics Performance Comparison

	GPCmark PLBlit Results*				
Workstation	Printed Circuit Board	System Chassis	Cylinder Head	Head	Shuttle
VAXstation 4000 Model 90 SPXgt	13.2	11.8	8.5	8.3	13.5
VAXstation 4000 Model 60 SPXgt	12.3	11.1	8.4	8.5	12.5
DECstation 5000 Model 240 PXG	10.0	11.7	14.9	19.2	18.3

Note:

*GPCmark is a quantitative measurement of performance, determined by dividing a normalizing constant by the elapsed time, in seconds, required to perform the test.

References

1. G. Uhler et al., "The NVAX and NVAX+ High-performance VAX Microprocessors," *Digital Technical Journal*, vol. 4, no. 3 (Summer 1992, this issue): 11–23.
2. J. Crowell et al., "Design of the VAX 4000 Model 400, 500, and 600 Systems," *Digital Technical Journal*, vol. 4, no. 3 (Summer 1992, this issue): 60–72.
3. *TURBOchannel Hardware Specification* (Palo Alto, CA: Digital Equipment Corporation, TRI/ADD Program, 1990).
4. *Small Computer System Interface (SCSI)* (New York: American National Standards Institute, ANSI X3.131–1986, 1986).
5. *Digital's VAXstation 4000 Family Performance Summary, Version 3.0* (Maynard: Digital Equipment Corporation, 1992).

Brian Porter

VAX 6000 Error Handling: A Pragmatic Approach

The VMS operating system's CPU-dependent support of the VAX 6000 family of computers implements a complex and sophisticated set of error-handling routines. At the start of a VMS session, these routines help construct the necessary framework to support the I/O subsystem as the system begins to emerge. For much of a VMS session, these routines then lay dormant within the SYSLOA image. Periodically, when aroused, they peer into hardware registers looking for signs of trouble. Often, all is well, and the routines return to hibernation. On those occasions when the hardware requires assistance, error handling takes complete control of the system. It has but one mission: identify the error; recover if possible, but at all costs ensure that the integrity of the system remains intact and that data is preserved.

Error handling is the set of routines that resides in the CPU-dependent loadable image known as SYSLOA. Each processor model that supports the VAX system architecture and VMS operating system has its own SYSLOA image. Error handling is implemented with other common routines like console support and secondary processor start-up. Error handling is unique for each processor model. Individual processor models bring with them a wealth of error detectors and consistency checkers. Each device has to be independently interrogated and reset once triggered.

Error handling of one form or another resides throughout the VMS operating system. In some contexts, trying to edit a file in a directory structure that does not exist can be considered an error. This paper discusses only errors that deal with the underlying CPU and memory hardware on which the VMS system is running. It describes the development of error handling to support the CPU modules and memory controllers that make up the system kernel in the VAX 6000 series. This paper explains our error-handling strategy to not only reduce the amount of unique coding, but also provide an opportunity to enhance, mature, and improve existing VAX 6000 products.

Development of Error-handling Routines for the VAX 6000 Platform

The VAX 6000 platform provided a unique opportunity to develop error-handling routines. As shown in Figure 1, the XMI backbone of the system allows the creation of increasingly powerful systems that retain much of their operating characteristics. Increases in processor capability are gained by merely exchanging processor modules for more powerful models. We decided that error handling should not be any different. On prior systems,

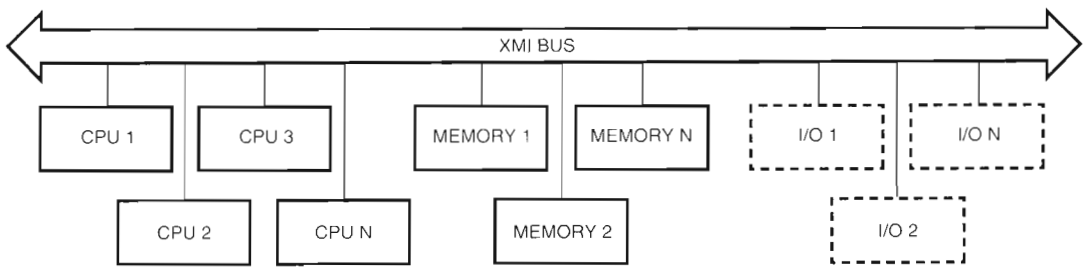


Figure 1 System Block Diagram

a complete set of error-handling routines for each CPU model had to be implemented. We adopted an approach to error handling that could be carried forward from one processor to the next with little or no change to the initial error-handling model. This approach handles identical errors in the same way with the same code base.

The protocol of the XMI bus was modified to allow support of write-back caching schemes of the VAX 6000 Model 500 and VAX 6000 Model 600. However, this had no ill effect on the overall error-handling model we decided to use in the support of the VAX 6000 family of processors.

VAX 6000 Family Error Delivery

Identical mechanisms were used to structure error delivery on each processor in the VAX 6000 family. Each processor has two system control block (SCB) interrupt vectors and a single SCB exception vector. The interrupt vectors deliver hard and soft errors. The exception vector delivers machine check exceptions.

Hard Error Interrupts Hard errors can be categorized in the following way. Hard errors occur as conditions that are not synchronous to the program counter (PC). In almost all instances, systems cannot recover from hard errors. They indicate that data or machine state has been lost. Hard errors are normally fatal. Hard errors are delivered through SCB vector 60 (hexadecimal); interrupt priority level (IPL) is raised to 29 decimal.

Soft Error Interrupts Soft errors, on the other hand, generally signal an asynchronous condition, with respect to the PC, that has been corrected by hardware, or that can be overcome with some software intervention. Soft errors are normally always benign to system operation. Soft errors are delivered through SCB vector 54 (hexadecimal); IPL is raised to 26 decimal.

Machine Check Exceptions Machine check exceptions are internal processor conditions that are synchronous to the PC. If the condition can be corrected when the instruction that caused the exception is reexecuted, the result is the same as if the condition had not occurred. Many of the machine check exceptions that are reported by the VAX 6000 family of processors allow recovery so that normal operation can continue. Machine check exceptions are delivered through SCB vector 4; IPL is raised to 31 decimal.

Objectives

Error handling must identify the error and recover if possible. Above all, it must guarantee the integrity of the system and the preservation of data.

An important project goal was to produce a robust and quality product that would have predictable performance. We chose to have a single error-handling model that could be implemented for all VAX 6000 CPU models. We also adopted an implementation methodology that included the capability to allow rigorous testing of the many code paths contained in the various configurations. To accomplish this goal, we designed the test and verification strategy in conjunction with the overall system design of the kernel error-handling subsystem. In addition, we designed and implemented an object-oriented code base for errors that are common across the platform. Errors are handled in this way when they are associated with main memory, with XMI bus protocols, or with the support of vector processors.

Most frequently occurring errors are associated with main memory. The error handling for main memory is composed of three major functions. The first handles the complexity of support for two different memory controller types and their internal error conditions. The other two functions are logically split between single-bit error correction code (ECC) failures and double-bit ECC failures.

Common error-handling interfaces and routines were established for the VAX 6000 family of processors. The use of common files and interfaces ensures that errors are handled in exactly the same way for each CPU model.

Full Support of the Symmetric Multiprocessing Paradigm

The VAX 6000 family of CPUs are symmetric multiprocessing (SMP) systems. The error-handling model assumes that more than one CPU is always active. The synchronization of error handling throughout the system has numerous benefits. If an error condition were detected throughout the system, it would be a very complicated procedure to ensure that all CPUs reacted consistently. Such errors would clutter the error log with reports from every CPU and XMI device.

Error Logging Synchronization

In the VAX 6000 scheme, error logging is synchronized across the system. If an error affects all nodes, this information is included with the

first CPU to respond to the error. Machine state is created that informs other CPU nodes that the event has been logged on their behalf. As each CPU node responds to the error condition, it can interrogate this state. In the event that all error conditions have been logged on behalf of a CPU, the error condition is cleared and the interrupt or exception is dismissed. The one entry in the error log for these types of errors clearly indicates that other nodes were active. Information about the nodes affected and state indicating how the node was affected is recorded in the single error log entry.

CPU Configuration Data in the Error Log

A CPU running with some of its hardware disabled may have operating characteristics that cause other CPUs to incur error conditions of some type. An error log entry from a VAX 6000 CPU always includes the configuration of other active CPUs on the system. For example, if the CPU at node 6 is running with its backup cache disabled, other CPUs include this information with their error log data. Thus, potential error conditions can be easily identified.

Error Log Filtering

Some errors that occur at too high a rate are filtered from the error log. Errors that are delivered by the soft error vector are invariably benign to system operation. It is important that they be reported because they can indicate an impending fatal error in some subsystem. However, if these errors are occurring too often, only a subset is sent to the error log. The algorithm is based on an error count over time. If an error is occurring too rapidly, logging of the errors is inhibited. At a later time, logging is reenabled. Errors that do not appear in the error log are still counted, and the accumulated totals are displayed by other error conditions that are sent to the error log.

Message Facility

Error handling on the VAX 6000 has the unique ability to output formatted messages. Integral to the error-handling subsystem is a message processing facility that is composed of specialized routines and modified versions of several VMS system services. The modified system services include SYSFAO and SYSCVRTIM. The message facility provides the error-handling subsystem with the capability to output formatted messages that contain both text and data. These messages are time-stamped and sent to the system console device OPA0:.

Messages can be output in two different modes. Interrupt driven mode is the most common and uses the standard terminal driver functions of the running VMS session. Messages that use this mode describe the disabling of some part of the CPU kernel at system start-up or during the current session. The other mode of output is synchronous and is in line with error processing. This mode is reserved for hardware errors that are nonrecoverable and result in a system crash. The message is output just prior to calling the BUGCHECK mechanism that would terminate the current VMS session abnormally. Messages are always descriptive of the error or exception condition and contain all the machine state available at the time of the error.

Formatted messages allow for errors that occur as the system is being initialized to be reported and described should the system fail to boot. The output of messages is fully synchronized between the primary and secondary CPUs of SMP systems. The primary CPU outputs messages about errors occurring on secondary processors.

Error Rate Checking and Loop Detection

The VAX 6000 family of CPUs provides a great deal of error detection. The error conditions signaled in many cases are benign to the system if the appropriate action is taken. However, blind recovery from errors can be a downfall in itself. It is not uncommon for so many benign failures to occur that error handling is the only task being performed by the system. Error handling on the VAX 6000 family implements a system of rate checking and loop detection to combat this problem.

Rate and Loop Detection Time Base

The timing standard used by the rate checking and loop detection subsystems is the CPU TODR register. The TODR hardware register is independent of software and increments every 10 milliseconds.

Rate Checking of Errors

Each error condition has an associated rate check database. The database tracks TODR values for the three most recent errors. If these errors occur too fast, special action is taken in addition to that required to service the error. This may involve disabling the signaling of the error condition itself. For example, some errors that are reported outside the

CPU can be turned off. When sufficient data about an error has been collected, the error may be disabled for a period of time. Hardware features such as internal cache errors can also be disabled. If cache errors occur that are recoverable, but are occurring too fast, the cache is disabled. The occurrence of multiple errors can indicate a broken structure, whereas a single error can indicate a single transient event.

Loop Detection

Multiple errors of different types can also occur frequently. In this situation, the system is operational, but it is continuously at high IPL, servicing error interrupts or exceptions. This operating scenario is detected by the frequency of transitions in and out of error handling. When error-handling code threads are entered and exited, the TODR value is saved. During execution of error handling, the enter TODR value is compared to the last exit TODR value. If the result is too close, a count is incremented. If the close relationship of exit to entry continues to occur, a loop condition is declared and appropriate action is taken. Most often this means the system is shut down.

Error-handling Model

The traditional approach to error handling in the VMS operating system has been to interrogate registers and act on the data directly in real time. Another approach has been to save only a subset of processor state that has a linkage to the error delivery vector and then act on this data during a later parse operation.

When designing the error-handling model for the VAX 6000 series, we decided to save all CPU state that is visible to macro programmers in buffers specific to each CPU. All interrogations are then made on the data in this buffer. Information on the hardware state is saved as well as the current system time. Any action taken by error handling is also recorded in the buffer. This approach has several advantages. First, a distinct footprint of the last error is contained in the system image in memory. Should the system fail, the data is saved when a crash dump is taken. Second, the many execution thread possibilities are made easier to test and verify. Finally, conditions are easier to diagnose if the original data that error handling processed and the actions that were taken are recorded in an error log.

The error-handling process for the VAX 6000 series consists of six distinct steps:

- Setup and synchronization
- Saving of state
- Parsing of data
- Processing and accounting of state
- Error logging
- Error reset and dismissal

This logical organization provides flexibility to the implementation being addressed at the time. The parsing of data step was added at the same time that support of the VAX 6000 Model 400 was implemented.

Setup and Synchronization

Synchronization is accomplished by acquiring the MCHECK spinlock. The use of spinlocks is a VMS technique that provides atomic access to code threads and data structures and ensures that only one CPU at a time is in error handling. Thus it is possible to compress an error condition occurring throughout the system into a single error log entry by the first CPU to service the error.

Following synchronization, the SMP sanity and spinlock acquisition timers are disabled. If an error occurs at the boundary of one of these timers, a false termination of the session can occur due to the time consumed by the execution of error handling. The SMP sanity and spinwait timers are mechanisms used in VMS to ensure that CPUs active in a multiprocessor system are interactive with each other and the synchronization primitives that control access to various resources. The sanity timer is used as a watchdog timer to ensure that CPUs respond to hardware clock interrupts on a regular basis. Each CPU active on the system monitors another CPU for its response to hardware clock interrupts. The spinwait timer guarantees that one CPU does not retain ownership of a spinlock resource for more than an allotted time period. Error handling is always executed at an IPL above which hardware clock interrupts can be serviced. As a result, it defeats the sanity timer mechanism. Some of the actions taken by error handling can cause a spinwait timer to expire if the error being serviced occurs too close to the timer boundary. By disabling these SMP timers, a time period is started over when the error being serviced is dismissed and timers are reenabled.

The buffer associated with the CPU experiencing the error is initialized to zero and is ready to receive the latest error state. If the error is a machine check,

stack space is allocated and initialized to allow for error dismissal and error-handling exit.

When machine checks or soft error interrupts are serviced, the cache subsystem is unconditionally disabled. Error handling does this to preserve any error state that may be in the cache. If the error happens to be cache-related, the state can be extracted at the appropriate time. Cache-related errors are not reported by hard error interrupts on some VAX 6000 models. When hard error interrupts are serviced on these processor models, the cache is not disabled by software.

The machine check flow has an additional check to determine if the error is associated with a recovery block. Recovery blocks on the VMS system provide kernel-mode macro programmers with the ability to protect an execution thread from the effects of fatal machine check exceptions. Normally when a machine check occurs in kernel mode, the code thread being executed loses control. Unless the instruction can be restarted, the VMS session is terminated. The placement of kernel-mode execution threads within the context of a recovery block ensures that a machine check will cause control to be passed to the end of the recovery block, along with status to indicate that the machine check has happened. The macro programmer can select what type of machine check to be protected from. In general, this is limited to those machine checks caused by references to the physical address space that do not respond or return data.

If it is determined that the current delivered machine check is protected by a recovery block and that error handling established this condition, the error is dismissed immediately without further action. More details are given in the following section.

Saving of State

All available CPU error state is saved regardless of error type or delivery mechanism. Machine checks also save the internal state passed by microcode on the stack. Each register is read into its local storage buffer within the context of a recovery block. A valid bit is associated with each local copy register cell according to its status as it exits the recovery block context. This is important because each cell has been initialized before use. A register value of zero may be significant, and a failed register read would allow the initialized value to be interpreted as not having any error or state bits set. Failed register read indications can help in the diagnosis of the original error condition.

The recovery blocks used when error state is collected have special flags to indicate that they were established by error handling. If an error does occur, control is returned to the appropriate point in the error-handling execution thread. The original saved state of the first error is not overwritten.

Parsing of Data

The parsing of data step was added to support the VAX 6000 Model 400 and later models. The data collected in the saving of state routines is parsed as a separate step. When error data is parsed and processed in a single step, as in the VAX 6000 Model 200 and VAX 6000 Model 300, it is difficult to make the necessary errors invisible to the error log. When the error data is parsed and an error mask is produced that represents the error conditions present, it is much easier to detect if all current conditions have been serviced. Since it is also easier to detect conditions with only one error present, expected error conditions can be processed. The VAX 6000 Model 400 and later models have many benign error syndromes that have their logging filtered.

Processing and Accounting

During the processing and accounting step of error handling, the data in the CPU private local buffer is parsed and acted upon. These routines detect if a specific error condition is global and sensed throughout the system or local to the particular CPU. Global errors include the state from other CPUs and devices in the error log if required. If the global error is the only one present and it is expected, machine state is set to indicate that error logging should not occur. Should this CPU be the first to process the global error, it samples data in registers of the other CPUs and devices and leaves state to indicate the error condition has been serviced and is expected. Consequently, the context of global errors is included into a single entry in the error log. XMI parity errors are serviced in this way. Local errors record only the state from the CPU experiencing the error in the error log.

Error handling supports the notion of expected errors, or errors that sometimes occur as a result of operations performed by error handling. These errors are not reported to the error log. For example, duplicate tag parity errors can sometimes occur when the backup cache on the VAX 6000 Model 200 and VAX 6000 Model 300 is invalidated.

To cause these errors to be invisible to the error log, a mask of error bits to ignore is set up when backup cache invalidate operations are executed. At the same time, a fork process thread that ultimately clears the appropriate mask bit is queued. If an error that would be invisible to the error log occurs, the "ignore-this-error" bit is sampled in the recovery thread for the error condition. If the bit is set, the error is ignored. If the error does not occur, eventually IPL is lowered until the queued fork process runs and clears the bit in the mask. This guarantees that later real errors that have previously been expected and have not occurred are not excluded from the error log.

Error Logging

The data collected by the error-handling routines is sent to the VMS system's error log after it is tallied for size. The amount of record space is allocated by internal VMS routines. The raw data that describes the context of the current error is copied to the VMS error log buffer along with the current values of accounting data for the CPU. The accounting data is a count of the individual error conditions that have occurred on this CPU for the current session. Any CPU that has some part of its functionality disabled includes that data in the error log as well. For example, a CPU that is executing with a disabled cache may cause errors to occur on other CPUs. It is useful to know that a CPU is running in a degraded mode when investigating problems that are occurring on a system. The error log records of all CPUs clearly indicate any CPUs operating at reduced capacity. If all CPUs are running unimpeded, the error log contains a flag to indicate this status.

The amount of data included in the error log for any given error can be different. The data describing the CPU context is the same except in the case of machine checks. These errors also include internal state passed by the microcode through the stack. Depending on the error condition, context from the XMI bus, the memory subsystem, or an external XMI adapter can be included. The error data is organized into various subpackets that are signaled to be present by a flags field contained in a header section of the CPU context packet. For example, an error can occur that describes a failure of a transaction between the CPU and memory. If the data collected from the memory subsystem during the processing and accounting step indicates an error is present, this is included in the error log record. If there is no indication of a memory subsystem error, a flag to indicate that no memory errors are present is set. This reduces the burden on the error log buffers of the VMS system and reduces the clutter and confusion of error registers from a device that does not have an error condition present.

Any error that ultimately causes the system to crash is also logged to the system console terminal through the SYSLOA message facility. Errors can occur during start-up before VMS error logging is available. Errors can also occur and terminate the session before the system completes initialization. For these reasons, fatal errors are always output to the console terminal before the session is terminated. Errors that occur at start-up of secondary processors are monitored by the primary processor. Any output required is done by the primary processor.

Error Reset and Dismissal

The last step of error handling resets error conditions that have been serviced and dismisses the interrupt or exception. The image of the data saved is used as a mask to reset error conditions. This technique guarantees that double error conditions are not lost.

Registers that require initialization are reset using the contents that were read when the error was first serviced. Most error conditions are write-one-clear. That is, to reset the error condition, a mask of the error conditions set has to be written to the appropriate register to clear the error. The use of the original contents of the register as a mask guarantees that an error condition occurring during the processing of an error cannot be lost. Reset of the VAX 6000 Model 200 and VAX 6000 Model 300 error registers includes a later probe of the register for the absence of error indications. Should an error still be present, the error-handling process is restarted and it treats the condition as a new error. After errors are reset, the cache subsystem is invalidated and a check is made to determine if it should be reenabled. Processing of the error could determine that the cache or indeed the CPU should be taken off-line because of an error rate or finite count that is too high. If all is well, the cache subsystem is reenabled. The MCHECK spinlock is now released and the interrupt or exception is dismissed by executing a return from exception or interrupt (REI) instruction.

If the error being dismissed is a machine check, the additional storage allocated by error handling

and error state left by the microcode has to be removed. As shown in Figure 2, the additional storage is an array of quadwords. These quadwords represent program counter/processor status longword (PC/PSL) pairs that direct control to routines that must be executed prior to control being returned to the exception PC. The additional postprocessing that takes place for noncorrectable memory errors and errors that cause a process to be aborted are dispatched using this mechanism.

Machine check processing takes place at IPL 31. Fatal memory error recovery uses the VMS system's page fault code threads. These threads use spinlocks that cannot be acquired from IPL 31. When dismissing machine checks, the PC/PSL pairs are interrogated to determine if they are nonzero. If the probed quadword is zero, the stack pointer is updated to unwind by the quadword allocated. This continues until all three array elements have been probed. If the array element is nonzero, the REI passes control to the PC and PSL described by that array element. Synchronization is thus preserved, and spinlock acquisition rules are obeyed. Eventually the array is traversed, and each element is removed. State left by the microcode is removed, control is passed back to the original exception PC, and instruction retry is attempted. If error handling determines that the execution thread should be aborted, the original exception PC is replaced by a PC/PSL pair that returns control to VMS exception routines. From these routines, control is normally passed to an appropriate mode condition handler. If a condition handler has not been established, the VMS process is aborted. Kernel-mode threads that experience fatal machine checks always result in the termination of the VMS session.

Support of the VAX 6000 Model 200 and VAX 6000 Model 300

The error-handling support for the VAX 6000 Model 200 and VAX 6000 Model 300 is identical. These two processor models are the same logical CPU. The VAX 6000 Model 300 is a selected faster component set of the VAX 6000 Model 200.

The VAX 6000 Model 200 system presented a unique problem for error handling because the primary cache is internal to the CPU chip. Errors from the primary cache do not cause an interrupt or exception. These errors can never cause a failure or wrong result should they occur. Because all cache structures on the VAX 6000 Model 200 are write-through, data can be both in cache and in memory, and it is always consistent. If a parity error occurs on either the data or tag section of the primary cache, microcode can always fetch another copy of the data from memory. If a primary cache tag or data error occurs, microcode sets a status bit to indicate the error in an internal processor register. The internal processor register is private to the local CPU. Previous CPUs with this type of error signaling used a polling technique to detect these failures. On SMP systems, only the primary CPU is interrupted on a regular basis to allow polling routines to run.

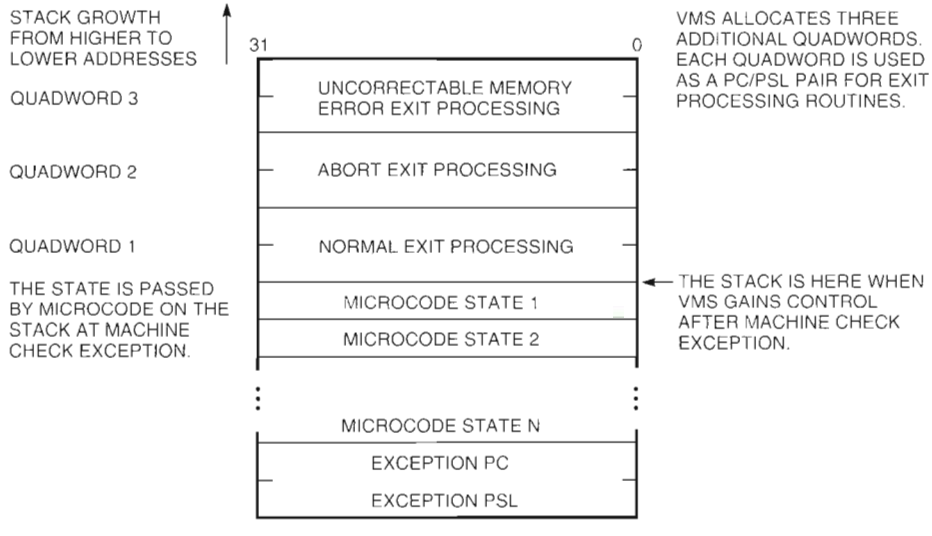


Figure 2 Machine Check Exception Exit Processing Stack Format

Since we had no precedent for reference, we designed a system whereby the primary CPU uses the interprocessor interrupt mechanism to interrupt secondary processors. When the secondary CPU receives the CPU-specific interprocessor interrupt, it reads the appropriate internal processor register, places the data in a known location, and sets an indicator flag. On later poll cycles, the primary CPU sees the indication from the secondary CPU and interrogates the known location for any error bits. If no errors are detected, each secondary CPU is polled once every ten seconds. Should an error be found, the secondary CPU with the error has its polling frequency increased to once every second. If ten successive polls indicate error conditions, the secondary CPU is signaled to disable its primary cache. If this occurs, entries are made in the error log and to the system console by the primary CPU on behalf of the secondary CPU.

During systems integration of the VAX 6000 Model 200, certain random-access memory (RAM) devices used for the backup cache exhibited excessive parity error failures. The problem was so severe that special error-handling software and additional CPU hardware functionality were developed to isolate and diagnose the failures. This work was so successful that the hardware functionality was made a permanent feature of the processor, and the error-handling routines were made a permanent part of the SYSLOA image. The hardware functionality and software routines allowed for the failing data bit in the backup cache to be identified at the time of the failure. The VAX 6000 Model 200 experience had a lasting effect on error handling across the VAX 6000 family. The ability to diagnose cache parity errors to the bit level in the operating system remains a characteristic of error handling on all VAX 6000 systems.

Support of the VAX 6000 Model 400 and VAX 6000 Model 500

Although the CPU chips on the VAX 6000 Model 400 and VAX 6000 Model 500 are the same, the SYSLOA images are not. The major difference between the two systems is the write-back cache subsystem implemented by the VAX 6000 Model 500. To facilitate write-back cache strategies, the XMI bus was enhanced to support a directory-based broadcast coherence protocol.[1]

The VAX 6000 Model 400 and VAX 6000 Model 500 systems represented a dramatic increase in system complexity for error handling. The amount of error detection incorporated within each increased and became more complex. The overall model implemented on the VAX 6000 Model 200 was maintained, but a step was added between the steps of saving of state and processing and accounting. The new step, parsing of data, was previously a part of processing and accounting. Error handling support of the on-board CPU electrically erasable programmable read-only memory (EEPROM) was also added. The EEPROM was until now used only by CPU console support.

The overall model now became

- Setup and synchronization
- Saving of state
- Parsing of data
- Processing and accounting
- Error logging
- Error reset and dismissal

Storage space for machine check and hard error is shared in the VAX 6000 Model 200 system. However, this support became too complicated to manage. In the VAX 6000 Model 400 and later models, both sets of error state are available in crash dumps.

EEPROM Support

The experience gained from systems integration of the VAX 6000 Model 200 showed that real-time diagnosis by the operating system has many benefits. However, the scheme used by the VAX 6000 Model 200 was cumbersome and recorded only the resulting diagnostic data to the error log. The challenge was twofold: to make the mechanics of cache parity error diagnosis easier, and to make the data more widely available. We achieved both goals by using the EEPROM on the CPU module. The VAX 6000 Model 500 made additional improvements by using both on-board and high-speed RAM and EEPROM.

EEPROM and RAM structures exist within the physical address space of the VAX 6000 family. These structures are primarily used by the console for cross-session storage of data. High-speed RAM is used for general heap storage by the console. RAM and EEPROM structures have physical addresses that are in the I/O region of the address space. Address references to I/O address space do not cause cache lookups. The code threads that perform data extraction were placed in the EEPROM and RAM

structures to avoid special hardware operating modes. A few simple routines enabled easier diagnosis of cache parity error failures and a method of disabling the cache that does not disrupt error state. The error-handling SCB vectors were pointed to the EEPROM so the routines that disable the cache could do so without making cache references. (On the VAX 6000 Model 500, the cache disabling routines are placed in the high-speed RAM.)

When an error occurs, control is first passed to individual routines that reside in I/O space. These routines disable the cache subsystem and then return control to the SYSLOA image in main memory.

The VAX 6000 Model 500 has an error transition mode (ETM), which allows the backup cache to be partially disabled. New blocks are not allocated when in ETM mode. Data requests are filled from the cache. Error interrupts or exceptions on the VAX 6000 Model 500 dispatch to routines that execute from I/O space and place the write-back backup cache into ETM and disable the write-through primary cache.

The EEPROM on both the VAX 6000 Model 400 and VAX 6000 Model 500 is also used to store failure information. When errors occur, a counter that is associated with the specific error condition is incremented. The number of error conditions is finite and fully described by the error mask produced by the parsing of data routines. Writing to the CPU EEPROM is time-consuming compared to writing to main memory. A byte write to EEPROM takes on the order of 15 milliseconds. To avoid this overhead, the EEPROM VMS data actually resides in main memory during a VMS session. As each CPU is initialized by the VMS system, the contents of the VMS area are read into individual CPU memory regions. Updates that are required are made to these regions. When CPUs are stopped or when the system is shut down or has crashed, the region of memory associated with a particular CPU is written back to that CPU's EEPROM. In addition to error information, a count of seconds run in a VMS environment is tallied.

Vector Processor Support

One set of routines supports the VAX 6000 Model 400 and VAX 6000 Model 500 vector processors. These routines are organized in an identical manner to the CPU routines and follow the same steps related to CPU error conditions. During the processing and accounting of CPU error conditions, a check determines if any vector processor errors are present. If vector errors are detected, the appropriate support routines—soft error, hard error, or machine check—are invoked.

Error handling supports a maximum of four vector processors. If the number of errors or the rate of errors becomes too great, vector processors are removed from use. Error handling never removes the only or last remaining vector processor. Support of vector processor errors has the same characteristics as support for CPU-related error conditions. Portions of the vector processor are disabled if the associated error rate becomes too great. If other errors continue, the unit is removed from use. The notions of expected errors and errors that are invisible to the error log also exist.

Support of the VAX 6000 Model 600

Error checking and detection on the VAX 6000 Model 600 are very complex processes. There are well over 160 unique soft and hard error conditions as categorized by the software. The actual count declared by the hardware is much greater. The disparity results from the way software groups error conditions. The VAX 6000 Model 600 error handling followed the enhanced model implemented on the VAX 6000 Model 400. The state saved for interrogation by VAX 6000 Model 600 error handling consists of 40 internal and XMI-addressable registers. Support of the VAX 6000 Model 600 also included support of the on-board CPU EEPROM for the long-term storage of failure information. The support of the EEPROM was extended to include the history of the cache subsystem performance in previous sessions.

Like the VAX 6000 Model 500 system, the VAX 6000 Model 600 implements a write-back backup cache strategy. The VAX 6000 Model 600 backup cache operates using a directory-based broadcast coherence protocol.[1] Each 32-byte cache block is in one of three states: invalid, valid/read-only, or valid/written. Multiple caches may hold read-only data simultaneously; written data may be held by only one cache in the system at a time. Write privilege for a block must be obtained before modifying the data in that block.

Certain backup cache error conditions are severe enough to disable the cache. The backup cache may contain written data that is unavailable elsewhere in the system. To access that data, the backup cache is put into ETM, a state which allows written data to be accessed by the cache controller, but disallows the use of read-only data.

A cache enters ETM as a function of either software or hardware. The cache is put into ETM by hardware only when cache data may have been corrupted, or when cache data may be inconsistent with data in memory. Thus, correctable backup cache errors do not cause a transition into ETM, but uncorrectable errors do. A parity error on the NVAX data and address lines (NDAL) interface causes the cache to enter ETM because an invalidate or write-back request may have been missed. A cache transition into ETM occurs when a request for write privilege or a write back does not complete successfully on the VAX 6000 Model 600. The state of the cache is likely to be inconsistent with that of memory.

Three requirements govern cache operation during ETM: (1) The state of the cache is preserved as much as possible to allow software to diagnose the problem. (2) Memory references that hit written blocks in the cache are processed, since this is the only source of data in the system. (3) Cache coherency requests from the NDAL are processed normally so that cache state remains consistent with memory.

Although complex, ETM allows the software to choose when and how to disable the cache. To make the process of error handling less cumbersome, the backup cache is unconditionally put into ETM by the software when any error condition is being serviced.

ECC protects both tag and data stores on the backup cache on the VAX 6000 Model 600. Correctable ECC errors in the backup cache have a record of failed syndromes kept by error-handling routines. Should the same syndrome fail on more than one occasion in a single VMS session, the backup cache is disabled.

If uncorrectable backup cache errors occur, the error-handling routines determine if the block is owned by the CPU and attempt to flush the block back to memory. If successful or if the block is not owned, the backup cache is disabled before returning the system to normal operation. If the data cannot be recovered, the VMS session is terminated.

If the backup cache is disabled by error handling for any reason, that fact is recorded in the CPU EEPROM. Records on disabled status are also kept for the primary cache and virtual instruction cache (VIC). Subsequent sessions of VMS interrogate the EEPROM and cause these structures to remain disabled if they were disabled in a previous session. When this occurs, an appropriate message is sent to the console terminal during system start-up.

Tag parity errors that occur in the VIC are diagnosed in an unusual manner. Unlike other caches on the VAX 6000 Model 600, the tag store of the VIC contains a virtual address. To determine which bit has failed when a parity error occurs, the tag store is probed to retrieve the contents of the failing tag location. The associated data store location is probed to retrieve its contents; each bit in the bad tag address is then flipped in turn. As each bit is flipped, the range of the resultant virtual address is compared to page tables to determine its validity within current context. The virtual address is translated, and the resulting physical address is mapped to allow error handling to read the contents of the page. The appropriate contents of the newly mapped page are compared to the contents read from the VIC data store. If one and only one match is found, the failing tag bit is identified. Masks of failing bits from all VAX 6000 Model 600 cache structures are stored in the CPU EEPROM along with other failure information.

The instruction pipeline complicates VAX 6000 Model 600 error handling. In many instances, errors experienced are in no way related to the current instruction being executed or interrupted. When an error does occur, care must be taken to fully understand in what context the error has an effect.

Correctable Memory Errors

Correctable memory errors are data errors that are corrected by the memory controller before data is returned to the requester. They occur primarily because of alpha particle radiation, affect only a single cell, and are transient in nature. Correctable memory errors are completely benign. To determine if a memory controller reporting correctable errors has real defects, multiple errors must be viewed.

VAX 6000 error handling implements a scheme whereby error data reported by memory controllers for correctable errors is compressed into a structure called a footprint. The footprint reduces the data reported into a form that uniquely describes the error that just occurred. The intent of the footprint is to uniquely index the source component of memory, the dynamic RAM (DRAM). Hence, for a given memory subsystem, the number of valid footprints would equal the number of DRAMs. Furthermore, the footprint

block maintained per footprint is used to track the context (repeat errors, scrubbing, etc.) of this error as well as other errors that match this footprint ID.

The assumption here is that most correctable memory errors are a result of DRAM component faults, hence the granularity of the unique DRAM. As shown in Figure 3, the footprint forms a 32-bit integer from the XMI node ID, the ECC syndrome of the error, and the memory controller bank in error. The integer is used to locate other correctable errors that have occurred in an internal database. Along with the footprint, the address of the correctable error is passed to a set of routines that performs all processing of correctable memory errors. The database tracks the range of addresses that have experienced correctable errors for the same footprint. This aids in the diagnosis of row and column failures with the DRAMs that make up memory controller storage. On the VAX 6000 Model 500 and VAX 6000 Model 600 systems, memory scrubbing status is also tracked.

Memory scrubbing is a technique for reducing the number of error interrupts from locations that are reporting errors caused by alpha particle disturbance. Scrubbing removes transient faults from the system, which in turn reduces the number of interrupts that result from such errors. In addition, it helps to differentiate transient errors from permanent DRAM component faults, as captured in the error log. This information was previously unavailable.

When the VAX 6000 memory controller detects a correctable memory error, circuitry in the controller corrects the data returned for the request.

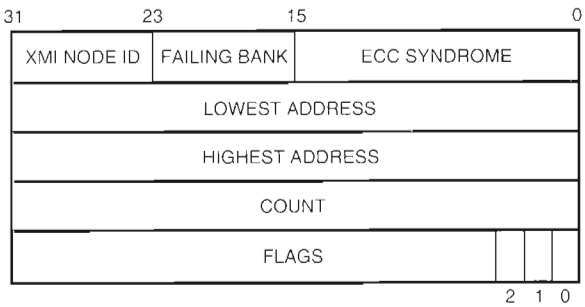


Figure 3 Memory Correctable Error Footprint Structure

The data is not corrected in the storage DRAMs on the controller. If the location is read over and over again, the same error and correction cycle occurs each time. This continues until the location is updated with write data. An interrupt can be generated for each error correction cycle. Care must be taken when scrubbing memory locations. The data in any given memory address can be shared by any number of CPUs or I/O devices. When this is the case, a higher-level software protocol is normally used to synchronize access. Error handling would not be privy to these protocols. VAX 6000 Model 500 and VAX 6000 Model 600 memory scrubbing is possible because of the XMI2 bus protocol. Before a CPU can modify any location in memory, it must be the exclusive owner of the 32-byte block in which the address resides. Ownership is effected at the primitive hardware level and so exclusive access is guaranteed.

When a correctable error interrupt occurs on a VAX 6000 Model 500 or VAX 6000 Model 600 system, error handling rewrites the failing location with its contents. The ability to cause an interrupt is disabled in memory controllers that continue to report errors with the same footprint or that have not responded to scrubbing. This action occurs after sufficient data has indicated that something other than alpha particle disturbance has occurred and the memory controller may require service.

The rate of correctable memory error interrupts is checked to reduce the burden on the system. If the rate of errors occurring becomes too high, the ability to interrupt is disabled at the problem controller for a period of time. Correctable memory error data collected during a VMS session is sent to the error log at the end of the VMS session.

Uncorrectable Memory Errors

Uncorrectable memory errors experienced by the CPU are reported as machine checks. These machine checks are synchronous with the PC making the reference. Uncorrectable memory errors occur when data is lost by the memory controller and cannot be re-created by its ECC circuitry; fortunately, these errors seldom occur. Uncorrectable memory errors represent a serious problem to the execution thread that experiences them. The hardware cannot assist in the recovery of this type of error; recovery is totally a software function.

If the page that experiences an uncorrectable error is a process private page that has not been modified, and the code thread currently executing

is at pageable priority, the error is not considered fatal. The error-handling routines arrange for the page to be re-created in a different physical page in memory by invalidating the necessary memory management structures. As a result, a translation-not-valid exception occurs when the instruction that experienced the exception is retried. The page fault mechanisms of the VMS system do the actual re-creation. The original page with the error is put on a list of bad pages internal to the VMS system. If the page does not meet the criteria for replacement, either the process is deleted or the VMS session is terminated. If the process is deleted, the page is marked "bad" by error handling, and the process run-down routines in VMS retire the page to the bad page list.

Testing

Early in the project, we decided the ability to test and verify had to be built into error handling to produce a predictable, robust, and quality product. Although the VAX 6000 family and CPUs in general have a number of features that allow errors to be generated, they tend not to be general-purpose. In most cases, they are designed for use by special diagnostic software that does not operate in the context of an operating system, e.g., the VMS operating system. We chose to implement a scheme whereby errors would be simulated in software on the target hardware. This approach gave us several clear advantages. The most important was that the approach could be extended as the power and complexity of CPU models increased and that complete control was with the designers. No special hardware equipment or CPU feature would be required. The only precondition was that certain software implementation guidelines had to be followed to make use of the simulator.

Machine check test (MTEST) consists of two parts, a utility and an error-handling implementation methodology. The methodology consists of using main memory storage as the primary agent that is acted upon by error handling. This method also fit into our model of retaining data in memory. The other requirement was the strategic placement of the DEBUG_TRANSFER macro. DEBUG_TRANSFER expands to produce a code segment that determines if the current error being serviced is an error simulation or not. If it is, data that resides in memory that is being interrogated is modified, in concert with MTEST, to reflect the error condition being simulated. DEBUG_TRANSFER code segments represent synchronization points between an error-handling execution thread and the MTEST simulator.

The MTEST simulator is a privileged image and consists of a user interface, a number of nonpageable internal buffers, and simulator routines. The user interface allows the internal buffers to be selected and loaded with data patterns of the user's choice. The user interface also allows the user to pass control to the SCB vectors of the VMS system. In our case we used the vectors that are the linkage to error-handling routines. Once in control, error handling would execute its model until it reached a DEBUG_TRANSFER code segment. The segment would determine that this was an error being simulated and return control to MTEST. MTEST would then decide if the synchronization point was one for which the user has data. The data would be transferred from the buffer named in the DEBUG_TRANSFER code segment to the address also declared in the segment. By judiciously placing the DEBUG_TRANSFER synchronization points and carefully selecting an appropriate data pattern, we were able to simulate any and all error conditions for the appropriate CPU.

In this way, we were able to verify many complex algorithms and code paths that would have been difficult to exercise. We were also able to verify error handling and error logging from the point of error to the error log file. MTEST can be either interactive or procedure-driven. This aspect allowed us to maintain a library of procedures that could be used at any time to verify that operational characteristics for individual errors had not changed when code paths that affected many error types were modified.

MTEST was the primary tool we used for testing. During the test and verification phase, prototype hardware that had real error conditions became available, and we used these prototypes.

Conclusions

The VAX 6000 family now has a robust and complete set of error-handling routines that accomplished our project goals. In fact, many routines were never before part of the VMS system. These routines include the ability to report complete error context to the system console and the ability to group failures occurring across the system to a single error log entry. An important SMP feature is the ability to recognize and retire failing processors from the active set of a VMS session and allow the session to

continue. These routines and others support the entire range of VAX 6000 CPU models. The object-oriented approach to error conditions not on the CPU module has made support and introduction of newer routines easier. The ability to test at will any or all error-handling routines has been a tremendous advantage.

Acknowledgments

Our success resulted from a number of factors, including the advantages of designing the ability to test into the product. There is no substitution for actually executing a code thread to determine the effectiveness of its design goal. The various engineering groups involved in designing the many 6000 CPUs showed great discipline in producing engineering specifications that met the needs of both hardware and software engineering groups. The many hours spent painstakingly describing intricate details of error conditions and the production of parse trees allowed the structured approach we set out to achieve. Special thanks to Mike Uhler for his parse trees and to Nick Carr, who suggested this paper be written.

Reference

1. G. Uhler et al., "The NVAX and NVAX+ High-performance VAX Microprocessors," *Digital Technical Journal*, vol. 4, no. 3 (Summer 1992, this issue): 11–23.

digital™

ISSN 0898-901X

Printed in U.S.A. EY-J884E-DP/92 11 02 19.0