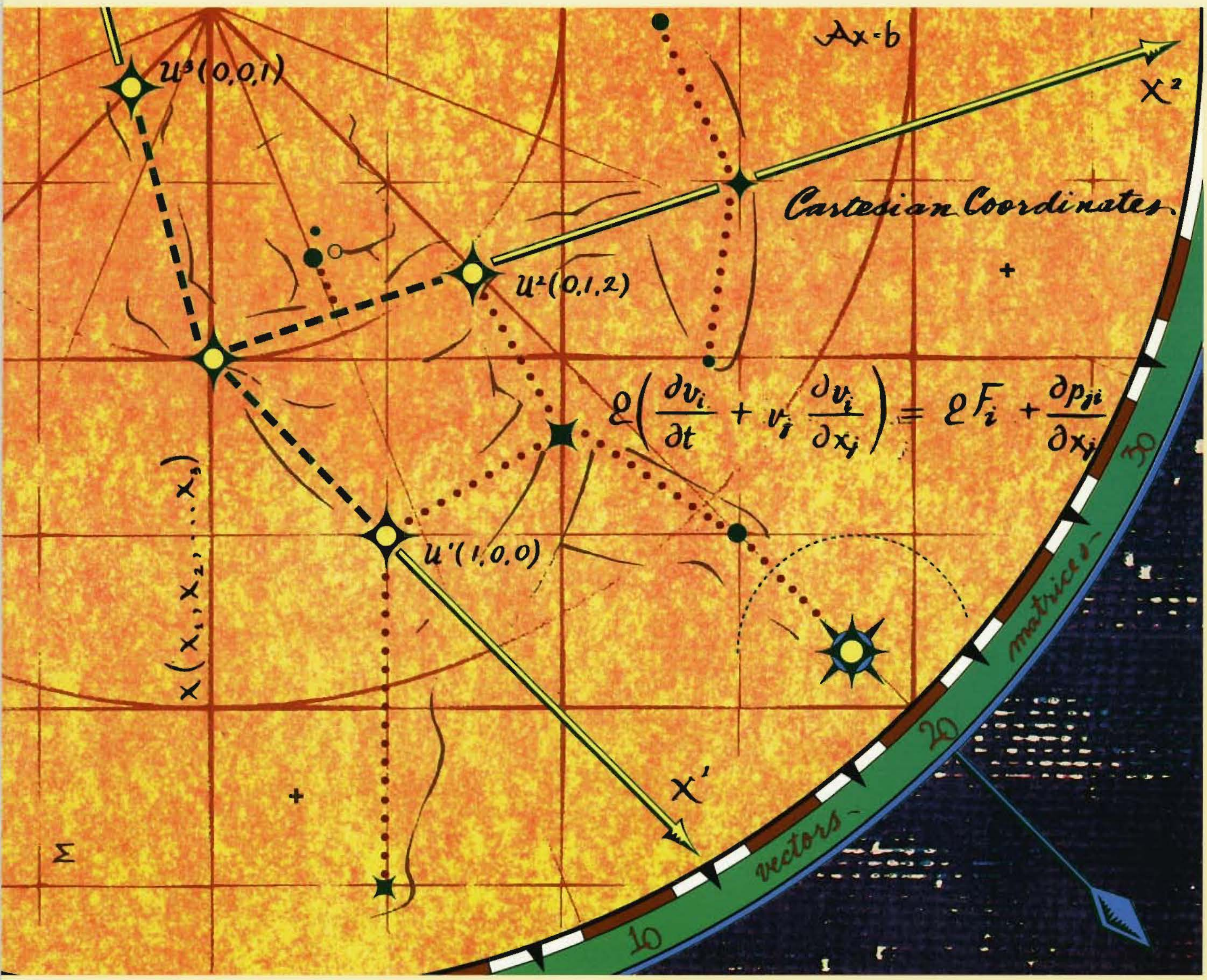
VAX 6000 Model 400 System

Digital Technical Journal

Digital Equipment Corporation



$\mathcal{U}^3(0,0,1)$
$Ax=b$
X^2
Cartesian Coordinates
$\mathcal{U}^2(0,1,2)$
$\varrho\left(\frac{\partial v_i}{\partial t} + v_j \frac{\partial v_i}{\partial x_j}\right) = \varrho F_i + \frac{\partial p_{ji}}{\partial x_j}$
$X(x_1, x_2, \ldots x_3)$
$\mathcal{U}^1(1,0,0)$
X^1
Σ
30
matrices
20
vectors
10

Volume 2 Number 2

Spring 1990

The *Digital Technical Journal* is published quarterly by Digital Equipment Corporation, 146 Main Street MLO1-3/B68, Maynard, Massachusetts 01754-2571. Subscriptions to the Journal are $40.00 for four issues and must be prepaid in U.S. funds. University and college professors and Ph.D. students in the electrical engineering and computer science fields receive complimentary subscriptions upon request. Orders, inquiries, and address changes should be sent to The *Digital Technical Journal* at the published-by address. Inquiries can also be sent electronically on NEARNET to DTJ@CRL.DEC.COM. Single copies and back issues are available for $16.00 each from Digital Press of Digital Equipment Corporation, 12 Crosby Drive, Bedford, MA 01730-1493.

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, cost center, site location code and address, and group name. U.S. engineers in Engineering and Manufacturing receive complimentary subscriptions; engineers in these organizations in countries outside the U.S. should contact the Journal office to receive their complimentary subscriptions. 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-C197E-DP

Book production was done by Digital's Educational Services Media Communications Group in Bedford, MA.

Cover Design
Our cover depicts some of the common equations and terminology used in vector processing, which is one of the featured topics in this issue. The VAXvector processor extends the VAX 6000 family to address the computing needs of numerically intensive applications. The VAX 6000 Model 400 system is a bright star in Digital's family of midrange multiprocessors and this issue's main product theme.

The cover was designed by David Comberg and Randy Ziegler of the Corporate Design Group.

Contents

9 ***Foreword***
Pauline A. Nist

VAX 6000 Model 400 System

11 ***Vector Processing on the VAXvector 6000 Model 400***
Debra L. Slater, David M. Fenwick, D. John Shakshober, and Douglas D. Williams

27 ***The VAX 6000 Model 400 Scalar Processor Module***
Patrick Sullivan, Michael A. Callander, Sr., James R. Lundberg, Rebecca L. Stamm, and William J. Bowhill

36 ***An Overview of the VAX 6000 Model 400 Chip Set***
W. Hugh Durdan, William J. Bowhill, John F. Brown, William V. Herrick, Richard C. Marcello, Sridhar Samudrala, G. Michael Uhler, and Nicholas Wade

52 ***VAX 6000 Model 400 Physical Technology***
John T. Bartoszek, Robert J. Hannemann, Stephen P. Hansen, Robert J. McCarty, and John C. Sweeney

64 ***VAX 6000 Model 400 CPU Chip Set Functional Design Verification***
Richard E. Calcagni and Will Sherwood

73 ***Test and Qualification of the VAX 6000 Model 400 System***
John W. Croll, Larry T. Camilli, and Anthony J. Vaccaro

84 ***Development of the DECstation 3100***
Thomas C. Furlong, Michael J. K. Nielsen, and Neil C. Wilhelm

89 ***Compiler Optimization in RISC Systems***
Larry B. Weber

Editor's Introduction

Jane C. Blake
Editor

This Spring 1990 issue marks the second issue to be published on the new quarterly schedule of the *Digital Technical Journal*. This is also the first year that the Journal is available by subscription — a service our readers have asked for and which we are glad to be able to offer.

The Journal will continue to focus each issue on a product theme. In fact, two products are featured in this issue. The main theme is the latest addition to the VAX 6000 family, the Model 400. With its multiprocessing capabilities, this midrange family of systems provides a highly configurable and expandable computing environment. Because the same cabinet, buses, and power systems are used by all family members, systems can easily be upgraded to achieve higher levels of performance. Papers in this issue describe VAX 6000 Model 400 innovations and additions, including a new vector processor and a higher performance scalar processor module, chip set design and verification, physical technology advances, and system test. The second theme comprises two papers related to Digital's workstation development, specifically the DECstation 3100, and compiler optimization in RISC systems.

Opening this issue is a paper on one of Digital's first vector processors. Dave Fenwick, John Shakshober, Debra Slater, and Doug Williams review the design alternatives for the VAXvector 6000 Model 400 processor and describe its function units. They then give examples of how the units combine to deliver high performance for computationally intensive applications.

The Model 400 also has a new scalar processor, with nearly twice the performance of its predecessor, the Model 300. In their paper, Pat Sullivan, Mike Callander, Jim Lundberg, Rebecca Stamm, and Bill Bowhill discuss the module design and give particulars on how difficult signal integrity problems were resolved.

The five system chips that reside on the module are the topic of our next paper by Hugh Durdan, Bill Bowhill, John Brown, Bill Herrick, Rich Marcello, Sri Samudrala, Mike Uhler, and Nick Wade. From their discussions of the chip designs, we learn how the best features of the VAX 8700 ECL-based, pipelined system and of previous VLSI designs were incorporated in the chip set, which achieves a cycle time of 28 nanoseconds.

This fast cycle time was one of several requirements that drove a significant design effort for the physical technology. John Bartoszek, Rob Hannemann, Steve Hansen, Bob McCarty, and John Sweeney describe the technological advances achieved in a number of areas, including tape-automated bonding, semicustomized ceramic single-chip package design, and testability.

The two papers that close this collection of papers on the VAX 6000 Model 400 address chip design verification and system test. Rick Calcagni and Will Sherwood explain the engineers' multipronged approach to design verification, an approach necessitated by the complexity of the chip set. Then, John Croll, Larry Camilli, and Tony Vaccaro present a paper on the methods and tools designed to completely test the interaction of VAX 6000 Model 400 system's hardware and software.

In the last two papers, the topic turns to workstations. Tom Furlong, Mike Nielsen, and Neil Wilhelm provide an overview of the successful project undertaken to build a fast, competitively priced, RISC-based, ULTRIX workstation, called the DECstation 3100. The Journal is fortunate also to have a related paper on compiler optimization in RISC systems by Larry Weber, vice president, MIPS Systems, Inc. MIPS Systems built the RISC chip set incorporated in the DECstation 3100 workstation.

I thank Steve Holmes of the Midrange Systems Business Group for his help in selecting topics on the Model 400, and Gillian Scholes of Digital and Joanne Hasegawa of MIPS Systems, Inc., for their help in obtaining the workstation and RISC papers in this issue.

Jane Blake

John T. Bartoszek John Bartoszek currently manages the Physical Technology Group within the SDE that is responsible for physical technology applications and product designs. John previously managed the PTG physical technology program that spawned the physical technology used on the VAX 6000 Model 400 CPU module. He joined Digital in 1981 and has a B.S. in nuclear engineering from Lowell Technological Institute. He holds two patents for thermal control devices. John has authored several papers on spacecraft thermal control, solar energy, and electronics cooling and interconnect technologies.

William J. Bowhill As a principal engineer with the Semiconductor Engineering Group, William Bowhill is project leader for an engineering team that is designing an execution unit section of a large CMOS-based microprocessor. He has applied for two patents for his design work on the vector interface and backup control chip on the VAX 6000 Model 400 system. Bill also holds a patent for his work in relation to the Model 400's floating point accelerator chip. He joined Digital in 1985. Bill was educated in Great Britain and received a B.Eng. (honors) in electronic engineering from Liverpool University.

John F. Brown After receiving an M.S.E.E. from Cornell University in 1980, John Brown joined Digital's engineering staff. At present, he is a principal engineer and design team manager for the instruction decode section of the next generation CMOS-based VAX microprocessor. John's previous responsibilities include technical contributions to both the VAX 6000 Model 400 and the Model 200 chip sets. He was also the hardware engineer for the extended floating point data type enhancement to the VAX-11/780 system. John currently holds one patent, has two patent applications pending, and has authored technical papers for several publications.

Richard E. Calcagni A member of the Semiconductor Engineering Group's VLSI microprocessor verification group, Richard Calcagni has contributed to the design of several VLSI microprocessors in the areas of microcode, verification, and prototype system debugging. He has published several papers on CPU design, modeling, and verification. Prior to his work with microprocessors, he worked on module test process development for Digital's Customer Services Manufacturing organization. Before joining Digital in 1979, Rick worked for Burroughs Corporation. He received a B.S. (1976) in electrical engineering from the University of Rhode Island.

Michael A. Callander, Sr. Michael Callander is a principal engineer in Digital's Semiconductor Engineering Group. At present, he is responsible for the architecture of a future VAX CPU module. Mike led the VAX 6000 Model 400 system's REXMI chip set design project. His previous experience with Digital includes the design and verification of the CPU module for the VAX 8200 system and the VAX 8300 system. Mike received his B.S.E.E. degree from the University of Massachusetts in 1982. He joined Digital upon graduation.

Larry T. Camilli Since joining Digital in 1979, Larry Camilli has been a member of several product development projects. He is currently a software engineering supervisor and project leader for the architecture verification software development project, which focuses on the development and maintenance of test software for current and future architectures. Larry's previous responsibilities include the development of components for a microcode compiler, and a software trace and analysis package. He holds a B.S.E.E. from Clarkson University and is a student in the M.S.C.S. degree program at Worcester Polytechnical Institute.

John Croll John Croll is a principal software engineer in the Midrange Systems Engineering Group. In this position, he is responsible for developing systems test tools for a future hardware product. John was a team leader in the Systems Integration Group for the VAX 6000 Model 400 system. He was also a project leader for the development of the VAX 6000 Model 400 console software. John joined Digital in 1978, and his previous responsibilities include the development of device drivers and other system software. He received a B.S.E.E. (1978) from Drexel University and is a member of IEEE and ACM.

W. Hugh Durdan A graduate of the Rensselaer Polytechnical Institute, Hugh Durdan is a consulting engineer in the Semiconductor Engineering Group. In this position, his major responsibility is management of chip development. Hugh became involved in the VAX 6000 Model 400's chip design at its earliest beginnings in 1984. He led the behavioral modeling effort and the cache controller chip design project. He also managed the development of the custom chip set for the VAXvector processor and architected and specified the vector interface bus. Hugh joined Digital in 1980 and worked on the chip design of the VAX 8200 processor.

David M. Fenwick The architect of the VAXvector 6000 Model 400 processor, David Fenwick joined Digital's United Kingdom office in 1980 and transferred to the United States in 1985. He is a principal engineer for the Low End Midrange Systems Group. His previous experience with Digital includes field service regional support in the United Kingdom and project leadership for the DMB32 communications controller for the European engineering office. He also worked on the VAXBI and XMI programs in the United States. Dave received a B.Sc. (Honors) from Loughborough University of Technology in England.

Thomas C. Furlong The development of Digital's RISC-based workstations is the responsibility of Engineering Manager Thomas Furlong. It was Tom's Palo Alto-based design group that brought the MIPS RISC technology into Digital and developed the DECstation 3100 workstation. Tom has been with Digital for ten years. In addition to his work with RISC-based workstations, he was a member of the start-up team for the VAXstation group. Tom is originally from Detroit, Michigan, and earned a B.S.E.E. from Michigan State University. He holds two patents related to graphics workstation design and has three patents pending on the next-generation products.

Robert J. Hannemann Robert Hannemann is group manager of the SCIT Design and Engineering Physical Technology Group. His group is responsible for the delivery of IC packaging, module, physical design and test technology for Digital's microprocessor-based systems. A senior consultant engineer, Rob joined Digital in 1978. His prior experience includes positions at Bell Telephone Laboratories and the University of Maryland, where he was a member of the faculty. Rob holds the Sc.D. degree in mechanical engineering from MIT. He also holds two patents and has published several papers on heat transfer engineering and electronics packaging.

Stephen P. Hansen Senior Manager Stephen Hansen manages the SDE/PTG Technical Office. In this position, he coordinates programs related to cost-effective packaging for future CMOS-based products, defines technical solutions for the next generation of semiconductor packaging, and provides technical coordination for external packaging and interconnect-related activities. In his twelve years with Digital, Steve has developed several packages and assembly processes for internally manufactured CMOS products, including the tape-automated bonding process. He holds two patents in the areas of tape-automated bonding and packaging.

William V. Herrick Senior Consultant Engineer William Herrick is currently managing a new generation VAX chip design project in the Semiconductor Engineering Group. He joined Digital in 1977 and has been a member of many ZMOS- and CMOS-based product development projects, including the PDP-11/23 system and the VAX 8200 system. Before coming to Digital, Bill worked for Raytheon and GTE Sylvania. He has coauthored several papers on solid-state physics and MOS chip design. Bill received a B.S.E.E. (1969, magna cum laude) from Tufts University, and an S.M.E.E. (1971) and E.E. (1971) from MIT and is a member of Tau Beta Pi, Eta Kappa Nu, and Sigma Xi.

James R. Lundberg James Lundberg joined Digital in 1985. Initially, he was a product engineer with the MOS Product Engineering Group and worked on many projects, including BIIC and CQBIC. Jim is currently a senior engineer in Digital's Semiconductor Engineering Group. He was responsible for the signal integrity on the VAX 6000 Model 400 system and is now working on the signal integrity for an advanced CMOS CPU chip set and module. Before coming to Digital, Jim operated his own business. He is a member of Tau Beta Pi and Phi Kappa Phi. He received a B.S.E.E. (1985, honors) from the University of Illinois.

Richard C. Marcello A contributor to the design of the VAX 6000 Model 400 system, Richard Marcello is currently an engineering manager for a new-generation VAX chip design project. Rich was involved with reliability analysis of semiconductor devices before moving into design five years ago. He worked for Fairchild Semiconductor before joining Digital in 1981. Rich coauthored the paper "System, Process and Design Implications of a Reduced Supply Voltage Microprocessor," which he recently presented at the ISSCC. He received B.B.A. and B.S.E.E. (1980) degrees from the University of Notre Dame and a M.S.C.S. (1985) degree from Boston University.

Robert J. McCarty Since joining Digital in 1982, Robert McCarty has been involved in several major product development efforts. He managed the VAX 6000 Model 400 system's physical design project and M31 console and instrumentation module development, and led the PDP-11/84 system engineering project. Before coming to Digital, Bob worked for AM International as the project leader for a laser-based document printer that was part of a document communication system. He holds a B.S. in electrical engineering from the University of Michigan and an M.B.A. in marketing from the University of Chicago. He is a member of Tau Beta Pi and Eta Kappa Nu.

Michael J. K. Nielsen A consultant engineer in the Workstation Systems Engineering Group, Michael Nielsen is presently completing his responsibilities as architect and chief designer of the DECstation 5000 Model 200 workstation base platform. Mike joined Digital in 1984. Among the many projects he has worked on since that time are the DECstation 3100 workstation, for which he was the architect and chief designer, and the VAXstation 3520/3540, for which he was a member of the architecture and design team. Mike holds B.S.E.E., M.S.E.E., and Ph.D.E.E. degrees from Stanford University. He is a member of Tau Beta Pi and Phi Beta Kappa.

Sridhar Samudrala Currently acting as project leader for a floating point unit, Sridhar Samudrala is a principal hardware engineer in the Semiconductor Engineering Group. Sri joined Digital in 1977. Since that time, he has worked on testing and diagnostics, as well as the VAX 8200 system microcode, and floating point architecture and design. He holds two patents for his work in floating point design. Sri has an M.Sc. (Technology) from Andhra University in India and an M.S.E.E. from the University of Wisconsin.

D. John Shakshober John Shakshober is a senior hardware engineer in the Low End Midrange Systems Group. Previously involved in the hardware design of M31, a VAX parallel processor, John is now a member of the VAX 6000 Model 400 hardware group, where his particular focus is vector processor design. He joined Digital in 1984 after receiving a B.S. in computer engineering from the Rochester Institute of Technology. John received a M.S.E.E. from Cornell University in 1988. He is a member of IEEE and Tau Beta Pi. John's latest published technical paper, "Parallel Algorithms for Super Performance," was presented at SuperComputing 89.

Will Sherwood As a software consulting engineer, Will Sherwood manages the Semiconductor Engineering Group's VLSI microprocessor verification group. Previously, he managed the DECSIM logic simulation group. Will joined Digital in 1975 after receiving B.S.E.E. and M.S.E.E. degrees from Carnegie-Mellon University. In addition to his contribution to the *Digital Technical Journal,* he has published 15 technical papers and is a contributing author to three books. Will is a member of the IFIPS 10.2 working group on computer hardware description languages and has served as publicity chairman and program committee member for several international conferences.

Debra L. Slater In her position as principal software engineer, Debra Slater leads a group that provides performance modeling and analysis support to hardware development teams. She and her group were integral members of the VAXvector 6000 Model 400 vector processor design team. Prior to joining Digital in 1987, Debra worked for the Montreal Engineering Company, initially as a programmer/analyst and later as an independent consultant. She received a B.Sc. in mathematics and computer science in 1980 from Bishop's University in Quebec. In 1981, Debra received a master's degree in applied mathematics from the University of Waterloo in Ontario.

Rebecca L. Stamm Rebecca Stamm is a principal hardware engineer in the Semiconductor Engineering Group. She is currently leading the design of the backup cache, bus interface, and pin bus for a new VAX CPU chip. Rebecca was the architect of the backup cache controller chip for the VAX 6000 Model 400 system. She has also worked on design and verification of a RISC microprocessor. Rebecca joined Digital in 1983. She is a member of Eta Kappa Nu and IEEE, holds one patent, and has coauthored several technical papers for the ISSCC. She received a B.A. (1979) in history from Swarthmore College and a B.S.E.E. (1983) from MIT.

Patrick Sullivan The project leader for the VAX 6000 Model 400 CPU module development, Patrick Sullivan is a hardware consultant engineer in Digital's Semiconductor Engineering Group. Pat is now managing a new CPU module development project. Before his work on the VAX 6000 Model 400, he led the group effort that brought MCA emitter-coupled logic (ECL) into the corporation. He is also responsible for the development of a number of Digital's main memory products and participated in the development of several 36-bit CPUs. Pat holds a patent for a memory controller interface. He received his B.S. and M.S. degrees from Northeastern University.

John C. Sweeney Currently a principal engineer, John Sweeney is working on the testability and test process development for a future Digital product. His previous experience includes being part of the test process development for the VAX 8600 and VAX 8800 systems. John was an application engineer for GENRAD before joining Digital in 1981. He has authored several papers on boundary scan and fault isolation, and has one patent pending in relation to the VAX 6000 Model 400 system's test structures. John received a B.S.E.E. (1980) from Rensselaer Polytechnical Institute and has taken graduate courses at Northeastern University.

G. Michael Uhler G. Michael Uhler is a consulting engineer in the Semiconductor Engineering Group, where he is currently leading the architectural definition for a new CPU. As the CPU architect for the VAX 6000 Model 400 system, Mike was responsible for the CPU architecture, performance evaluation, and CPU microcode. Since joining Digital in 1978, he has also worked on the development of symmetric multiprocessing in the TOPS-10 operating system, and on the microcode and hardware development for PDP-10 CPUs. Mike received a B.S.E.E. (1975) and M.S.C.S. (1977) from the University of Arizona and is a member of IEEE, ACM, Tau Beta Pi, and Phi Kappa Phi.

Anthony J. Vaccaro A senior engineer in the Midrange Systems Evaluation Group, Anthony Vaccaro is at present responsible for several evaluation projects. These projects include the FV64A VAX 6000 Model 400 vector processor and KDM70 mass storage controller. Tony joined Digital in 1976 as a field engineer. Some of his earlier project responsibilities include product evaluations for the CIBCA-B VAXcluster CI adapter and KA825 VAX processor. He was also a member of the team that developed a certification process for new VAXBI adapters. Tony holds a B.S. (1975, cum laude) from Suffolk University and is studying for an M.S.C.S. at Rivier College.

Nicholas Wade The implementation of the backup cache control for the next-generation VAX CPU chip is being led by Nicholas Wade. Nick is a senior engineer in the Semiconductor Engineering Group. He joined Digital in 1986 and has worked on several projects, including the VAX 6000 Model 400 chip set. He performed the engineering evaluation and debugging for the system support chip on the VAX 3500 and VAX 6000 Model 200 systems. Nick was also a member of the behavioral design and implementation feasibility project for a CPU-XMI interface. He holds B.S. (1985) and M.S. (1986) degrees from Cornell University and is a member of IEEE.

Larry B. Weber As vice president of software development for MIPS Computer Systems, Inc., Larry Weber is responsible for the development, quality assurance, and integration of all systems software products. Larry is one of MIPS first employees, having joined the company in 1984. Prior to joining MIPS, he worked for Dialogic Systems and for IBM. Larry helped develop a PASCAL compiler for both the IBM mainframe and IBM RISC project. He has authored and coauthored a number of articles on compilers and languages. Larry holds a B.S. in mathematics from the State University of New York and an M.S. in computer science from the University of Colorado.

Neil C. Wilhelm Neil Wilhelm, a senior consultant engineer, is responsible for the development and maintenance of the Workstation Systems Engineering Group's CAD system and the design engineering of a low-cost workstation. Neil also designed Digital's first RISC-based system. Neil brought an extensive technical background to Digital when he joined the company in 1982. He has worked for Hewlett-Packard and Xerox Corporation, founded Ridge Computers, and taught at the University of Rochester. Neil holds a B.S. (1970) in engineering from Harvey Mudd College, and an M.S. (1971) and a Ph.D. (1973) in electrical engineering from Stanford University.

Douglas D. Williams MIT graduate Douglas Williams is a principal engineer in the Midrange Systems Engineering Group. He worked on the architectural definition of the VAXvector 6000 Model 400 processor. He also supervised performance modeling and vector control chip development efforts for that processor. Among the many other projects on which Doug has worked since joining Digital in 1981 are the RISC processor development, memory interconnect design, and VLSI design. He holds an S.B. and S.M. in electrical engineering and is a member of Eta Kappa Nu and Tau Beta Pi. Doug holds a number of patents and has several patent applications pending.

Foreword

Pauline A. Nist
Group Engineering Manager
Midrange Systems Business

Because microprocessor-based computer systems are complex, the work to design and architect custom chips must be initiated long before module and systems work begins.

Looking back at recent history, Digital introduced the VAX 6000 family of computers in April 1988 with the Model 200 series, which utilized the first generation of the CMOS-based VAX microprocessor. The Model 200 was fabricated in Digital's CMOS-1 (complementary metal oxide semiconductor) process. A single-processor Model 210 provided 2.8 times the performance of a VAX-11/780 system. One to four processor configurations provided up to a total of 11 times the performance of a VAX-11/780 system. (Papers discuss these chips and systems in the August 1988 issue of this journal.)

In January 1989, Digital introduced the second generation of the VAX 6000 family, the Model 300 series. The Model 300 increased single-processor performance from 2.8 to 3.8 times the VAX-11/780 system and total performance for a six-processor system to 22 times the VAX-11/780 system. The 30 percent increase in single-processor performance was made possible by a direct shrink of the die from Digital's 2.0 micron CMOS-1 process to Digital's 1.5 micron CMOS-2 process. The new process supported a 25 percent reduction in lateral and key vertical dimensions and a 78 percent improvement in circuit density. Together, these changes improved chip performance by approximately 30 percent.

However, a simple shrink of the existing die did not permit full exploitation of the new circuit density. Newly architected and designed parts had to be tailored to take full advantage of the density and speed available with the CMOS-2 process.

The VAX 6000 Model 400 series was formally introduced in July 1989, a mere six months after the Model 300 was introduced. The Model 400 utilized this newly architected chip set to provide single-processor performance of 7 times that of the VAX-11/780 system and up to 36 times the VAX-11/780 system for six-processor systems. The performance of the Model 400 was over twice the single-processor performance and more than three times the multiprocessor performance of the Model 200 series, which had been announced only a short 15 months earlier.

To support such aggressive product introduction cycles, advanced development work on the new generation of CMOS-2 chips began in mid-1984, within months of the start of the CMOS-1 designs. Actual design work began approximately a year later. During this period, Digital made a major decision to formally extend the VAX architecture to incorporate full support for vector processing into the base instruction set for all future VAX processors. To provide this support, the chip designs already under way had to be modified to incorporate the new instructions.

The scalar chip set developed consists of five custom VLSI parts. They are the CPU chip, the floating point accelerator chip, the vector/cache controller chip, the system support chip, and the clock chip. The development of the custom chips required a team of over 40 people, including logic, circuit, and layout designers, and verification engineers. The scalar CPU module design, standard cell interface design, and associated verification team comprised an additional eight engineers. Additionally, the new vector coprocessor module required three new custom parts, a new gate array, and a separate module design effort.

Since the direct shrink of die from the CMOS-1 to CMOS-2 process would account for only a 30 percent performance increase, the processor architecture had to be substantially changed to achieve more aggressive performance. Early in the project the chip design team established a clear goal to meet or exceed the performance of the VAX 8700 processor, which has a performance of five times that of the VAX-11/780 system. Some of the architectural changes included the following:

- A more pipelined architecture, specifically, a six-level pipelined engine built around three autonomous pipes

- A 64-bit wide data bus with 27 separate address lines versus a 32-bit multiplexed data/address bus used for the CMOS-1 chip

- Support for decode of the new VAX vector instructions and transfer of instruction operand information to the vector interface bus and onto the vector coprocessor module
- A 2 kilobyte (KB) primary on-chip cache with single-cycle access supported by a 128KB off-chip secondary cache
- A 16-byte instruction prefetch queue
- Two quadword write buffers in the bus interface unit

Experience with the CMOS-I chip showed that the fabrication line was capable of producing a distribution of die across a performance range of 80 to 100 nanoseconds (ns). As a result, whereas all CMOS-2 new designs supported the target of a 40 ns CPU cycle time, it was an explicit goal to support devices as fast as 28 ns, should chip yields produce sufficient quantities of faster parts. The yields at 28 ns actually exceeded predictions and permitted faster parts to be used in all products produced.

First passes of all scalar CPU chips were available in April 1988. These chips could successfully boot both the VMS and ULTRIX operating systems. This success was due in large part to the aggressive use of computer-aided design (CAD) techniques. Functional design verification efforts alone represented 25 person-years of work on the scalar chip set. An additional 39 person-years were necessary to complete the scalar CPU module and the vector coprocessor verification efforts.

The power-on of first-pass parts represented a significant accomplishment to those who worked on the chips and the module. However, much "behind the scenes" work was necessary to achieve this milestone. The success in this area represents the culmination of work across a number of disciplines. Besides the semiconductor devices, a new 224-lead multilayer ceramic package was developed. New techniques, including tape-automated bonding (TAB), were explored to attach the die to the package, and new specifications were necessary for the actual printed wire board material and board layup. Finally, new manufacturing processes were necessary to permit surface-mount assembly and test of these devices on both sides of the module.

Although initializing the operating system on the first CPU modules marks a key deliverable for the chip and board designers, it is only a starting point for the systems activity that is necessary to fully test and qualify a new product prior to the start of high-volume manufacturing. It is often difficult to understand the time lag between the availability of the first prototype unit running the operating system and a product that can be shipped to the customer. However, a substantial amount of work must be done between these two events. As the formal qualification process for new semiconductor devices begins, a parallel effort is undertaken to build a large number of early systems. These systems are used to identify any problems that may occur when the pieces of the system are assembled into configurations typical of those used by actual customers. Testing is divided across several aspects, including actual beta test of prototype units at customer sites, formal testing by any required government agencies (e.g., FCC, UL, VDE), systems design verification tests, and architectural testing that ensures that the new system complies with the formal VAX architecture standards. Once the majority of testing is in process and the required interim milestones have been met, manufacturing begins turning the assembled inventory into finished products to support volume availability of the system.

When a series of systems such as the VAX 6000 family has established a history in the market, it becomes increasingly important to ensure that the announcement of the latest family member coincides with manufacturing's ability to quickly deliver a high volume of product on a worldwide basis. If manufacturing cannot do so, a demand will have been created that cannot be filled. Revenue and sales are lost. The full payback from the many person-years of design, simulation, design verification, and systems test is only finally realized when volume manufacturing has begun.

The papers in this issue of the journal will provide insight not only into how microprocessors and systems are designed and architected, but also into the multidisciplinary efforts necessary to bring a successful product to market. The design of one of the first VAX vector coprocessors is also reviewed. This review offers a summary of how new architectural issues are resolved and how design trade-offs are made.

Moving a product from advanced development to engineering, through manufacturing, and into the customer site, over a five-year period, requires the efforts of many people around the world. Although only the direct work of a small percentage of those people are represented in these papers, the credit for the success of the products goes equally to all members of the team.

Debra L. Slater
David M. Fenwick
D. John Shakshober
Douglas D. Williams

Vector Processing on the VAXvector 6000 Model 400

The VAXvector 6000 Model 400 processor extends the VAX 6000 family of midrange CMOS-based multiprocessors to address the computing needs of numerically intensive applications. The three function units of the vector processor combine to form an overall vector pipeline that operates at speeds of up to 90 MFLOPs for single-precision calculations and 45 MFLOPs for double-precision calculations. The processor's performance can also be enhanced by taking advantage of overlapping and out-of-order instruction execution, as well as chaining. Further, applications can be tuned to the VAXvector 6000 hardware through algorithm optimizations in areas such as equation solvers and signal processing routines to achieve optimal performance. Using the VAXvector 6000 Model 400 system, performance increases ranging from 3 to 35 times that of the VAX 6000 Model 400 scalar system have been realized.

Vector processing has significantly evolved over the past two decades. In the late 1960s and early 1970s, it was pioneered as a way to increase scientific application computer performance over that achieved by more traditional scalar computers. However, the technology was limited to an elite few who could afford multimillion-dollar supercomputer systems and who were willing to significantly re-engineer software applications.

In the early 1980s, more sophisticated vectorizing compiler technology was developed. This technology allowed users to effectively program in high-level languages, such as FORTRAN, rather than to manually vectorize using low-level assembly language. During this period, there were also significant developments in computer algorithms that were better matched to the parallelism available in vector hardware.

Over the past few years, a new breed of vector processor, the mini-supercomputer, has emerged. This class of machine includes many of the performance features of traditional supercomputers, but at costs more commonly associated with superminicomputers. Because vector processing is now a mainstream style of computing that is applicable to a wide range of uses, the VAX architecture was recently extended to include vector operations. Further, the VAX product line has been expanded to include vector processing in both the VAX 6000 midrange systems family and the VAX 9000 mainframe family of systems.[1]

VAX Vector Processing Overview

The extension of the VAX architecture to include vector processing features was done in a manner that permitted a wide range of possible implementations.[2] The extension also allowed existing VAX processors to execute code utilizing the new vector instructions under software emulation.

The vector extensions to the VAX architecture include:

- The addition of 16 vector registers, each containing 64 64-bit elements
- A set of load/store instructions used to move up to 64 elements of a vector register to and from memory
- A set of vector register-to-register arithmetic and logical instructions, operating on up to 64 elements at a time
- A set of instructions for synchronization between scalar and vector processing subsystems

Conceptually, the implementation of vector instructions within the VAX family of processors is similar to that of floating point instructions. To implement floating point arithmetic, some systems use dedicated floating point hardware, some systems use microcode, and others emulate floating point in macrocode. In vector processing, vector instructions differ from floating point instructions in that they are designed to be executed in a semi-autonomous manner with scalar instructions. Thus, vector instructions can be executed in parallel with scalar instructions or in parallel with other vector instructions. Although the scalar and vector units operate somewhat independently, the units are closely coupled to ensure that memory management exceptions are precisely reported. Special operations ensure floating point exceptions and coherence between vector and scalar memory references are synchronized.

From a vector perspective, a typical VAX vector implementation can be reduced to five basic units. The latter four units are collectively referred to as a vector processor or vector unit. The basic units are:

- A scalar processor that executes scalar instructions, decodes vector instructions, which may contain multiple internal function units
- A vector instruction-sequencing control and register scoreboard
- A vector register file
- An arithmetic pipeline or pipelines that consist of one or more arithmetic/logic units
- A load/store unit for memory references

The vector control and scoreboard logic accepts instructions and operands from the scalar processor and dispatches them to the individual function units within the processor. It also reports exceptions and interrupts to the scalar processor. Since multiple vector instructions can be executed in parallel, the unit may contain scoreboard logic to identify and manage resource conflicts between instructions.

The vector register file contains the 16 vector registers, each of which consists of 64 64-bit elements. The register file has multiple ports that permit loads or stores to operate while operands are sent to the arithmetic pipes and results are received.

The vector arithmetic/logic pipelines implement all the integer, logical, and floating point instructions. These pipelines may be composed of separate pipelined add, multiply, and logical units. Or, they may be composed of multiple pipes that operate in parallel, with each pipe consisting of a pipelined add/multiply/logic unit.

The load/store unit is responsible for memory references. It generates the required virtual addresses (VA), performs translation from virtual to physical addresses, and loads or stores the data to or from the register files to memory. The load/store unit controls a cache memory and contains a virtual-to-physical address translation buffer (TB). Depending on the design of the scalar and vector units, there are two ways to implement the design memory interface for the scalar and vector processors:

- A combined scalar and vector processor that shares a common cache, address translation logic, and path to memory subsystem
- Separate scalar and vector units with separate caches and address translation buffers

Both of these approaches have their relative merits and disadvantages. When significant data-sharing between scalar and vector units exists, the combined approach provides more favorable cache performance because the common cache is updated on both scalar and vector references. Separate caches may result in additional cache misses as data is "sloshed" between scalar and vector caches. For limited data-sharing instances, the separate approach may offer more favorable cache performance. In a combined cache, vector references can displace needed scalar data and vice versa. This problem does not arise with separate caches because the scalar and vector data each has a dedicated cache. The separate cache approach also allows scalar and vector cache operations to occur in parallel.

In implementing a vector processor, the selection between the above alternatives is often driven more by technology constraints than issues of architectural elegance. The VAX 9000 system, which is implemented in emitter-coupled logic (ECL), chose the combined approach. This approach supported sharing costly cache RAMS and a common path to memory. The VAXvector 6000, which is implemented in complementary metal oxide semiconductor (CMOS) technology, chose the separate approach for two reasons. First, module space and package pin count constraints made it difficult to implement both scalar and vector functions on a single module. Second, the cost penalties for separate scalar and vector cache RAMS and separate paths to memory were not prohibitive.

VAX 6000 Vector Processor Description

System Block Diagram

The system block diagram for a vector-capable VAX 6000 Model 400 machine is shown in Figure 1.

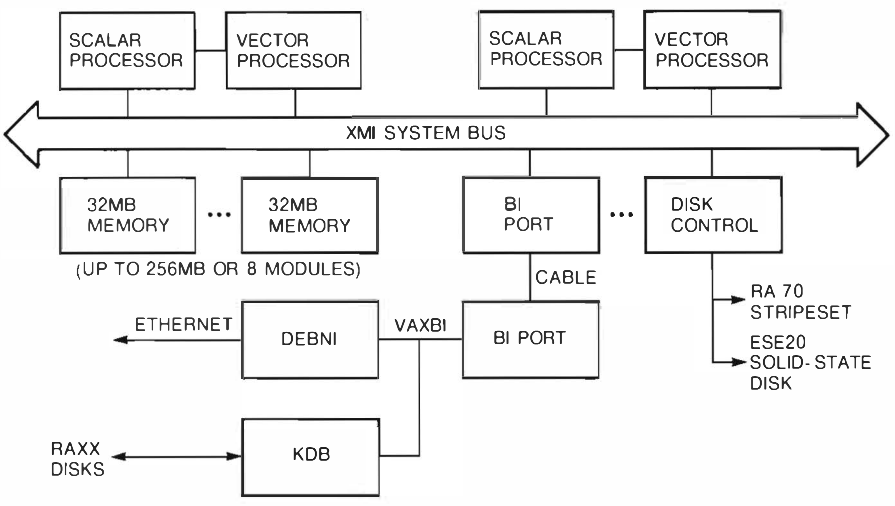


Figure 1 VAXvector 6000 Model 400 System Block Diagram with Dual Scalar and Vector Processors

The vector processor occupies a slot adjacent to the scalar processor, and both are interconnected by a short interface cable. The vector processor receives all instructions from and returns status to the scalar processor across this cable. For memory references, the vector processor has its own independent path to main memory. The VAX 6000 Model 400 system supports configurations of up to six scalar processors. However, vector systems have additional configuration constraints because of the increased memory bandwidth and XMI slot requirements of the scalar/vector processor. The VAXvector 6000 Model 400 system supports configurations of single or dual scalar/vector processors, or configurations of one scalar/vector processor and up to three additional scalar processors. To satisfy memory bandwidth requirements, VAXvector 6000 systems with a single scalar/vector processor require at least two memory controllers. Dual scalar/vector systems or single scalar/vector systems with additional scalar processors require at least four memory controllers.

VAX 6000 Model 400 Vector Processor

The block diagram for the vector processor is shown in Figure 2. The machine is divided into three separate function units that can operate in combination or independently:

- A vector controller, implemented as a single chip
- Arithmetic pipelines implemented by four pairs of chips, i.e., register file and vector data path
- A load/store unit, implemented by one chip, which also controls a 1 megabyte (MB) cache

Vector Control Chip

When the scalar processor encounters a vector opcode, it parses and fetches the operands. The opcode and all its operands are dispatched through the instruction bus to the vector processor. For arithmetic instructions, the scalar processor will proceed to decode the next opcode in the instruction stream. However, for load and store instructions, the scalar processor is stalled until all address translations are completed. Stalling guarantees that any memory management violations are synchronous and that the scalar processor can restart the faulting instruction correctly. Within the vector unit the vector control chip is responsible for all scalar vector communication. When instructions are received by the vector controller chip, the vector controller buffers the instructions and controls instructions issuing to the other function units within the vector processor.

An important aspect of the vector control chip is the register scoreboard logic, which identifies potential register conflicts when vector instructions are executed in parallel. By maintaining accurate register usage data, the vector control chip can optimize parallelism with the vector processor. Optimal performance is achieved by executing arithmetic operations in parallel with load and store operations, chaining the results of arithmetic opera-

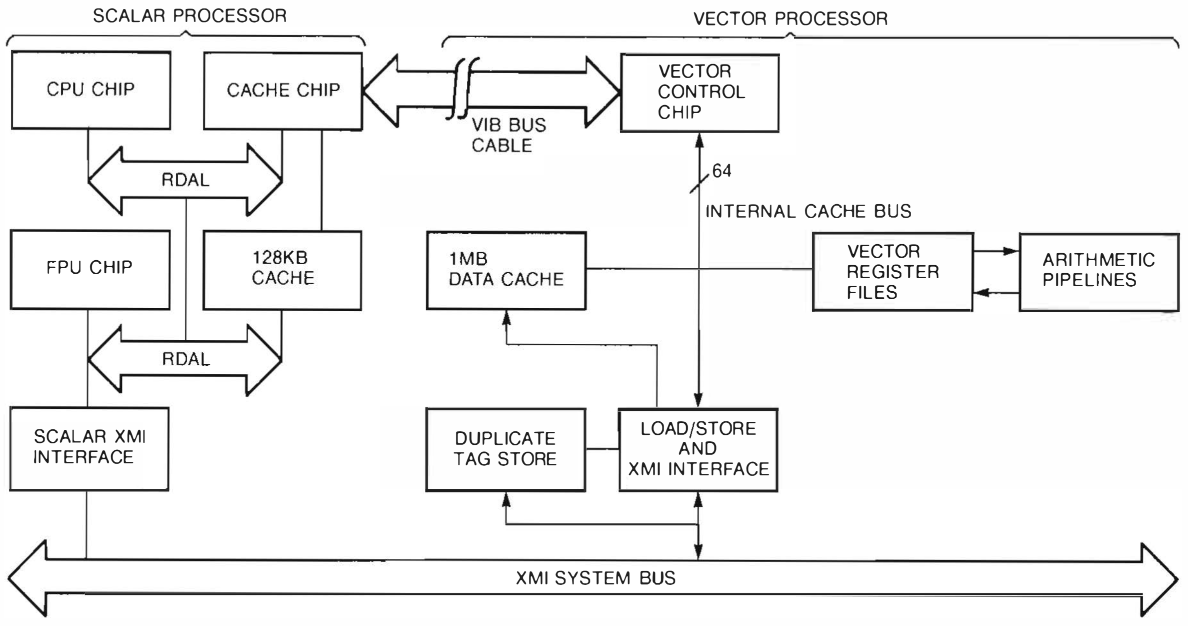


Figure 2 VAXvector 6000 Model 400 Scalar/Vector Processor Block Diagram

tions into store operations, and even dynamically re-ordering the execution of arithmetic instructions relative to load and store instructions to improve parallelism.

The vector control chip sends all error status and machine checks to the scalar processor. When an error is encountered, the control chip attempts to retry the failing transaction. If the retry is successful, a soft error interrupt is sent to the scalar processor. If the retry fails, either a hard error interrupt or a machine check is sent to the scalar processor. Read operations that fail result in machine checks. Write operations that fail result in hard error interrupts.

Although not part of the overall control function, the vector control chip also contains logic to implement the IOTA instruction. The IOTA instruction builds a set of offsets in a vector register. This function did not fit conveniently in any other vector function. The control chip was selected because it had the space available to contain the function.

Vector Register File and Arithmetic Pipeline

The VAXvector 6000 processor's arithmetic pipeline is organized as four pipes. Each pipe consists of a quarter of the register file (every fourth element of the vector registers), and an associated arithmetic/logic unit. Each individual pipeline can retire one single-precision calculation every cycle or one double-precision calculation every two cycles. The four pipes collectively retire four single-precision calculations every cycle, or two double-precision calculations every cycle. Thus, a much higher floating point performance is achieved than with only one individual pipeline.

The register file chips receive instructions from the vector controller and data from the cache or load/store unit. The register file chip provides read operands to the arithmetic pipeline and stores write results and mask information. To maximize the use of cache bus bandwidth, two 32-bit operands can be combined into a single 64-bit transfer that is simultaneously read or written to two separate register file chips. The register file internally has four 64-bit ports. (One is a read/write port for memory data; two are read ports for operands; and one is a write port for results. While one instruction is writing its results, a second can start reading its operands. Thus, the instruction pipeline delay is hidden. Variations in pipeline length between instructions are smoothly handled to ensure that no gaps exist in the flow of write data.

The register file can hold two outstanding arithmetic instructions in its internal queue. Therefore, the vector controller can preload the arithmetic instruction queue with a second instruction, i.e., deferred instruction. Preloading allows the vector controller to free the cache data bus, which is also used for instruction issuance, for use by a subsequent load or store instruction. This feature improves performance because the arithmetic pipeline can execute two arithmetic instructions in the time it takes to execute one load or store instruction.

The register file's operand and result ports are used by the vector arithmetic pipeline chip. Operand data is sent over a 32-bit bus that is driven twice per cycle. Results are returned on a separate 32-bit bus that is driven once per cycle. The two operands for single-precision instructions can be passed in one cycle, while double-precision operands take two cycles to transfer. Each arithmetic pipeline chip has a throughput of one single-precision operation per cycle, one double-precision operation per 2 cycles, and one single-precision or double-precision divide per 10 or 22 cycles. The arithmetic chip has a pipeline delay of six cycles for double-precision multiplications, and five for all others (except divides), including the data transfer cycles. Integer operations are recoded internally as double-precision floating point data types. The vector arithmetic pipeline chip is a full custom implementation largely based on the design of the scalar processor's floating point unit.[3]

Load/Store Unit

The control chip uses the vector processor's internal bus to issue instructions to the function units. However, once a load or store instruction is issued, the load/store chip becomes bus master and controls the internal bus. Either the load/store chip, vector register files, or the cache can drive the bus. Once a load or store instruction starts execution, no further instructions can be issued until it completes. This rule simplifies the control chip scoreboarding because once a load or store instruction is started, no further instructions can start. Therefore, scoreboarding of these instructions against the outstanding load or store instruction is not necessary. Because scoreboarding of outstanding instructions requires considerable logic complexity in the vector control chip, it was important to keep the complexity of this operation minimal. An additional benefit was the simplification of the internal bus protocol. It was excessively complex to implement the capability to stop load or store instructions in progress. This alternative was not pursued because the resulting performance benefit was minor in comparison to the amount of work involved.

The load/store chip executes the vector load, store, scatter and gather memory reference instructions. These instructions involve virtual-to-physical address translation, cache management, and interaction with the memory bus. If a load or store instruction requires an offset register, such as scatter or gather, the offset register is first read into a buffer and then added to the instruction's base address. This process eliminates turning around the internal bus for each offset read, which would add more overhead. For strided load or store instructions, the address is generated by adding the stride to the instruction's base address.

Load or store instructions can operate on either 32-bit (i.e., long word, single-precision) or 64-bit (i.e., quadword, double-precision) data types. When executing unity-stride 32-bit load or store instructions, the load/store chip operates on two elements at a time. Two 32-bit elements are combined into a single 64-bit cache reference. This combination significantly enhances performance enhancement in unity stride single-precision data operations.

Virtual-to-physical address translation is performed using an on-chip, 136-entry, 68-way-associative translation buffer (TB). This configuration maximizes address translation efficiency, which is very important because only limited chip space was available. To optimally service TB miss conditions, the load/store chip contains dedicated logic that directly references page table entries upon a TB miss. A simpler alternative would have been to use microcode in the scalar processor to fetch new page table entries upon a TB miss. However, the dedicated logic approach was chosen to enhance performance for applications that exceed the size of the TB. Under certain TB miss conditions, the vector processor may be unable to compute a new virtual-to-physical address translation. This situation can occur when the addressed page is invalid or has been paged-out to disk. When such a miss occurs, the vector unit reports an exception back to the scalar processor. Once the scalar processor corrects the situation, the instruction is retried from the beginning.

Since the scalar processor must be able to restart the faulting vector instruction, it is important to precisely identify any vector memory management exception with the associated vector load or store instruction. This identification is achieved by blocking issuance of further instructions until the vector unit notifies the scalar processor that the vector instruction is free of memory management faults. The vector unit contains memory management prediction logic, called MMOK logic. MMOK logic allows the scalar processor to issue additional instructions in parallel with the execution of the vector load or store instruction. During execution of a strided vector load or store instruction, once it is established that the current vector element references the same TB entry as the last element of the load or store instruction, and that the associated TB entry is free of memory-management exception conditions, the load/store unit can safely report "address translation successful," i.e., MMOK, to the scalar processor. Early prediction of successful address translation permits the scalar processor to be released and allows it to operate asynchronously with the remainder of the vector load or store instruction.

Once a physical address is obtained, the load/store chip references its 32K entry tag store. The address is delayed and passed to the 1 MB cache data store. This delay permits cache tag lookup and compare to complete before data is written to the cache on store operations. In parallel, the corresponding register file address is presented to the four register file chips. The data and addresses are automatically aligned for load and store operations to permit correct reading and writing of the register file and cache data RAMs. Upon cache miss, the load/store unit queues the associated 32-byte block read operation with the memory interface logic and continues processing other elements. Up to four cache misses can be outstanding before the read data for the first miss returns. Hits continue to be processed while the misses are outstanding. On vector processors, the most important factor is the time required to complete the entire load or store operation, rather than the time needed to fetch an individual element. The cache miss handling feature permits the vector processor to maximize its use of available XMI bandwidth.

The vector cache tag and data are parity-protected. Should a cache parity error occur, the cache is disabled and the instruction retried from the beginning. This method was the simplest option for soft recovery of cache parity errors. The operating system receives a soft error interrupt and can, at its option, re-enable the cache.

The load/store chip contains a 32-element write buffer to enhance performance of store operations. Since the vector cache operates at higher bandwidths than the system bus, the buffer isolates the store performance from the slower XMI memory bus. Furthermore, a subsequent load instruction that hits in cache can execute while the write buffer is being written to memory. If the load instruction takes a cache miss, the load stalls until the store completes. This simple scheme improves instruction overlap when load instructions follow store instructions without adding undue complexity to the load/store unit design.

Performance Characteristics

The interaction between the different functional units of the vector processor creates a number of situations that affect the performance and execution of vector instructions. These include:

- Overlapping instructions
- Out-of-order instructions
- Chaining

Overlap of Instructions

Arithmetic and load/store instruction execution may overlap because the functional units are independent. In order to achieve this overlap, the following conditions must be met.

- The arithmetic instruction must be issued before the load or store instruction.
- There must be no register conflict between the arithmetic and load/store instructions.

In the following examples of arithmetic and load/store instruction interactions, an "I" represents instruction issue time, and an "E" represents instruction execution time. The "VR" represent vector registers. The expression "std" is used to represent the stride. A series of periods "..." represents wait time in the arithmetic unit for deferred instructions. (Note: These examples are not intended as timing diagrams.)

```
VVADDx  VR1,VR2,VR3   IEEEEEEEE
VLDx    A,std,VR1      IEEEEEEEEEEEEEE
```

As can be seen in the example above, the execution of the vector load instruction (VLDx) can overlap the vector add instruction (VVADDx) because there are no register conflicts between the two instructions. In the next example, instruction overlap is inhibited because the VVADDx instruction is writing to the register to be loaded, VR3.

```
VVADDx  VR1,VR2,VR3   IEEEEEEEE
VLDx    A,std,VR3              IEEEEEEEEEEEEEE
```

In comparing these two examples, it is clear that the overlap of the execution of the VVADDx and the

VLDx greatly reduces the total execution time of the instruction sequence. By taking advantage of this hardware feature, application codes can show greatly improved performance.

Out-of-order Instruction Execution

The arithmetic unit includes a deferred instruction queue of length 1. This queue allows the vector control and scoreboard logic to queue one instruction to the arithmetic unit while that unit is still executing a previous instruction. The vector controller checks the queue's status for an instruction when it checks the function unit's availability. Both the deferred and currently executing instructions are checked for register availability. This queue frees the issue unit to process another instruction rather than waiting for the arithmetic unit to complete its current instruction.

For the following instruction sequence,

```
VVADDx VR1,VR2,VR3
VVMULx VR3,VR1,VR4
VLDx   A,std,VR2
```

execution without a deferred instruction queue would resemble this example:

```
Issue VVADDx   IEEEEEEEE
Issue VVMULx            IEEEEEEEE
Issue VLDx               IEEEEEEEEEEEEEE
```

Execution with a deferred instruction queue would look like the following:

```
Issue VVADDx            IEEEEEEEE
Issue deferred VVMULx    I.......EEEEEEEE
Issue VLDx                IEEEEEEEEEEEEEE
```

These examples illustrate the use of a deferred arithmetic instruction. If a deferred instruction queue was not implemented, the VVMULx instruction could not be issued until the VVADDx was either completed or nearly completed. The VLDx instruction would not issue until after the VVMULx was issued and would complete much later than in the deferred instruction case. Once the VLDx instruction is issued, no other instructions may be issued. The instruction overlap execution made possible by the deferred instruction queue significantly reduces total execution time.

Comparing the last two examples, in the case where a deferred instruction queue was used, the VLDx instruction can begin executing before the VVMULx. It also could complete before the VVMULx instruction completes simply because the VVMULx instruction is sent to the deferred arithmetic instruction queue. This out-of-order execution of instructions allows increased overlap of instructions, which again reduces the total execution time of the instruction sequence.

Chaining

Vector operands are generally read from and written to the vector register file. An exception to this process occurs when a store instruction is waiting for the results of a currently executing arithmetic instruction. (Divide instructions are not included in this exception because they do not have the same degree of pipelining as the other instructions.) As results are generated by the arithmetic instruction and are ready to be written to the register file, they are also immediately available for input to the waiting store instruction. Therefore, the store instruction can begin processing the data before the arithmetic instruction has completed. This process is called "chain into store." The store instruction will not overrun the arithmetic instruction because the store instruction can process data faster than the arithmetic unit can generate results.

The following instruction sequence

```
VVADDx VR1,VR2,VR3
VVMULx VR1,VR2,VR4
VSTx   VR3,A,std
```

would resemble the example in Figure 3 if executed without the chain into store process.

```
Issue   VVADDx              IEEEEEEEE
Issue   deferred VVMULx      I.......EEEEEEEE
Issue   VSTx                          IEEEEEEEEEEEEEE
```

Figure 3 Sample Instruction Sequence without Chain into Store

If the instruction sequence were executed with the chain into store process, however, it would follow this example:

```
Issue VVADDx            IEEEEEEEE
Issue deferred VVMULx   I.......EEEEEEEE
Issue VSTx                IEEEEEEEEEEEEEE
```

In these examples, the VSTx instruction requires the result of the VVADDx instruction. Without the chain into store operation, the VSTx instruction must wait for the VVADDx to complete before beginning. The chain into store operation allows the VSTx instruction to begin while the VVADDx is still executing, increasing the amount of instruction execution overlap. As a result, the instruction sequence requires a shorter period of time to complete execution.

Load and Store Unit Performance

Memory access instructions are typically slower than arithmetic instructions and will frequently dominate the performance of vectorized applications. Certain coding techniques help minimize the time spent waiting for load and store instructions to complete and reduce the resulting impact on performance.

Maximize Instruction Execution Overlap Three important hardware features help maximize instruction execution overlap in the load/store unit. First, a load or store instruction can execute in parallel with up to two arithmetic instructions, provided the arithmetic instructions are issued first. Second, the chain into store sequence can reduce the perceived execution time of a store instruction. Finally, early detection of no memory faults allows scalar-to-vector unit communications to overlap with load or store instruction execution.

In the instruction sequence shown in Figure 4, the main loop of a SAXPY or DAXPY BLAS 1 routine, there is very little instruction overlap.[4]

In the reordered instruction sequence shown in Figure 5, the VVMULx and second VLDx instructions overlap, and less total execution time is required than in the first example.

The only real difference between the instruction sequences in Figures 4 and 5 is the order in which they are issued. By recognizing that the VVMULx does not require the result of the second VLDx and can precede that instruction, a significant reduction in execution time is achieved.

The overlap of load and store instructions can also be effectively maximized by preceding, wherever possible, all load and store instructions by at least two arithmetic instructions. In this way, both the load and store pipeline and the arithmetic pipeline are in use.

Minimize Register Conflict Waits A load instruction cannot begin execution until the register to which it will write is free. A register conflict may occur if the destination register of a load instruction is the same as the register for a preceding arithmetic instruction. If using a different register for the load instruction would permit instruction execution overlap to occur, the destination register should, if possible, be changed.

Locality of Reference of Data The locality of reference of data is important in determining the performance of load and store operations. Because unity stride load and store instructions are the most efficient memory access instructions, whenever possible, data should be stored in the sequential order in which it is usually referenced.

Non-unity stride load and store operations can have a significantly higher impact than unity stride

```
VLDx    X,std,VR1     IEEEEEEEEE
VLDx    Y,std,VR2               IEEEEEEEEE
VVMULx  VR3,VR1,VR1                       IEEEEE
VVADDx  VR1,VR2,VR2                        I....EEEEE
VSTx    VR2,Y,std                               IEEEEEEEEE
```

Figure 4 Sample Instruction Sequence of Main Loop of a SAXPY or DAXPY BLAS 1 Routine

```
VLDx    X,std,VR1     IEEEEEEEEE
VVMULx  VR3,VR1,VR1              IEEEEE
VLDx    Y,VR2                     IEEEEEEEEE
VVADDx  VR1,VR2,VR2                         IEEEEE
VSTx    VR2,Y,std                            IEEEEEEEEE
```

Figure 5 Sample Instruction Overlap Sequence of a Main Loop of a SAXPY or DAXPY BLAS 1 Routine

operations on the performance level of the XMI memory bus. More memory references are required for non-unity stride operations of the same vector length. If the ratio of cache miss load and store instructions to arithmetic instructions is sufficiently high and non-unity stride is used, bus speed and bandwidth can limit performance.

Load and store operations that hit in cache are less costly than those that miss cache. Any piece of data must be loaded from memory to cache the first time it is referenced. If data that is referenced more than once remains in the cache, i.e., is not displaced by subsequent data accesses, other references to that data will not incur memory access costs, and better performance results.

On any cache fill, a 32-byte data block is read into the cache. (Note: This is equivalent to 8 long words or 4 quadwords. Single-precision data is a long word; double-precision data requires a quadword.) When non-unity stride loads are used, performance can be improved by using the additional data read in to cache. This improvement can be achieved by following cache miss non-unity stride loads with non-unity stride loads that reference the additional data and will, therefore, hit in cache.

Large data arrays and strides can also have an impact on the efficiency of the translation buffer. For large strides, i.e., greater than 256 single-precision elements or 128 double-precision elements, a translation buffer miss can occur for each vector element. Even with unity stride, the translation buffer miss rate will be higher for large data arrays.

Algorithms It is sometimes necessary to consider the algorithm that is represented by the code to be optimized because some algorithms are not as well suited to vector processing as others. It may be more effective to change the algorithm used or the way it is implemented than to optimize the existing code. Several examples that illustrate effective optimization methods are discussed in the Algorithm Optimization section of this paper.

Arithmetic Unit Performance

Once an instruction begins execution in the vector arithmetic unit, it continues executing until all results are completed. A deferred arithmetic instruction may only begin execution after the instruction in the pipeline completes, or if the first results of the deferred instruction will not complete before the last results from the current instruction are completed. The instruction overlap will be particularly significant for shorter vectors because the startup time, i.e., the time that can be overlapped, for an arithmetic instruction is fixed overhead that represents an increasing portion of the execution time as vector length decreases.

Peak Performance

The Model 400 vector processor has a cycle time of 44.44 nanoseconds. For single-precision operations, this cycle time translates to a theoretical peak performance of 90 MFLOPs. For double-precision operations, the theoretical peak performance is 45 MFLOPs. Theoretical peak performance is calculated from the number of results per cycle and the cycle time as follows:

$$\textit{Single-precision peak} = 4 \,/\, (44.44 \times 10^{3})$$

$$\textit{Double-precision peak} = 2 \,/\, (44.44 \times 10^{3})$$

Crossover Point

For any given instruction or sequence of instructions, there is a particular vector length where both the scalar and vector processing of equivalent operations yield the same performance. This vector length is the crossover point between scalar and vector processing and is unique to the particular instruction or sequence. Scalar operations are faster for vector lengths below the crossover point. Vector operations are more efficient for vector lengths above the crossover point. A low crossover point is considered a benefit because it indicates that it is easier to take advantage of the power of the vector processor. A low crossover point means that more code can benefit from the vector processor.

For any single, isolated vector instruction, the crossover point on the Model 400 system is quite low, generally about 3 or 4. However, an instruction is not performed in isolation. In a routine or application, other factors affect the performance of the operations on short vectors. These effects are particularly seen when the short vector's data is used in several vector operations.

On the Model 400 system, performing as much code as possible on the vector processor, including short vector length sections, can mean higher system performance. Performance is improved because the cache is used more optimally. Specifically, once vector instructions have referenced a piece of data, that data is included in the vector unit's cache. Subsequent scalar operations on that data will require moving the data from the vector cache into the scalar cache. Continued code sections of vector references followed by scalar references tend to invalidate the two caches too frequently. Therefore, a vector operation is usually more efficient than a scalar operation. The crossover point on the Model 400 system is low enough that scalar processing is the faster alternative only for isolated operations on short vectors.

Algorithm Optimization Examples

The previous section of this paper discussed how the characteristics of the VAXvector 6000 Model 400 system's vector processor can affect performance. The following examples illustrate how that performance information can be used to build optimized routines. The examples also show how an algorithm and its implementation can change the performance of an application on the VAXvector 6000 processor.

Algorithm changes can alter the data access patterns to use the memory subsystem more efficiently, can increase the average vector length, and can minimize the number of vector operations required. By applying Amdahl's Law of vectorization, we can improve performance by increasing the percentage of code that is vectorized.

To take advantage of the processing power of the VAXvector 6000 Model 400 system, we concentrated on four basic optimization methods:

- Rearrange code for maximum vectorization of the inner loop and remove data dependencies within the loop
- Vectorize across contiguous memory locations to produce unity stride vectors for increased cache hit rates and optimized cache miss handling
- Reuse the data already loaded into the vector registers as frequently as possible to reduce the number of vector load and store operations
- Maximize instruction execution overlap by pairing arithmetic instructions between load and store instructions wherever possible

(Note: Further information on optimization techniques in FORTRAN can be found in the *VAX FORTRAN Performance Guide* available with the FORTRAN-High Performance Option.[5] Additional information on macrocoding for the VAXvector 6000 Model 400 vector processor can be found in the *VAX 6000 Vector Processor Programmer's Guide.*[6])

By analyzing the groups of applications that have high vector processing potential, we identified two basic areas where optimization techniques can be most useful: equation solvers and signal processing routines. For example, computational fluid dynamics, finite element analysis, molecular dynamics, circuit simulation, quantum chromodynamics, and economic modeling applications use various types of simultaneous or differential equation solvers. Applications such as air pollution modeling, seismic analysis, weather forecasting, radar imaging, speech and image processing, and many other scientific and engineering applications use signal processing routines, such as fast Fourier transforms, to obtain solutions.

Equation Solvers

Equation solvers generally fall into four categories: general rectangle, symmetric, hermitian, and tridiagonal. The most common benchmark used to measure a computer system's ability to solve a general rectangular system of linear equations is Linpack.[4] The Linpack benchmarks, developed at Argonne National Laboratory, measure the performance across different computer systems while solving dense systems of 100, 300, and 1000 linear equations.

These benchmarks are currently written to call subroutines from the Linpack library. The subroutines, in turn, call the basic linear algebra subroutines (BLAS) at the lowest level. For each benchmark size, there are different optimization rules which govern the type of changes permitted in the Linpack report. Optimizations to the BLAS routines are always allowed. Modifications can be made to the FORTRAN source or by supplying the routine in macrocode. Algorithm changes are only allowed for the largest problem size, the solution to a system of 1000 linear equations.

The smallest problem size uses a two-dimensional array that is 100 by 100. The benchmarks are written to use Gaussian elimination for solving 100 simultaneous equations. This two-step method features a factorization routine, xGEFA, and a solver, xGESL. Both are column-oriented algorithms and use vector-vector level 1 BLAS routines. Column orientation increases program efficiency because it improves locality of data based on the way FORTRAN stores arrays.

As shown in Figure 6, the BLAS level 1 routines allow the user to schedule the instructions optimally in vector macrocode. Deficiencies in BLAS 1 routines include frequent synchronization, a large calling overhead, and more vector load and store operations in comparison to other vector arithmetic operations.

The performance of the Linpack 100 by 100 benchmark, which calls the Figure 3 routine, shows how an algorithm with approximately 80 percent vectorization can be limited by the scalar portion. One form of Amdahl's Law relates the percentage of vectorized code compared to the percentage of scalar code to define an overall vector speedup. This ratio between scalar runtime and vector runtime is described by the following formula:

$$\textit{Vector speedup} = \textit{time scalar} \,/\, (\%[\textit{scalar}] \times \textit{time scalar}) + ([\%\textit{vector}] \times \textit{time vector})$$

Under Amdahl's Law, the maximum vector speedup possible, assuming an infinitely fast vector processor, is:

$$\textit{Vector speedup} = 1.0 \,/\, (0.2) \times 1.0 + (0.8) \times 0 = 1.0 \,/\, 0.2 = 5.0$$

As shown in Figure 7, the Model 400 processor achieves a vector speedup of approximately 3 for the 100 by 100 Linpack benchmark when using the BLAS 1 subroutines. It follows Amdahl's Law closely because it is small enough to fit the vector processor's 1 Mbyte cache and, therefore, incurs very little overhead due to memory hierarchy.

For the Linpack 300 by 300 benchmark, optimizations include the use of routines that are equivalent to matrix-vector level 2 BLAS routines.

```
xAXPY - computes Y(I) = Y(I) + aX(I)
        where x = precision = F, D, G

        MSYNC                           ;synchronize with scalar
LOOP:
        VLDx     X(I),std,VR0           ;X(I) is loaded into VR0
        VSMULx   a,VR0,VR0              ;VR0 gets the product of VR0
                                        ;and the scalar value "a"
        VLDx     Y(I),std,VR1           ;Y(I) get loaded into VR1
        VVADDx   VR0,VR1,VR1            ;VR1 gets VR0 summed with VR1
        VSTx     VR1,Y(I),std           ;VR1 is stored back into Y(I)

        INC      I                      ;increment I by vector length
        IF (I < SIZ) GOTO LOOP          ;Loop for all values of I
        MSYNC                           ;synchronize with scalar
```

Figure 6 Core Loop of a BLAS 1 Routine Using Vector-Vector Operations

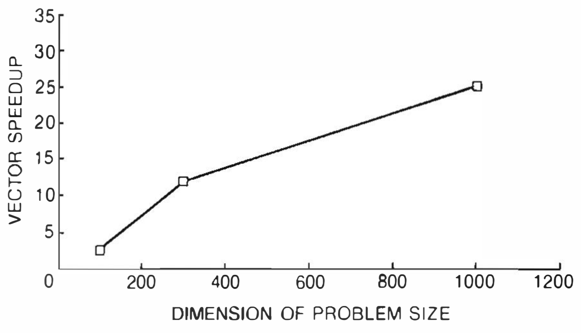


Figure 7 Linpack Performance Graph, Double-precision BLAS Algorithms

Figure 8 details the core loop of a BLAS 2 routine. BLAS 2 routines make better use of cache and translation buffers than the BLAS 1 routines do. Also, BLAS 2 routines have a better ratio between vector arithmetics and vector load and stores. The larger matrix size increases the average vector length. Performance is improved by amortizing the time to decode instructions across a larger work load.

By removing one vector load and one vector store from the innermost loop, the BLAS 2 routine has a better ratio of arithmetic operations to load and store operations than BLAS 1 routines. Although the 300 by 300 array fits into the vector processor's 1 MB cache, not all the cache can be mapped by its translation buffer. By changing the sequence in which this routine is called in the program, the data access patterns can be altered to better use the vector unit's translation buffer. Thus, higher performance is obtained.

The percent of vectorization increases primarily because of the increase in the matrix size from 100 by 100 to 300 by 300. With a vector fraction of approximately 95 percent, Figure 7 shows the speedup improvement in the 300 by 300 benchmark when using methods based on BLAS 2 routines. With a matrix vector algorithm, the 300 by 300 benchmark yields speedups of between 10 and 12 over its scalar counterpart.

There are no set rules to follow when solving the largest problem size, a set of 1000 simultaneous equations. One potential tool for optimizing this benchmark is the LAPACK library developed by Argonne National Laboratory, in conjunction with the University of Illinois Center for Supercomputing Research and Development (CSRD). The LAPACK library features equation solving algorithms that will block the data array into sections that fit into

```
xGEMV - computes Y(I) = Y(I) + X(J)*M(I,J)
        where x = precision = F, D, G

        MSYNC                        ;synchronize with scalar
ILOOP:
        VLDx     Y(I),std,VR0        ;Y(I) is loaded as VR0
JLOOP:
        VLDx     M(I,J),std,VR1      ;VR1 gets columns of M(I,J)
        VSMULx   X(J),VR1,VR2        ;VR2 gets the product of VR1
                                     ;and X(J) as a scalar
        VVADDx   VR0,VR2,VR0         ;VR0 gets VR0 summed with VR2
        INC      J
        IF (J < SIZ) GOTO JLOOP      ;Loop for all values of J

        VSTx     VR0,Y(I),std        ;VR0 gets stored into Y(I)
        INC      I
        IF (I < SIZ) GOTO ILOOP      ;Loop for all values of I
        MSYNC                        ;synchronize with scalar
```

Figure 8 Core Loop of a BLAS 2 Routine Using Matrix-Vector Operations

a given cache size. The LAPACK library calls not only the BLAS 1 and BLAS 2 routines but also a third level of BLAS, called matrix-matrix BLAS or the BLAS level 3.[7]

Figure 9 shows that a matrix-matrix multiply is at the heart of one BLAS 3 routine. The matrix multiplication computation can be blocked for modern architectures with cache memories. Highly efficient vectorized matrix multiplication routines have been written for the VAX vector architecture. For example, a double precision 64 by 64 matrix multiplication achieves over 85 percent of the peak MFLOPs on the Model 400 system.

Performance can be further improved with other methods that increase the reuse of data while it is contained in the vector registers. For example, loop unrolling can be done until all the vector registers have been fully utilized. Partial results can be formed within the innermost loop to minimize the loads and stores required. Because both rows and columns are traversed, the algorithm can be blocked for cache size. The VAXvector Model 400 system exhibits vector speedups greater than 35 for the 64 by 64 matrix multiplication described above.

Although the overall performance of the 1000 by 1000 size benchmark is less than a single 64 by 64 matrix multiplication, it does indicate the potential performance when blocking is used. Improving the performance of this benchmark is most challenging because the 1000 by 1000 matrix requires about eight times the vector cache size of 1MB. Further analysis is being conducted to determine the most efficient block size that would maximize the use of BLAS 3 and remain within the size of the cache for a given block of code.

The vectorized fraction increases to approximately 98 percent for the 1000 by 1000 benchmark. The proportion of vector arithmetics relative to vector loads and stores is much improved for the BLAS 3s. Although the cache is exceeded, performance more than doubles when using a method that can block data based on the BLAS 3 algorithms. Therefore, the VAXvector 6000 Model 400 processor's performance for Linpack 1000 by 1000

```
xGEMM - computes Y(I,J) = Y(I,J) + X(I,K)*M(K,J)
        where x = precision = F, D, G

        MSYNC                         ;synchronize with scalar
IJLOOP:
        VLDx     Y(I,J),std,VR0       ;Y(1:N,J) gets loaded into VR0
KLOOP:
        VLDx     M(K,J),std,VR1       ;K(1:N,K) get loaded into VR1
        VSMULx   X(I,K),VR1,VR1       ;VR1 gets VR1 summed with
                                      ;X(I,K) as a scalar
        VVADDx   VR0,VR2,VR0          ;VR0 gets VR0 summed with VR2
        INC      K                    ;increment K by vector length
        IF (K < SIZ) GOTO KLOOP

        RESET    K                    ;reset I to SIZ
        VSTx     VR0,Y(I,J),std       ;VR0 gets stored into Y(I,J)
        INC      I                    ;increment I by vector length
        IF (I < SIZ) GOTO IJLOOP
        INC      J                    ;increment J by vector length
        RESET    I                    ;reset I to SIZ
        IF (J < SIZ) GOTO IJLOOP
        MSYNC                         ;synchronize with scalar
```

Figure 9 Core Loop of a BLAS 3 Routine Using Matrix-Matrix Operations

obtained a vector speedup of approximately 25, as shown in Figure 7.

Signal Processing—Fast Fourier Transforms

The Fourier transform decomposes a waveform, or more generally, a collection of data, into component sine and cosine representation. The discrete Fourier transform (DFT) of a data set of length N performs the transformation following the strict mathematical definition which requires $O(N^2)$ floating point operations. In 1965, the fast Fourier transform (FFT) was developed by Cooley and Tukey. FFT reduced the number of operations to $O(N \times LOG[N])$, which is a significant improvement for computational speed.[8]

As shown in Figure 10, the complex data in the bottom butterfly is multiplied in each stage by the appropriate weight. The result is then added to the top butterfly and subtracted from the bottom butterfly. If the algorithm is left in this configuration, it must use non-unity stride vectors, very short vectors, or masked arithmetic operations to perform the very small butterflies.

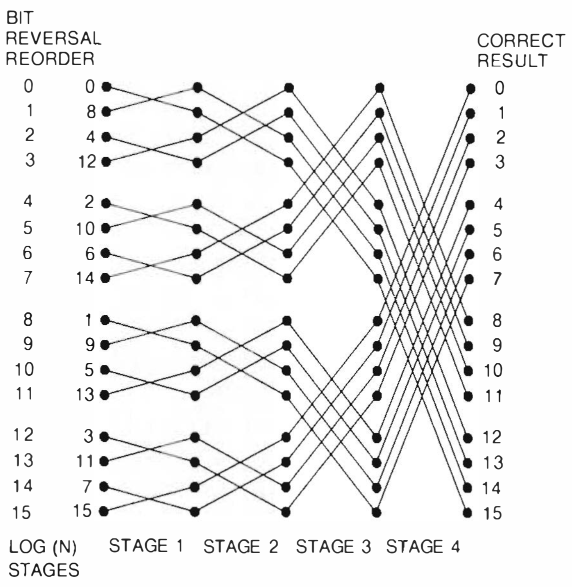


Figure 10 The Cooley-Tukey Butterfly Graph, One-dimensional Fast Fourier Transform for N = *16)*

Optimized One-dimensional Fast Fourier Transforms

The bit-reversal process that permutes the data to a form that enables the Cooley-Tukey algorithm to work is also shown in Figure 10. When using vectors, a common approach to performing the bit-reversal reordering is to use vector gather or vector scatter instructions.[9] These instructions allow vector loads and stores to be performed using an index register. Vector loads and stores require a constant stride. However, vector gather and scatter operations allow the user to build a vector of offsets to support indirect addressing in vector mode. Both gather and scatter instructions are available with VAX vectors.

A vector implementation of the FFT algorithm has been developed that is well suited for the VAX vector architecture. One optimization made to the algorithm involves moving the bit-reversal section of the code to a place where the data permutation will benefit vector processing. By doing so, two goals are accomplished. First, the slower vector gather operations are moved to the center of the algorithm such that the data will already be in the vector cache. In Figure 10, the first FFT stage starts out with large butterfly distances. After each stage the butterfly distance is halved. For the optimized version shown in Figure 11, the bit-reversal permutation is performed as close to the center as possible, when the *stage number* = $LOG(N)/2$. To complete the algorithm, the butterfly distances now increase again. Second, this process entirely eliminates the need for short butterflies.

Another optimization made to the FFT algorithm is the use of a table lookup method to access the sine and cosine factors, which reduces repetitive calls to the computationally intensive trigonometric functions. The initialization of this trigonometric table has been fully vectorized but shows only a modest factor of 2 performance gain. To build the table, a first-order linear recurrence loop is formed that severely limits vector speedup. Because this calculation is only done once, it becomes negligible for multiple calls to the one-dimensional FFTs and for all higher dimensional FFTs. The benchmark shown in Figure 12 was looped and includes the calculation of the trigonometric table performed once for each FFT data length.

Reusing data in the vector registers also saves vector processing time. The VAX vector architecture provides 16 vector registers. If all 16 registers are used carefully, data can be reused by two successive butterfly stages without storing and reloading the data. With half the number of loads and stores, the vector performance almost doubles.

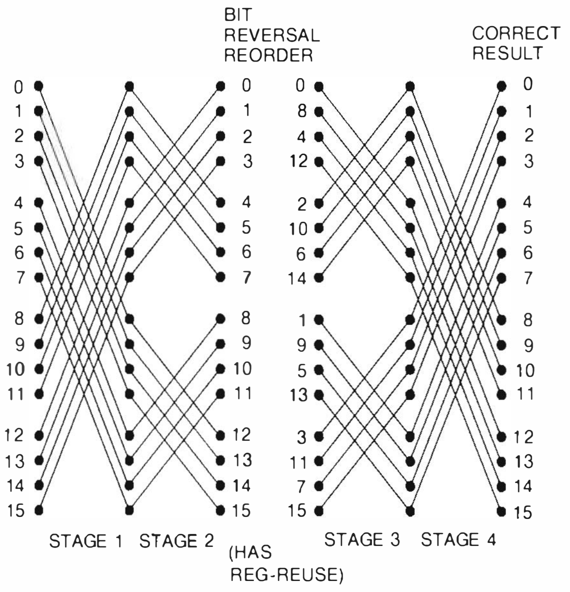


Figure 11 Optimized Cooley-Tukey Butterfly Graph, One-dimensional Fast Fourier Transform for N = *16*

Optimized Two-dimensional Fast Fourier Transforms

The optimized one-dimensional FFT can be used to compute multidimensional FFTs. Figure 13 shows how an N by N two-dimensional FFT can be computed by performing N one-dimensional column FFTs and then N one-dimensional row FFTs. The same routine can be called for column or row access FFTs by simply varying the stride parameter that is passed to the routine. (Note: In FORTRAN, the column access is unity stride and the row access has a stride of the dimension of the array.)

For improved performance on VAX vector systems, the use of a matrix transpose can dramatically increase the vector processing performance of two-dimensional FFTs for large values of N, i.e., $N > 256$. The difference between unity stride and non-unity stride is the key performance issue. Figure 14 shows that a vectorized matrix transpose can be performed after each set of N one-dimensional FFTs. The computation will be equivalent to Figure 10 but with a matrix transpose: each one-dimensional FFT will be column access which is unity stride. The overhead of transposing the matrix becomes negligible for large values of N.

When the value of N is relatively small, i.e., $N < 256$, the two-dimensional FFT can be computed by calling a one-dimensional FFT of length N^2.

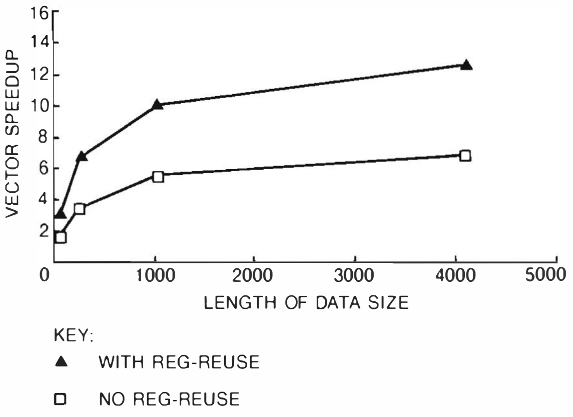


Figure 12 One-dimensional Fast Fourier Transform Performance Graph, Optimized Single-precision Complex Transforms

The small two-dimensional FFT can achieve performance equal to that of the aggregate size one-dimensional FFT by linearizing the data array. Figure 15 shows the trade-off between using the linearized two-dimensional routine (for small N) and the transposed method (for large N) to maintain high performance across all data sizes.

The optimization of an algorithm that vectorizes poorly in its original form has been shown. The resulting algorithm yields much higher performance on the VAXvector 6000 Model 400 processor. High performance is due to the unique way the algorithm touches contiguous memory locations and its effort to maximize the vector length. The implementation described above always uses unity stride vectors and always results in a vector length of 64 for FFT lengths greater than 2K (2×1024).

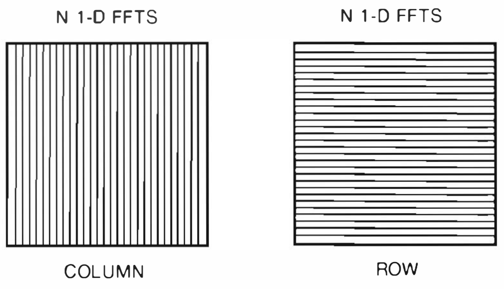


Figure 13 Two-dimensional Fast Fourier Transforms Using N *Column and* N *Row One-dimensional Fast Fourier Transforms*

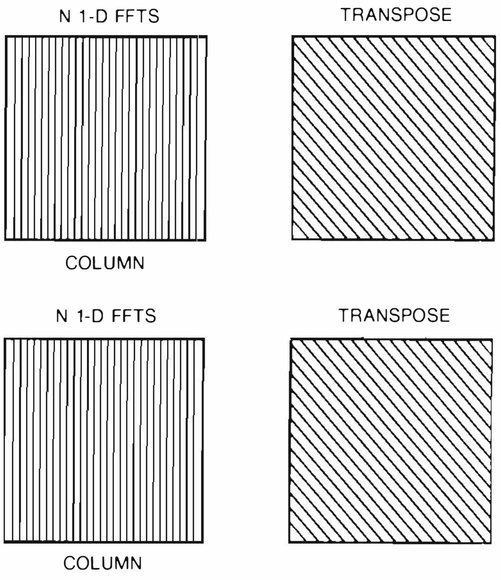


Figure 14 Two-dimensional Fast Fourier Transforms Using a Matrix Transpose between Each Set of N *Column One-dimensional Fast Fourier Transforms*

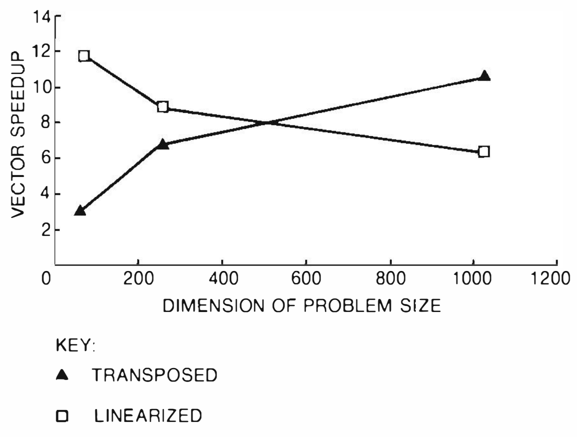


Figure 15 Two-dimensional Fast Fourier Transform Performance Graph, Optimized Single-precision Complex Transforms

Summary

The VAXvector 6000 Model 400 processor delivers high performance for computationally intensive applications. The CMOS-based VAXvector 6000 is capable of operating at peak speeds of 90 MFLOPs single precision and 45 MFLOPs double precision. Linear algebra and signal processing applications that utilize the various hardware features have demonstrated vector speedups between 3 and 35 over the scalar VAX 6000 Model 400 CPU times. With the integrated vector processing available on the VAXvector 6000 Model 400, the performance of computationally intensive applications may now approach that of supercomputers.

Acknowledgments

The authors would like to acknowledge the technical contribution of the following people: Paul Brodeur, Doug Burns, Giao Dau, Bob Dickson, Darrel Donaldson, Hugh Durdan, Bill Gist, Anil Jain, Chandrika Kamath, Dwight Manley, Mike Pline, John Redford, Sean Reilly, Tim Stanley, and Mike Uhler.

References

1. J. Croll, "VAX 6000 Model 400 Multiprocessor System Overview," Proceedings of COMPCON '90 (IEEE, forthcoming 1990).
2. D. Bhandarkar and R. Brunner, "Vector Extensions to the VAX Architecture," Proceedings of COMPCON '90 (IEEE, forthcoming 1990).
3. M. Gavrielov et al. "A 50 MHz Uniformly Pipelined Floating-Point Arithmetic Processor," Proceedings of ISSCC '89 (IEEE, February 1989): 50–51.
4. J. Dongarra, *Performance of Various Computers Using Standard Linear Equations Software in a FORTRAN Environment,* (Argonne, IL: Argonne National Laboratory, June 1989).
5. *VAX FORTRAN Performance Guide* (Maynard: Digital Equipment Corporation, Order No. AA-PB75A-TE, forthcoming 1990).
6. *VAX 6000 Vector Processor Programmer's Guide* (Maynard: Digital Equipment Corporation, Order No. EK60VAA-PG, forthcoming 1990).
7. C. Bischof et al., *LAPACK,* (Argonne, IL: Argonne National Laboratory, April 1989).
8. J. Cooley and J. Tukey, "An Algorithm for Machine Calculation of Complex Fourier Series," *Mathematical Computing,* no. 19 (1965): 297–301.
9. P. Swartztrauber, "Vectorizing the FFT's," *Parallel Computing* (Academic Press, 1982): 51–85.

Patrick Sullivan
Michael A. Callander, Sr.
James R. Lundberg
Rebecca L. Stamm
William J. Bowhill

The VAX 6000 Model 400 Scalar Processor Module

The VAX 6000 Model 400 CPU module is the latest generation of the compatible VAX 6000 family of computers. The Model 400 is a single-board, CMOS-based CPU that significantly extends the performance of the VAX 6000 series. The system provides nearly 7 VAX units of performance (VUPs) in single-processor applications and up to 36 VUPs in six-processor systems. The Model 400 module is a plug-in replacement for the Model 200 and Model 300 processors. Chip set and module designers of this new system cooperated closely to meet aggressive timing and performance goals. Several enhancements were made to the cache and bus interface units to improve multiprocessor performance. A vector interface was included for connection to a companion vector processor module. Signal integrity was an important consideration for both chip and module design.

When the Midrange Systems Business (MSB) Group began to develop the VAX 6000 series, the Semiconductor Engineering Group (SEG) had started development of a new CMOS-based CPU chip set.[1] The project's goals were the following:

- Achieve CPU performance at least equal to the 5.5 VUPs of the VAX 8700 system
- Support a 28-nanosecond (ns) module cycle time

At that time, the VAX 8700 system was the fastest VAX available and used emitter-coupled logic (ECL) to achieve a 45-ns cycle time. SEG designers believed a system implemented in CMOS technology could meet or exceed that performance level and at a much lower manufacturing cost. With reference to the second goal, a 28-ns cycle time would take advantage of chip sets that could run faster than the projected 40 ns cycle time.

In discussions with MSB, we realized our module project could be modified to include an XMI interface and, therefore, become another member of the VAX 6000 series.[2] We then agreed to undertake a joint development effort between MSB and SEG, which resulted in the VAX 6000 Model 400 scalar CPU. Development of this scalar processor module is the focus of this paper.

Halfway through the project, support for a vector processor module was included because the VAX architecture was extended to include vector instructions.[3] The vector processor module, which was developed by MSB, can provide a significant performance boost for certain classes of vector problems.

The first Model 400 systems were shipped in July 1989. This delivery was made just 15 months after the introduction of the initial 6000 Model 200 series.

Design Challenges

The aggressive 28-ns cycle time goal for the module design required a tight coupling of the chip set and module design efforts. With this short 28-ns cycle time goal, the module interconnect had to be treated as transmission lines. Consequently, signal integrity considerations were critical to design success and would impact all areas of the design—chip, package, and etch board. The approach taken to address signal integrity is described in more detail in the Signal Integrity section.

The performance goals also dictated a change in the VAX 6000 Model 400 data and address line (DAL) pin bus. The older, less complex multiplexed 32-bit bus would have to be separated into a separate 27-bit address bus (A-bus) and a 64-bit data bus (D-bus). That decision in turn resulted in the need for high pin count packages (224 pins) and the associated signal integrity challenges of dealing with as many as 90 output drivers switching simultaneously, while driving transmission lines with a low effective impedance of 60 ohms.

Two new technologies were needed to meet these challenges. First, new chip packages had to be developed to supply the increased number of signal pins. The packages use multilayer ceramic substrates to provide signal planes, as well as separate power and ground planes for both internal logic and pad ring power. The packages have 224 pins on a 25-mil pitch and are surface mounted for better module routability. Second, the 25-mil package pin pitch required the use of a finer geometry etch board: 13-mil module vias and 10-mil routing pitch. This specification required the physical design team to initiate a very close development and qualification effort with the etch board vendors.[1]

Major Module Subsections

The Model 400 module, or XRP, is a single-board VAX CPU implemented with the Model 400 chip set and a REXMI interface to the XMI bus.[2] The chip set includes five chips: a CPU chip (REX520), a floating point accelerator chip (F-chip), a backup cache controller chip (VC), a clock distribution chip (CLK), and a system support chip (RSSC). The REXMI interface consists of three chips: two copies of the data path chip (XDP), and a controller/interface chip (XCA).

A block diagram of the XRP module is shown in Figure 1. The module consists of four major sections:

- CPU and F-chip floating point accelerator
- VC chip and backup cache RAM array
- The RSSC system support chip
- The XMI interface, including REXMI

The REX520 is the first VAX microprocessor chip to implement a fully pipelined microarchitecture. The F-chip has 64-bit-wide data paths and a pipelined execution unit. These chips cooperate to implement the base instruction group of the VAX architecture. The two chips represent the CPU section of the XRP module, and both chips connect directly to the DAL. Both chips also have a private 8-bit bus for control and status information.

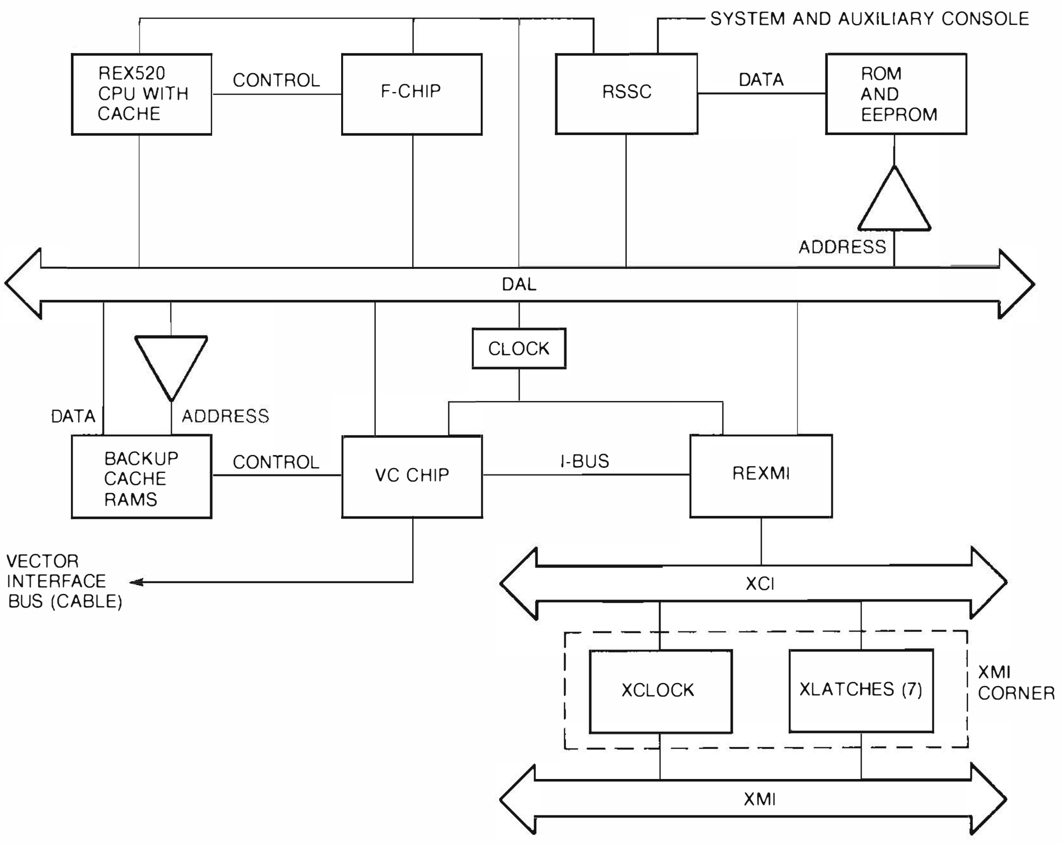


Figure 1 XRP Module Block Diagram

The REX520 provides the hardware and microcode necessary to parse specifiers, execute instructions, handle exceptions, and otherwise implement the VAX architecture. The REX520 also provides full VAX memory management, a 4-gigabyte (GB) virtual address space, and support for 512 megabytes (MB) of physical memory. The chip contains a 64-entry, fully associative translation buffer. Both process and system-space mappings are stored in the buffer. The chip also includes a 2-kilobyte (KB), direct-mapped instruction and data cache (primary cache) with an 8-byte block and fill size.

The F-chip enhances the computation phase of floating point and certain integer instructions in conjunction with the REX520 chip. The F-chip executes the F-, D-, and G-format floating point instructions, as well as the long word variants of integer multiply. The chip receives operands from the REX520, computes the result, and passes the result and status back to the REX520. The REX520 chip completes the instruction.

The VC chip implements a 2K tag store and necessary control for a 128KB backup cache. The cache uses 15-ns, 16K-by-4 static random-access memories (SRAMs) on the module. The VC chip also includes a copy of the primary cache tag store, the invalidate bus (I-bus), and an interface to the vector interface bus (VIB). The I-bus connects to the REXMI and boosts performance by doing invalidate filtering. The cache design is discussed in more detail in the Model 400 Caches section.

The RSSC chip is a modification of the SSC chip used in a number of previous CPUs. It incorporates the common core of functions that support the chip set in the XMI system environment. The RSSC chip is discussed in detail in the Model 400 System Support Chip section.

The XMI interface consists of the standard XMI-corner components and the REXMI chip set. The REXMI chip set interfaces the DAL to the XCI, which is the user side of the XMI corner. The XMI interface is discussed further in the XMI Interface and REXMI section.

DAL operations are synchronous on the XRP module. Therefore, a very low skew clock distribution system was required. The clock chip and controlled, equal-length clock lines to each of the chips provide this distribution. The clock chip receives a 143-megahertz (MHz) oscillator input and provides two sets of synchronous four-phase clocks. Each clock phase driver output can drive four 50-ohm lines in parallel. In the XRP design, there are eight chips that require all four phases, and each set of clock outputs drives four chip loads. Discrete Schottky diodes are used at the receiving ends of all the clock lines for termination. This design achieved a clock skew of less than 0.5 ns at the receiving chips.

The DAL pin bus is a fully synchronous, handshake protocol bus. The DAL is nonpended in that it has only one transaction outstanding at a time, and it is also nonmultiplexed with separate address and data lines. The design consists of a 27-bit address bus (A-bus), a 64-bit data bus (D-bus), and associated control signals. The DAL runs at a 28-ns cycle time, synchronous with the Model 400 chip set. The timing is controlled by 4 overlapping 14-ns-wide clock phases, which are separated from each other by 7 ns. The P-chip, F-chip, VC chip, backup cache RAMs, and the REXMI communicate through the DAL. The P-chip is the default bus master and contains the arbiter for the bus. The P-chip uses the DAL to initiate reads and writes to the backup cache and main memory through the REXMI. The backup cache and the REXMI send read data to the P-chip.

The bus master notifies other bus nodes of the start of a transaction. The receiving node can terminate a transaction in one of three ways. It can indicate a successful completion, indicate an error, or request that the transaction be retried. Once the receiver indicates one of the transaction termination signals, the bus master deactivates.

One DAL transaction takes a minimum of three cycles. The maximum transaction time depends upon the system response to read and write requests. The RSSC includes a bus timeout mechanism that prevents the system from hanging. Most DAL signals are transferred in three phases, although some control signals are transferred in two.

As noted earlier, vector interface capability was added after the VAX architecture was extended to support vector operations. The vector functionality was added to the VC chip because that chip could accommodate the extra pins required for the interface, i.e., 224 pins versus 164.

The major units of the XRP module are described in the following sections.

Model 400 Caches

The XRP module incorporates a two-level cache hierarchy that maximizes CPU performance. The first-level cache is the primary cache, called the P-cache, which is contained entirely in the P-chip. The second-level is the backup cache or B-cache, and it consists of the VC chip and 24 15-ns, 16K-by-4 static RAMs. The VC chip contains the tag store and the control logic for the B-cache. The SRAMs store the cache data. The P-cache and the B-cache contain both instructions and data.

The P-cache is a 2KB cache and is direct-mapped, with an 8-byte block size. The cache is read-allocate, no-write-allocate, and write-through. In write-through, writes that hit in the cache are simultaneously written to the cache and to main memory. The P-cache can perform a new access once every cycle.

Each P-cache tag entry includes an 18-bit tag, one valid bit, and one parity bit. There are 256 tags corresponding to 256 data blocks. Each data block contains 8 data bytes and 8 parity bits, with parity was implemented on each byte. Parity permits a byte write to be done without the need to recalculate parity across the other bytes. This process avoids the performance penalty that occurs when all bytes are not written at once, as in the read-modify-write process.

The B-cache is a 128KB cache and is direct-mapped, with an 8-byte access size. It has a 16-byte sub-block size and a 64-byte block size. The cache is also read-allocate, no-write-allocate, and write-through. There are 2048 entries in the B-cache tag store. Each entry contains a 12-bit tag, 4 valid bits, and a parity bit.

Backup Cache Hit

Designers optimized access time to the backup cache by connecting the cache RAMs directly to the DAL. The chips, the bus, and the specialized hit signals are shown in Figure 2.

When the P-chip issues a read on the DAL, it drives the address on the A-bus to the VC chip and cache RAMs. The P-chip asserts memory read (MEM_RD_L). The VC chip uses MEM_RD_L to enable the assertion of the cache RAM chip select lines (BC_CS_L<7:0>). This process accomplishes two things. The chip select lines are valid by the time the P-chip has driven the A-bus to a valid state. Further, the total time to perform the read from the RAMs is minimized.

The RAM access begins in parallel with the tag store access. If the VC chip finds a match in the tag store, it asserts BC_HIT_L to notify the P-chip. By the time BC_HIT_L is asserted, the data from the cache RAMs is valid on the D-bus, and the P-chip then accepts the data.

If the VC chip does not find a match in the tag store, it asserts BC_MISS_L. This signal notifies the

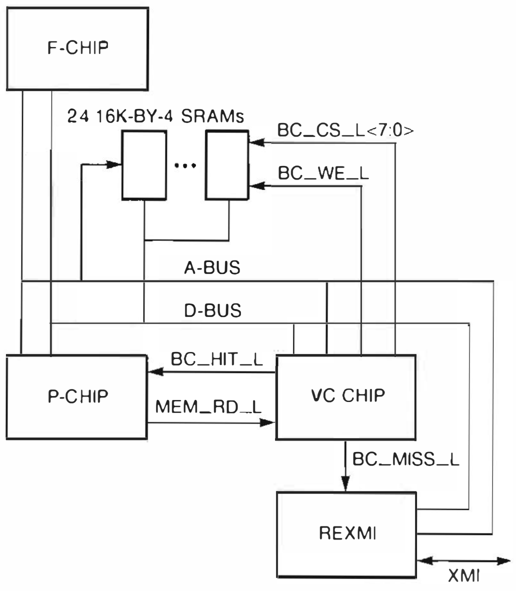


Figure 2 Backup Cache Access Diagram

REXMI to send the read request to memory. At the same time, the VC chip asserts the cache RAM write enable (BC_WE_L) and waits for the data to return from memory. When the REXMI returns the data, it asserts a transaction termination signal. This signal informs the P-chip, the VC chip, and the F-chip that data is ready on the bus. All three chips receive the data simultaneously, and the data is written into the cache RAMs. The transaction ends when the P-chip accepts the data.

XMI Interface and REXMI

The XRP module accesses all memory and I/O devices over the XMI bus. The REXMI interfaces the chip set to the XMI bus by means of two data path chips (XDPs) and one control/address chip (XCA). Each XDP is responsible for 32 bits of the REXMI's 64-bit data path. The XCA is responsible for the address data path, DAL control logic, XMI control logic, and control of the two XDPs. Both chips are implemented in CMOS-2 standard cells.

The primary tasks of the XMI interface are to

- Forward REX520 references to the XMI
- Implement a write buffer that reduces traffic to main memory
- Support control of cache fills and cache invalidates
- Support XMI interrupt logic

CMOS-2 Standard Cell Technology

Custom chips were not used to implement the XCA and XDP chips. Instead, two alternatives were investigated. These alternatives were to use either gate arrays or standard cells from external vendors, or internally built CMOS-2 standard cells.

A number of external vendors offered products with the density and internal gate speeds needed to implement the REXMI chips. However, none offered the performance and flexibility needed to interface to the DAL. Therefore, we chose the CMOS-2 standard cells, which could interface to the DAL.

We added a number of new cells to the standard cell library: several versions of a high-performance 3.3-volt output driver, input latch with the required 0-ns set-up time, and a low skew internal clock buffer capable of driving over 30 picofarads of capacitance. These new cells were largely based on circuits designed for the custom CMOS-2 CPU chips, which met the high performance goals set by the Model 400 program.

Performance Considerations

Performance bottlenecks in high-speed computer systems most often occur in the path to and from main memory. The XRP's two cache levels greatly reduce the number of reads required. However, since both caches are write-through, all writes must be forwarded to main memory by the REXMI. To improve performance, the REXMI implements a write buffer with four octaword (16-byte) entries.

The write buffer improves performance in two ways. First, it decouples the REX520 write rate from the slower XMI write rate. The REX520 can transfer 8 bytes of write data to the REXMI every three cycles. This data is loaded into the write buffer and later transferred to main memory at XMI speeds.

Second, the write buffer combines multiple REX520 writes into a single XMI write. The most efficient write transaction is a full octaword write. The REXMI always tries to combine multiple REX520 writes into full octaword writes. The REXMI loads write data into a write buffer entry. The write data is held until either a new octaword address is received or a purge write buffer condition occurs. When the REXMI receives a write to a new octaword address, the current write buffer entry is marked full. A new write buffer entry is then opened with the new octaword address. Write buffer entries are transmitted as they are marked full. To guarantee that the write buffer data is written to main memory in a timely manner, the write buffer is flushed before the following conditions:

- XMI I/O space read or write
- Interlock read or unlock write
- Interprocessor interlock
- XMI read to an octaword location that includes data contained in the write buffer
- In response to a clear write buffer command

Combining P-chip writes reduces the number of write transactions needed by over 40 percent. This reduction certainly improves single-processor performance. However, the greatest improvement is in multiprocessor performance where the XMI bandwidth required by each CPU is reduced.

Cache Coherence

The XRP module allows data to be shared among multiple processors. The XRP design assures that the most recently written copy of any data is provided to a running process. This process is called cache coherence.

In multiprocessing systems, coherence can be ensured in two ways. Cached copies of data that have been written can be invalidated, or each cache can be updated with more recent data. Since it is simpler to invalidate than to update, an invalidation scheme was implemented on the XRP module.

Every XRP processor write is sent to XMI memory. When an XRP module broadcasts a write on the XMI, the command, address, and data are captured in memory. Other XRP modules capture the command and address to invalidate any valid B-cache or P-cache entries that correspond to the address.

We could have opted to broadcast all XMI writes as invalidates on the DAL. However, this method would have greatly increased the DAL traffic and would have reduced the processor's performance. To increase the performance of multiprocessor systems, the VC chip provides a low-overhead invalidate mechanism through the I-bus. The REXMI can determine through the I-bus if data is currently cached. The REXMI sends an invalidate on the DAL only if the data is cached.

The VC chip maintains a duplicate copy of the primary cache tag store. The chip accesses the copy in parallel with the backup cache tag store whenever invalidate addresses are placed on the I-bus. When an I-bus address match is detected in either tag store, the chip notifies the REXMI of the hit. The REXMI broadcasts the invalidate address onto the DAL. The invalidate address notification is recognized by both the VC chip and the REX520.

The P-chip invalidates the P-cache entry, and the VC chip invalidates the entries in both the B-cache and in the copy of the P-cache tag store.

If the P-cache were a subset of the B-cache, we would not have had to implement the P-cache tag store copy on the VC chip. In this design, any address that hit in the B-cache would have been sent to the DAL, but this process would have caused an invalidate request to be sent to the P-cache as well. However, with this approach, the VC chip would have to send invalidates to the P-cache whenever a B-cache block was displaced. The transmission to the P-cache would have been required because the P-cache and B-cache block sizes are different. It was simpler to implement the copy of the P-cache tags and its control.

Another alternative would have been to use a duplicate copy of the B-cache and P-cache tag stores implemented in external logic on the module. The REXMI then would have interrogated the tag stores directly. This alternative was rejected because it would have used too much space on the module.

The Model 400 System Support Chip

A variety of support logic is required to complete the functionality of a VAX CPU module. The Model 400 system support chip (RSSC) integrates the common core of functions necessary to support the VAX system environment onto a single chip. It provides the operating system with the hardware primitives needed to implement the boot and console routines. The chip also provides several necessary timing mechanisms. The RSSC is designed to interface directly with the Model 400 chips.[1] It is based on the system support chip (SSC) that was designed for use with the earlier CMOS-based VAX systems.[5]

The Vector Interface Bus

The vector interface bus connects the Model 400 CPU module to an optional vector module.[3] The vector module can perform fast calculations on vector data. This capability greatly increases the execution speed of certain applications.

There are two levels to the interface. The first is the microcode that implements the vector instructions defined by the VAX architecture. The second is the hardware implementation features required by the microcode.

The REX520's microcode vector support is limited to decoding vector instructions, parsing the specifiers, and passing operand and control information to the vector unit. The scalar CPU module does not pass the actual vector elements to the vector module through the vector interface. Rather, the vector module references the vector data directly over the XMI bus. To perform this process, the vector module implements the full VAX memory management architecture.

The opcode and operand information is transferred to the vector module through the vector interface. The REX520 uses internal processor registers (IPRs) for read and write operations to transfer the information to the vector interface. IPRs are also used to read and write register information stored both in the interface hardware and on the vector module.

The vector unit executes most vector instructions in parallel with the scalar CPU execution of subsequent instructions. For some vector instructions, particularly memory transfer instructions, the REX520 microcode reads a vector unit register at the end of the instruction. At this point, the REX520 stalls until the vector unit responds and effectively forces synchronous execution of instructions.

The hardware implementation of the vector interface consists of two pieces. The first is an interface to the Model 400 DAL. This interface allows microcode-generated opcode, operand, and register data to be received from and driven to the REX520. The second piece is the vector interface bus (VIB) that connects the vector and scalar modules together over a cable. This interface and the connection to the VIB are implemented in the VC chip on the scalar side and in the VECTL chip on the vector side.

The vector module clock system is asynchronous to the scalar module. The VIB runs synchronously with respect to the vector module clock system for design simplicity. The VC chip implements all the asynchronous control logic that is required to transfer data between the two clocking systems.

Signal Integrity

The XRP design had very aggressive timing goals. For example, 14-ns and 21-ns data transfers were required to meet the 28-ns target cycle time. These requirements were made more difficult by the potential noise problems that could be caused by over 90 drivers switching simultaneously in one package.

Consequently, signal integrity assumed a major role in the XRP design. Signal integrity problems were compounded by fast CMOS-2 edge rates (i.e., 1.5 ns), long interconnects (up to 23 inches with as many as 8 loads per signal), and impedance mismatches associated with capacitive CMOS receivers.

The XRP signal integrity methodology was developed early in the design process. All simulations were made in the SPICE program.[6] Ideally, the entire module would have been modeled. However, the complexity of the module environment and computer resource constraints precluded that approach. Instead, only one driver was used in the simulations. The effects of intersignal coupling and parallel switching were included by appropriately scaling interconnect and package impedances. Even with this simplified method, an estimated 1500 CPU hours on a VAX 8700 system were required to perform the XRP signal integrity simulations.

Simulations and Models

A typical simulation included all circuits and any accompanying parasitics from the external enabling clock edge through to the receivers. All simulations modeled a three-stage clock receiver, on-chip resistance capacitance delay, the output driver, package impedance models for signals and internal/external power, receivers, and module interconnect.

The clock receiver model generated the on-chip clock phases from the low-skew double phases received from the clock chip. The model also contained an internal switching model. It switched several capacitances that were similar to the chip internal loading. Over a cycle, this modeling met the chip's maximum internal power. The switching model, together with the chip package impedance model for internal power, produced worst-case noise on the chip internal power rails and substrate.

The module interconnect was modeled as transmission lines. The impedance of a transmission line accounted for the coupling of adjacent signals switching. For example, if adjacent lines were switching in the same direction, the effective impedance would be increased; whereas switching in the opposite direction would lower the effective impedance.

Finally, the signal and external power impedance models were scaled to reflect worst-case coupling, and size and number of drivers switching in parallel. Collectively, these models accounted for the effects of an entire bus switching.

Worst-case Conditions

There are two distinct subsets of the simulation models. One set simulates worst-case signal noise or ringing. The other simulates worst-case, i.e., slowest, settling times. These subsets are, for the most part, mutually exclusive. In other words, parameters that tend to suppress ringing generally increase settling times and vice versa.

The overall design had to satisfy the criteria for achieving specified settling times and simultaneously reduce ringing to an acceptable level. Several modeling parameters were simulated to worst-case status. These parameters are listed in Table 1. The impact of these parameters can be further understood by referring to Figure 3. This figure represents the same simulation under worst-case slow and fast conditions.

Table 1 Worst-case Modeling Parameters

	Simulation	
Parameter	Worst-case Settling	Worst-case Noise
CMOS process corner	Slow	Fast
CMOS junction temperature	Maximum	Minimum
Internal power supply	Minimum	Maximum
Output series resistor	Maximum	Minimum
Number of drivers switching	Maximum	Maximum
Module interconnect length	Maximum	Maximum
Interconnect effective Z0	Minimum	Maximum

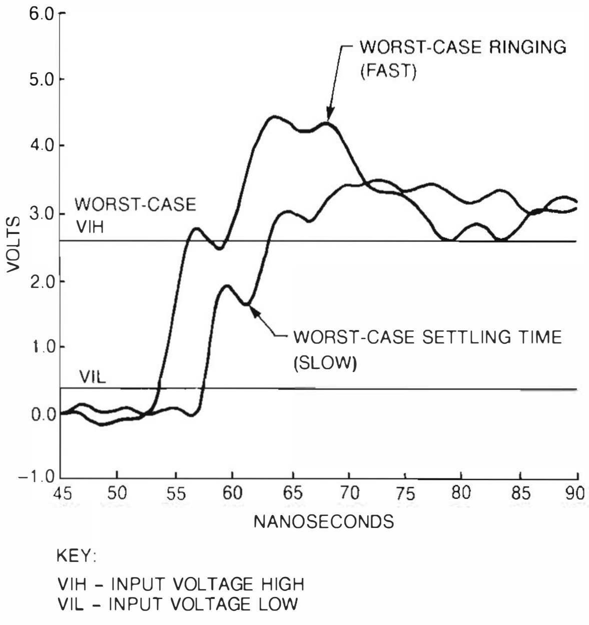


Figure 3 Simulation under Worst-case Slow and Fast Conditions

Signal Integrity Constraints

The signal integrity analysis had a direct impact on the design of the scalar CPU at the chip, package, and module levels. The impact at the chip level was seen in three areas. The first was the design of the I/O buffers. The second was the determination of the optimal series output resistors (on-chip) for different drivers. The third was the segregation of the external power buses to eliminate noise at quiescent drivers.

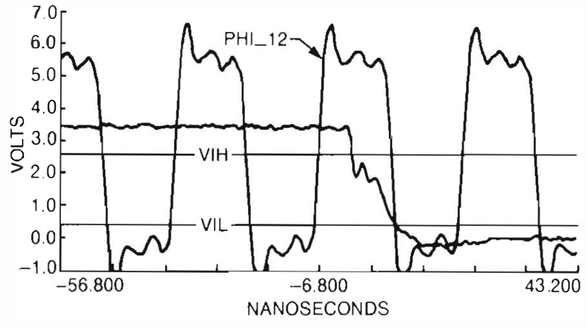
At the package level, two design decisions were made. The external power reference planes were split to prevent coupling of asynchronous buses. Power was delivered to the lower bonding tier to reduce power supply loop inductances. At the module level, several specifications were developed, which included the following:

- All discrete terminations and their placement
- The maximum allowable etch length per signal and order of connection
- The maximum allowable package dispersion etch length
- The module etch technology to best reduce coupling

Cross-organizational cooperation was essential to the successful production of these design levels. For example, a joint review of the chip packaging technology in the design stage ensured that the design met stringent signal integrity requirements. Working closely with the module PC designer ensured optimal component placement and interconnect routing. Cooperative efforts such as these helped ensure the reliability and performance of the design.

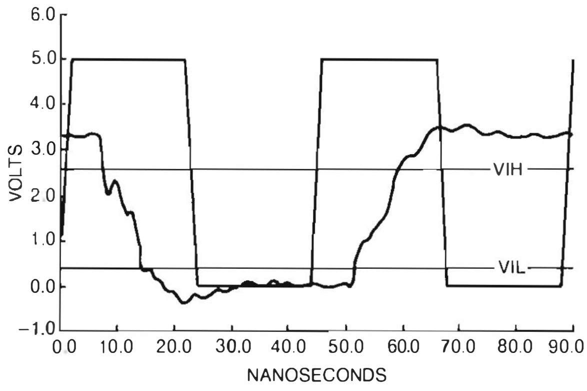
Results

Two correlations of bench versus simulation results were used for verification purposes. The first correlation was run on a test module early in the design phase. The difference between simulated and bench results averaged 4.4 percent. The correlation on the final XRP implementation presented an average discrepancy of 2 percent, which is less than 200 picoseconds (ps). These results strongly validated the modeling methodology. Figures 4 and 5 show the results of the bench and simulation of the REX520 driving the D-bus. The waveforms exemplify the excellent correlation obtained on the final module. (Note: The results shown in these figures represent nominal rather than worst-case conditions.)



KEY:
VIH - INPUT VOLTAGE HIGH
VIL - INPUT VOLTAGE LOW

Figure 4 Bench Results of the REX520 Driving the D-bus



KEY:
VIH - INPUT VOLTAGE HIGH
VIL - INPUT VOLTAGE LOW

Figure 5 Simulation Results of the REX520 Driving the D-bus

Signal integrity had a major impact on the design, performance, and reliability of the Model 400 scalar CPU. All critical signals were carefully simulated and analyzed prior to chip and module implementation. The Model 400 is currently the fastest VAX system in production. The Model 400 also has the distinction of needing *no* revisions, from prototype to final product, for signal integrity purposes.

Performance

The VAX 6000 Model 400 system, based on the XRP module, represents the highest performance VAX system yet released. Performance ranges from nearly 7 VUPs in a single-processor system to 36 VUPs in a six-processor system. The VAX 6000

Model 410 system provides roughly twice the performance of the previous generation CMOS-based system, the VAX 6000 Model 310. It also represents a 28 percent performance gain over the previous generation ECL system, the VAX 8700 system. (Note: One VUP is equal to the performance of the VAX-11/780 system.)

Table 2 compares the performance of the VAX 6000 Model 410 system to other VAX systems.

Table 2 Single-processor Performance Comparison

System	VUPs
VAX 11/780	1.00
VAX 6000 Model 210	2.53
VAX 6000 Model 310	3.38
VAX 8700	5.20
VAX 6000 Model 410	6.64

Up to six XRP modules may be configured in a single VAX 6000 Model 400 system. These modules deliver up to 36 VUPs. Ideally, a linear performance increase is expected as more processors are added to a multiprocessor system. However, a number of factors limit the overall system performance, such as contention for bus bandwidth, increased memory latency, and additional software overhead.

A great deal of effort was expended in the Model 400 design to limit the amount of performance lost in a multiprocessor system.

A number of multistream benchmarks were assembled. These benchmarks were run on the VAX 6000 Model 400 system to efficiently measure performance by simulating real work environments across a number of areas. The results for each system are shown in Table 3.

Table 3 Multiprocessor Performance Comparison

Work Area	1 CPU	2 CPUs	4 CPUs	6 CPUs
Engineering	1.00	1.92	3.71	5.31
Scientific	1.00	1.92	3.74	4.78
Commercial	1.00	1.98	3.80	5.25

Acknowledgments

The authors would like to acknowledge the following people for their contributions to the XRP module development effort: Dick Bagley, Jean Basmaji, Rick Calcagni, Glenn Garvey, John Guildoo, Andy Ladd, Bill LaPrade, Bob McCarty, Dina McKinney, Curt Miller, Roland Ouellette, Dave Rouille, John Sweeney, Mike Uhler, Mike Warren, and John Wilson.

References

1. H. Durdan et al., "An Overview of the VAX 6000 Model 400 Chip Set," *Digital Technical Journal,* vol. 2, no. 2 (Spring 1990, this issue): 36–51.
2. B. Allison, "An Overview of the VAX 6200 Family of Systems," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 10–18.
3. D. Slater et al., "Vector Processing on the VAXvector 6000 Model 400," *Digital Technical Journal,* vol. 2, no. 2 (Spring 1990, this issue): 11–26.
4. J. Bartoszek et al., "VAX 6000 Model 400 Physical Technology," *Digital Technical Journal,* vol. 2, no. 2 (Spring 1990, this issue): 52–63.
5. J. Winston, "The System Support Chip, a Multifunction Chip for CVAX Systems," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 121–128.
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.

W. Hugh Durdan
William J. Bowhill
John F. Brown
William V. Herrick
Richard C. Marcello
Sridhar Samudrala
G. Michael Uhler
Nicholas Wade

An Overview of the VAX 6000 Model 400 Chip Set

The VAX 6000 Model 400 processor is a CMOS implementation of Digital's VAX architecture, offering an average of seven times the performance of the VAX-11/780 processor at a cycle time of 28 ns. The processor comprises five custom chips implemented in Digital's proprietary CMOS-1 and CMOS-2 semiconductor processes. The chip set design incorporates the best features of the previous VAX 8700 and VLSI VAX designs and in addition implements new performance features. Among these are a larger translation buffer and primary cache, a de-multiplexed 27-bit address and 64-bit data bus, and a tightly coupled 128KB backup cache. The five chips, which are designed for multiprocessing environments, are the REX520 CPU, the floating point accelerator, the VC vector and cache controller chip, the RSSC system support chip, and the CLK clock chip.

Introduction

The VAX 6000 Model 400 chip set consists of five custom VLSI (very large scale integration) chips implemented in Digital's CMOS-1 and CMOS-2 processes. The five chips are the CPU chip (REX520), the floating point accelerator chip (F-chip), the vector and cache controller chip (VC chip), the system support chip (RSSC), and the clock chip (CLK chip). These chips are designed to be used in multiple-system environments, of which the VAX 6000 Model 400 series is one such example.[1]

The REX520 chip is a pipelined VAX CPU that implements the VAX base instruction group and controls the operation of all other chips.[2] The design for the REX520 is an evolution of both the previous generation CMOS processor chip and the VAX 8800 processor.[3,4] The REX520 is logically divided into four sections: the I-box, E-box, M-box, and bus interface unit (BIU). The I-box fetches and decodes VAX instructions, and provides this information to the E-box. The microcode-controlled E-box parses instruction specifiers, executes VAX instructions, and processes interrupts and exceptions. The M-box contains a 64-entry, fully associative translation buffer, and a 2-kilobyte (KB) on-chip primary cache. The BIU acts as the interface between the REX520 and the interchip environment described below.

The F-chip is a companion chip to the REX520. It accelerates the computation phase of the VAX F-, D-, and G-format floating point instructions, and the longword-length integer multiply instruction. The F-chip receives control information from the REX520, operands from either the REX520, the backup cache, or memory, and returns status and results to the REX520.

The VC chip provides the tag store and necessary control for a 128KB backup cache that is implemented in external random access memory (RAM) on the CPU module. The chip implements a duplicate tag store for the REX520 primary cache and an interface through which the system environment can determine if data is cached at a particular address. Through a vector interface bus (VIB), the VC chip also provides the control and status interface between the CPU module and an optional vector module.

The RSSC chip incorporates the common core of functions required to support the VAX 6000 Model 400 chip set in a system environment. RSSC supports read-only memory (ROM) and electrically erasable programmable read-only memory (EEPROM), and contains 1KB of battery backed-up RAM, the console terminal universal asynchronous receiver/transmitters (UARTs), interval and programmable timers, and a time-of-year clock.

The CLK chip receives a 143-megahertz (MHz) oscillator input. The chip provides four low-skew clock phases to the chip set and the module environment.

Interchip and Module Environment

The chips in the Model 400 chip set are connected together and to the rest of the module environment as shown in Figure 1.

Unlike the previous generation CMOS processor design,[4] the Model 400 chip set implements separate 27-bit address and 64-bit data buses. Backup cache reads, writes, and fills are done with the address on the A-bus. The data is driven to or received from the parity protected D-bus. Control for the backup cache RAMs is provided by the VC chip on dedicated control lines.

Memory, I/O space, and external processor registers are accessed by driving the address to the A-bus and the data to the D-bus. ROM and EEPROM control is provided by the RSSC on dedicated control lines.

F-chip operands are driven to the D-bus from the REX520, backup cache, or memory to the F-chip. Results are driven back to the REX520 on the D-bus. Control and status information for these transfers is performed on a private bus between the REX520 and F-chip.

Vector instructions are decoded by the REX520. The opcode and instruction operand information is transferred from the REX520 to the VC chip. From there, the information is transferred to the VIB cable and then to the optional vector unit. Note that only status and scalar operands (contained within the instruction stream) are transferred on the VIB. Data transfers into and out of the vector register file are performed by the vector processor through a direct port to the memory subsystem.

Interrupt requests are received by the REX520 from the module environment on nine dedicated interrupt request lines. Five of these lines are for requests for special purpose interrupts, such as interval timer requests.

Performance Goals and Design Considerations

The goal of the design was to meet or exceed the performance of the VAX 8700 processor. To meet this goal, a 40-nanosecond (ns) cycle time was required under worst-case conditions. As the design progressed, it became clear that the CMOS-2 process, in which most of the chip set is implemented, offered enough performance to allow the target cycle time to be decreased.

As a result, the cycle time was reduced from 40 ns to 28 ns. At this cycle time, the VAX 6000 Model 410 system runs at nearly 7 VUPs in most applications, or roughly 1.3 times the performance of the VAX 8700 processor. (The acronym VUPs stands for VAX units of performance; 1 VUP equals the performance of a VAX-11/780 system.) The performance of the system may be further expanded by adding processors to the system, to a maximum of 35 VUPs in the VAX 6000 Model 460.

To achieve the performance goals, a number of microarchitectural trade-offs were made relative to the VAX 8700 and previous VLSI VAX designs. In essence, the best features of each were incorporated into the design of the VAX 6000 Model 400 chip set.

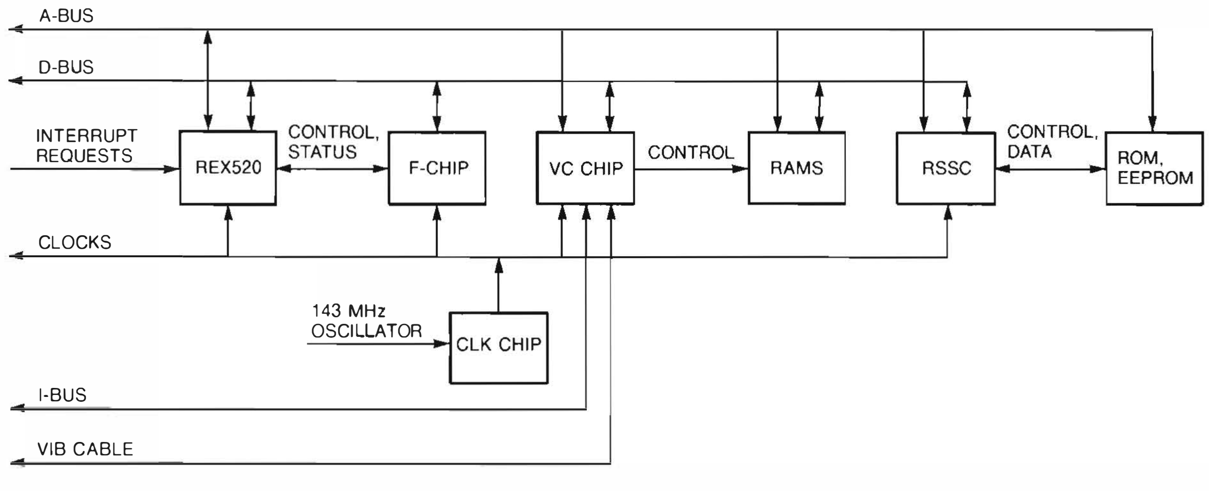


Figure 1 VAX 6000 Model 400 Interchip Environment

For example, the REX520 chip and the F-chip are fully pipelined designs that make shorter cycle times possible and improve performance. The REX520 design also includes read-and-run and write-and-run features. These features decouple subsequent execution in the CPU pipeline from the completion of references to cache and memory. As opposed to previous VLSI designs, a larger translation buffer and primary cache and a de-multiplexed 27-bit address and 64-bit data bus in the REX520 also improve performance. The tightly coupled 128KB backup cache significantly reduces the read latency seen by the CPU when a read misses in the primary cache. It also reduces the read traffic seen by the memory subsystem.

The chip set was designed from the beginning with multiprocessing in mind. Because caches must remain coherent across all CPUs in a system, a method must be provided to invalidate cached locations in all other caches when one CPU writes to that location. One option would have been to mirror all writes done by any CPU in the system onto the A-bus of all CPUs. However, mirroring is a relatively expensive operation, especially when most of these addresses are not cached in any other CPU.

Instead of the mirroring method, we chose to implement a duplicate copy of the primary cache tag store in the VC chip and to implement a low-overhead port, the I-bus. The module environment uses the I-bus to determine if the address is actually cached by either the primary or the backup cache. With this method, only those invalid addresses that correspond to cached locations must be mirrored onto the A-bus.

CMOS and Packaging Technologies

The chip set was implemented in Digital's proprietary second-generation complementary metal oxide semiconductor technology, CMOS-2. We selected CMOS as the chip processing technology because it offers high density, high reliability, low power, and low-cost performance.

By developing the technology in-house, Digital has gained competitive performance and time-to-market advantages for its low-end and midrange products. A summary of the transistor and pin-count statistics for the chip set is shown in Table 1.

CMOS-2 is an N-well, P-epitaxial, double-metal, 5-volt process with 1.5 micron minimum feature sizes. With respect to CMOS-1, the first generation process, CMOS-2 offers a 25 percent reduction in lateral and key vertical dimensions, a 78 percent improvement in circuit density, and nominally a 33 percent improvement in chip performance. The VAX 6000 Model 400 chip set cycle time and circuit density requirements helped drive the development of the process that has been optimized for microprocessor chip performance. Details of the process features and capabilities are shown in Table 2.

Metal oxide semiconductor field effect transistors (MOSFETs) are built in a P-epitaxial layer (30 ohm-cm) grown on a low-resistance P+ silicon substrate (0.02 ohm-cm). The high resistance of the epitaxial layer keeps parasitic junction capacitance low and allows better transistors to be fabricated. The low-impedance substrate dramatically reduces latchup, a phenomenon in which parasitic bipolar transistors are triggered into a sustained high current mode. Latchup disrupts normal circuit operation and often destroys the chip.

The P-channel MOSFETs are made in the N-doped well regions of the epitaxial layer. The N-channel MOSFETs are made in the as-grown P-regions of the epitaxial layer. This process optimizes the mobility of the N-channel MOSFETs and overall circuit speed.

An N+ polysilicon (polycrystalline silicon) and tungsten-disilicide sandwich material, polycide, forms the MOSFET gates. Polycide resistance is in order of magnitude lower than that of the polysilicon used in CMOS-1. The resulting smaller parasitic delays across the MOSFET gates and local interconnections help improve circuit speeds.

Table 1 Summary of Chip Statistics

				Transistor Count	
Chip	Signal Pins	Die Size	Power Dissipation	Memory	Control & Data Path
REX520 CPU	157	12mm x 12mm	6W	180K	140K
VC cache controller	178	10.7mm x 10.8mm	2.5W	184K	34K
F-chip floating point	103	12.7mm x 11mm	4W	—	134K
CLK clock	20	5mm x 5mm	2W	—	5K
			Total:	364K	313K

Table 2 CMOS-2 Process Features and Capabilities

Effective channel length	0.9 microns
Metal 1	3.0 micron width, 1.5 micron space
Metal 1 contact	1.5 micron × 1.5 micron
Metal 2	3.75 micron width, 1.5 micron space
Metal 2 contact	1.5 microns × 1.5 microns
Gate oxide thickness	225 angstroms
Metal 1 field oxide	1.1 microns
Metal 2 field oxide	1.9 microns
Polycide resistance	2-4 ohms/square
Typical gate delay	300-500 picoseconds
Polycide equivalent gate delay length	400 microns
Metal 1 equivalent gate delay length	5000 microns
Metal 2 equivalent gate delay length	8000 microns

Other interconnections are accomplished through two layers of aluminum. The first layer, metal 1, can connect to either polycide or the N+/P+ source/drain regions via metal 1 contacts. The upper layer, metal 2, can connect to metal 1 through metal 2 contacts. No other connections are allowed. Based upon its parasitic delay characteristics, polycide is used only for local interconnections. Metal 1 is used for signals communicating across distances less than half of the chip dimensions. Metal 2 is used for global signals, clock distribution, and power and ground distribution.

Because of the speeds and complexities of the VAX 6000 Model 400 chips, noise is a particularly difficult problem for the chip designer. CMOS-2 contains a deep P+ implant that can be used to provide a very low resistance connection between the top surface of the chip and the P+ substrate. The chip designer uses this deep P+ implant to reduce substrate noise that can upset the operation of dynamic circuits.

The VAX 6000 Model 400 chips were packaged in custom-designed, rigid perimeter-leaded, single-chip ceramic packages. The packages included four power and ground planes to help maintain inter-chip signal integrity and allow full-speed operation of the chip set.[5]

The REX520 CPU Chip

The REX520 CPU chip is a third-generation, single-chip VAX microprocessor. The REX520 provides the hardware and microcode sufficient to parse operand specifiers, execute instructions, and handle interrupts and exceptions. It cooperates with the F-chip to implement the base instruction group of the VAX architecture.

Although the REX520 hardware organization and placement resemble that of the previous generation microprocessor, the REX520 performance goals were met by tailoring the microarchitecture more closely to the VAX 8700 processor.[3,4] Any deviations from the microarchitecture of the ECL-based VAX 8700 system were made in the CMOS-based REX520 to compensate for technology differences and to exploit the beneficial aspects of VLSI design. A photomicrograph of the REX520 is shown in Figure 2.

The chip employs a six-level pipelined engine built around three autonomous pipes. These pipes provide simultaneous instruction prefetch and decode, instruction formatting, operand reference, execution, address translation and result store, and I/O access.

As shown in Figure 3, the major hardware functions of the REX520 are partitioned into the following:

- An instruction box (I-box) that contains the instruction decoder and a 16-byte prefetch queue (PFQ)
- A microcode-controlled execution box (E-box) that provides the capability for data manipulation in a 32-bit data path
- A memory box (M-box) that implements VAX memory management by utilizing a 64-entry, fully associative translation buffer
- A 2KB write-through, direct-mapped primary cache (P-cache) with a quadword (8 byte) fill size
- A bus interface unit (BIU) that controls a fully handshaked, synchronous chip bus

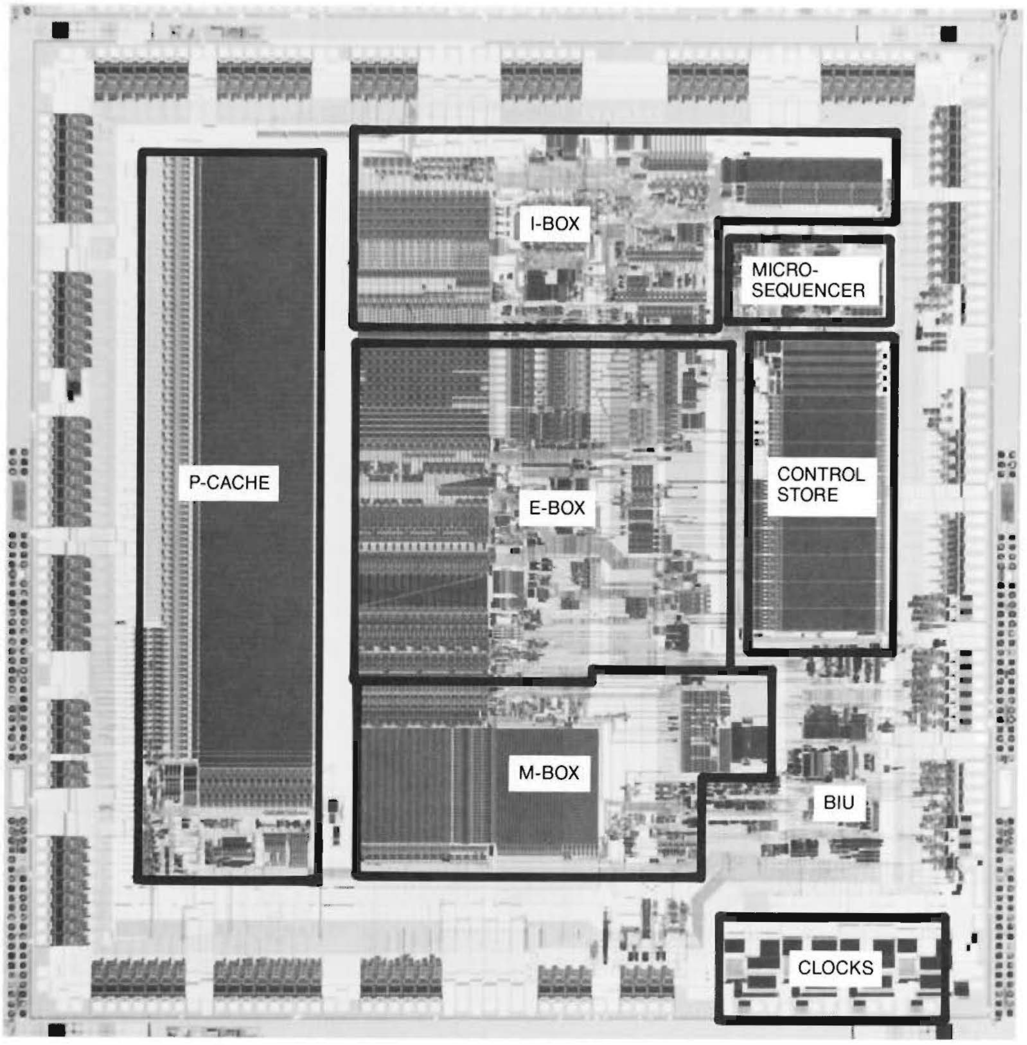


Figure 2 Photomicrograph of the REX520 Chip

REX520 Pipeline

The REX520 pipeline contains six functional segments that cooperate in the execution of instructions. As shown in Figure 4, there are two segments in the I-box, three microcode-controlled segments in the E-box and M-box, and a single segment for the BIU and P-cache control.

The I-box decodes the VAX instruction stream. A 16-byte prefetch queue (PFQ) is filled with the instruction stream asynchronously to the pipeline control during otherwise unused bus cycles. I-box segment 1 updates the PFQ and parses the next piece of the instruction stream. This segment sends microcode addresses to the E-box microsequencer. Segment 2 formats immediate data, the opcode, and the instruction data length for the E-box. Collectively, this information is called the context for the instruction.

The I-box divides each VAX instruction into a microcode subroutine, or microflow, for each specifier, and a microflow for the execution of the instruction. The control programmed logic arrays (PLAs) in the I-box cause it to sequence through the specifiers of each instruction, sending a microflow address to the microsequencer, and immediate data (if needed) to the E-box for each specifier. When the last specifier is parsed, the I-box sends a microflow address, the opcode, and the data length of the instruction to the E-box for the execution of the instruction.

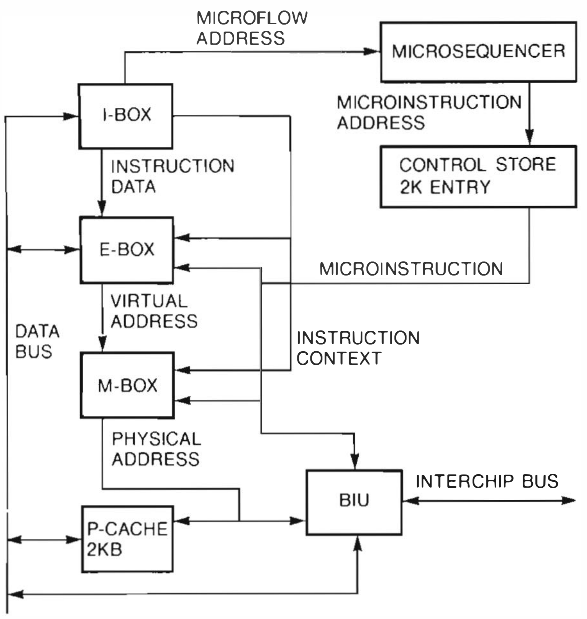


Figure 3 REX520 Block Diagram

The I-box pipeline runs autonomously to the E-box pipeline. That is, the I-box segments continuously parse instruction stream data, making microflow addresses available. The I-box pipeline advances whenever the microsequencer accepts a microflow address.

The microsequencer in the E-box performs a read of the 1696 word control store each cycle, overlapping in time with I-box segment 2. It presents the fetched microinstruction to E-box segment 1. The microinstruction is pipelined forward to E-box segments 2 and 3 in consecutive cycles.

The microsequencer fetches one or more microinstructions of a microflow starting at the initial microflow address supplied by the I-box. If a microflow contains more than one microinstruction, the microsequencer computes subsequent intraflow microaddresses and fetches the corresponding microinstructions. A field within the microinstruction indicates the end of each microflow.

The E-box performs all the address and data manipulation required for the REX520 to adhere to the VAX architecture. The three E-box segments operate under microinstruction control; the operand fetch segment reads operands from general-purpose registers (GPRs) or from the memory data (MD) file and presents them to the functional units; the execution segment performs data manipulation on the operands; and the result store segment writes results to registers or memory. The E-box pipeline segments are fully folded, i.e., each segment simultaneously operates on a different microinstruction.

The performance of memory read accesses is improved using a read-and-run technique in the E-box. The destination for data stream memory reads is the E-box MD file. Microinstructions that initiate a memory read simply queue the request to the BIU. The microflow may then continue, without waiting for the request to complete. In this manner,

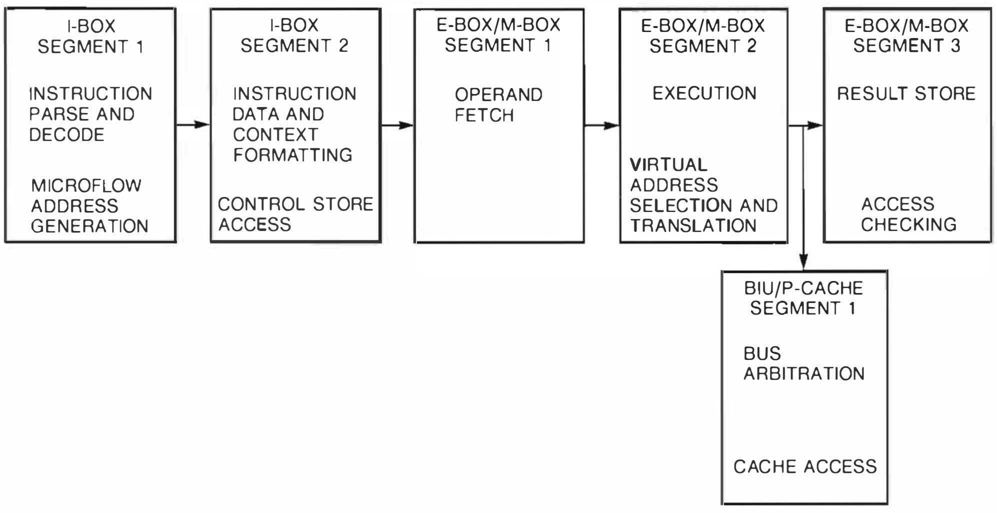


Figure 4 REX520 Pipeline Segments

some or all of the time required for memory accesses is hidden during productive microflows. The request completes when the BIU and memory subsystem return data to the MD file.

Because subsequent microinstructions may reference an MD file location for which a memory read has not yet completed, MD file accesses are synchronized through a valid bit mechanism. Each MD file location has a valid bit that is reset when a memory read is started. The bit is then set when data is written to the file. If a microinstruction attempts to reference an MD file location whose valid bit is not set, the pipeline stops advancing (stalls) and waits for the data to be returned.

The three M-box segments are microcode-controlled and typically run synchronously with the E-box segments. M-box segment 1 decodes the current microinstruction. Segment 2 then selects an appropriate address source and performs a virtual to physical address translation if required. The final segment issues the physical memory request to the BIU.

During normal pipeline flow, the BIU and P-cache operate in a single pipeline segment which is overlapped with segment 3 of the E-box/M-box pipeline. The BIU acts as an arbiter for the external and some internal buses, and supplies control to the P-cache. The BIU receives address and memory request information from the M-box. The BIU then decides whether the M-box or an external reference should receive service from the P-cache and sends the appropriate information to the P-cache to process the request. The BIU sends read and write requests that cannot be serviced by the P-cache to the off-chip memory subsystem.

Write buffers in the BIU improve the performance of memory write operations. Microinstructions that initiate memory writes queue the request to the BIU, and the microflow continues without waiting for the request to complete. In this manner, the time required for memory writes can be hidden under productive microflows. The BIU can store up to two quadword writes (32 bytes) in its buffers while waiting for the memory subsystem to become available.

Adaptations for VLSI

VLSI designs have relatively good signal integrity on the major bus structures within a chip. However, signal integrity diminishes when crossing a chip boundary because off-chip bandwidth is incapable of sustaining the internal data rates. This characteristic of VLSI design influences microarchitecture greatly, as demonstrated by the P-cache and control store designs.

The VAX 8700 system uses cache memory to decrease the effective memory access time by employing a 64KB cache in ECL RAM with a single-cycle access. It was not possible to implement a RAM of this size on the REX520, and signal integrity constraints dictated a minimum of three cycles to access off-chip RAM. The REX520 compensates by using a two-level cache. The REX520 has a 2KB primary cache on the chip with a single-cycle access, and there is a 128KB off-chip secondary cache. The cache hierarchy reduces the effective memory access time sufficiently to meet the performance goals, and it keeps the off-chip bandwidth within reasonable rates.

Similar signal integrity constraints are imposed on the REX520 control store design. The VAX 8700 system uses a 16K entry control store implemented in ECL RAM. The VAX 8700 system designers use the large control store address space to improve performance and simplify hardware. The control store access of the VLSI REX520 cannot cross a chip boundary and keep pace with the fast internal cycle time. Because of these limitations, an on-chip control store ROM was used. Chip area limitations constrained the size to 1696 entries. The designers of the REX520 compensated for the lack of control store address space by altering the coding style to make microflows more serial than the VAX 8700 system microflows. Additional hardware features also helped to absorb some of the work previously done in microcode.

The operand specifier microflows offer an example of microcode space compression facilitated through hardware enhancements. The REX520 microflows group similar specifier types together and provide a general routine for each. The general routines are parameterized in hardware to resolve the differences in specifier types within each group. In this way, the size of the microflow is minimized, and the performance characteristics of a dedicated microflow are preserved.

Each general specifier microflow performs the function appropriate to the specifier mode by using a parameterized access type and data length. For example, a microinstruction in the specifier flows may make an access-type dependent memory request. If the I-box supplied a read access type, the memory request hardware initiates a read. If the I-box parsed a result store specifier, the write access type parameter causes hardware to initiate a write. Operand data length is supplied in a similar fashion.

The general microflows leave operands in fixed locations in the E-box MD file. The exact location in the MD file is also supplied by the I-box.

Parameterizing the microflows with the access type, data length, and MD file location allows the generalized specifier microflows to be shared. The MD file parameter creates an independence between the specifier and execution microflows. The same specifier microflow can then be used for each specifier mode, independent of where it actually appears in the instruction. The execution microflow also needs to know nothing about the kind of specifier that supplied the operand.

The F-chip

The floating point accelerator chip, F-chip, is the companion processor chip to the REX520 for floating point operations in VAX 6000 Model 400 systems. A photomicrograph of the chip is shown in Figure 5.

The F-chip implements all VAX base instruction group floating point instructions and the longword

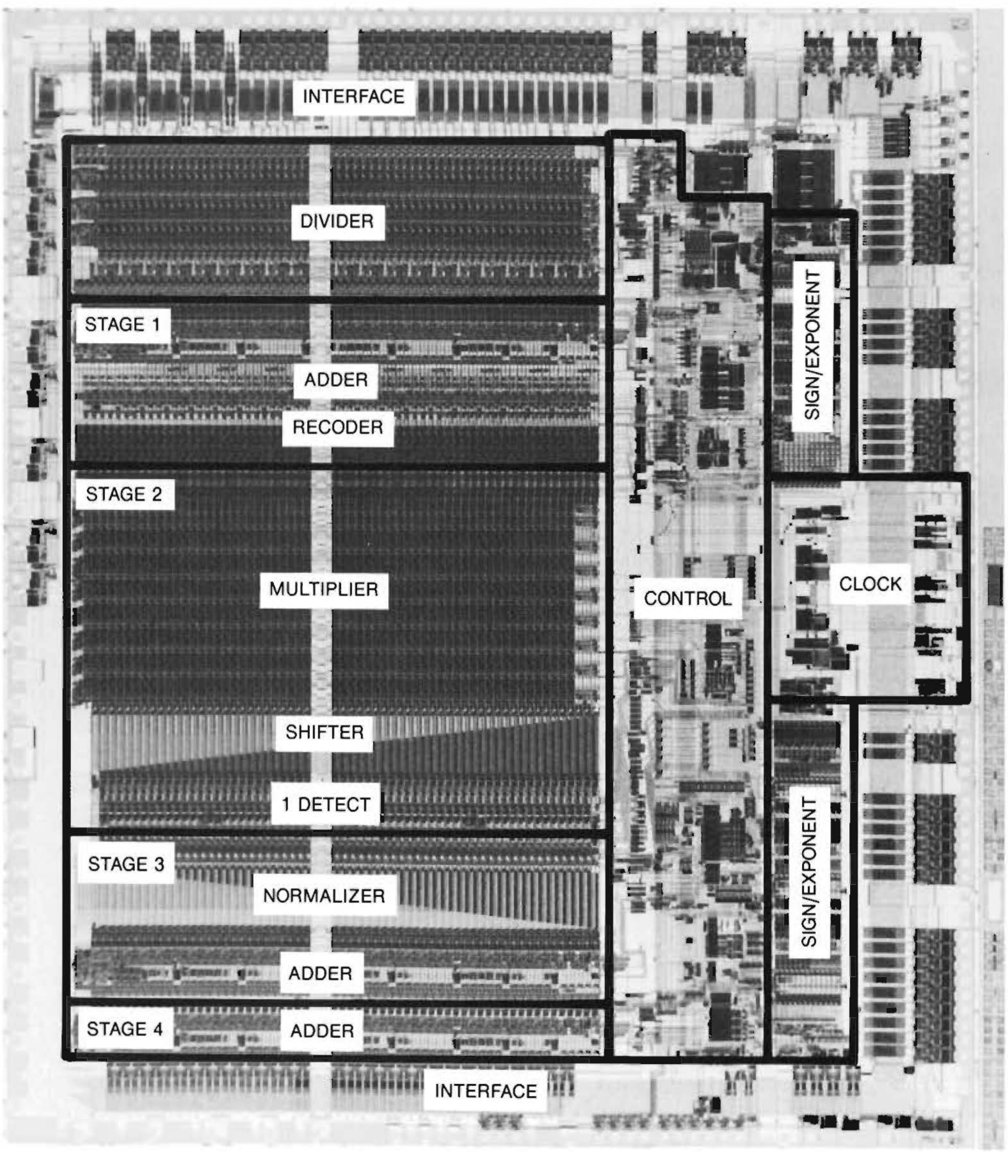


Figure 5 F-chip Photomicrograph

integer multiply instruction. The data types implemented by the F-chip are F-floating (1-bit sign, 8-bit exponent, 24-bit fraction), D-floating (1-bit sign, 8-bit exponent, 56-bit fraction), G-floating (1-bit sign, 11-bit exponent, 53-bit fraction), and integers (8-bits, 16-bits, and 32-bits).

Unlike the VAX 6000 Model 200 floating point chip, the F-chip employs a uniformly pipelined architecture that has performance which is independent of the operand values. Early in the design it was decided to use a pipelined microarchitecture in order to use the same execution core in future designs, such as the VAX 6000 Model 400 vector processor floating point unit. Consequently, the execution core of the F-chip was designed to include low latency for scalar applications and high throughput for vector processing.[6] The core executes most instructions in four cycles. Double-precision and integer multiply instructions take five cycles, and divide operations take 13 to 24 cycles, depending upon the data type.

Microarchitecture and Implementation

The F-chip is composed of an interface section and a five-stage execution core. A block diagram of the F-chip is shown in Figure 6. The F-chip interfaces to the rest of the VAX 6000 Model 400 system through the D-bus, eight status and control lines, and four bus control signals. The interface section receives the opcode from the REX520, the operands from the CPU chip, and cache and memory data on the D-bus. The interface section decodes the opcode and receives the required number of operands. It assembles the different pieces of the operands and supplies formatted operands to the execution core. After the execution of the floating point operation, the output interface transfers the formatted result back to the CPU chip.

The execution core consists of a divider, which is bypassed in all operations except division, and four pipelined stages that are uniformly utilized in the execution of all instructions. Each stage has a fraction data path, control, and sign and exponent data

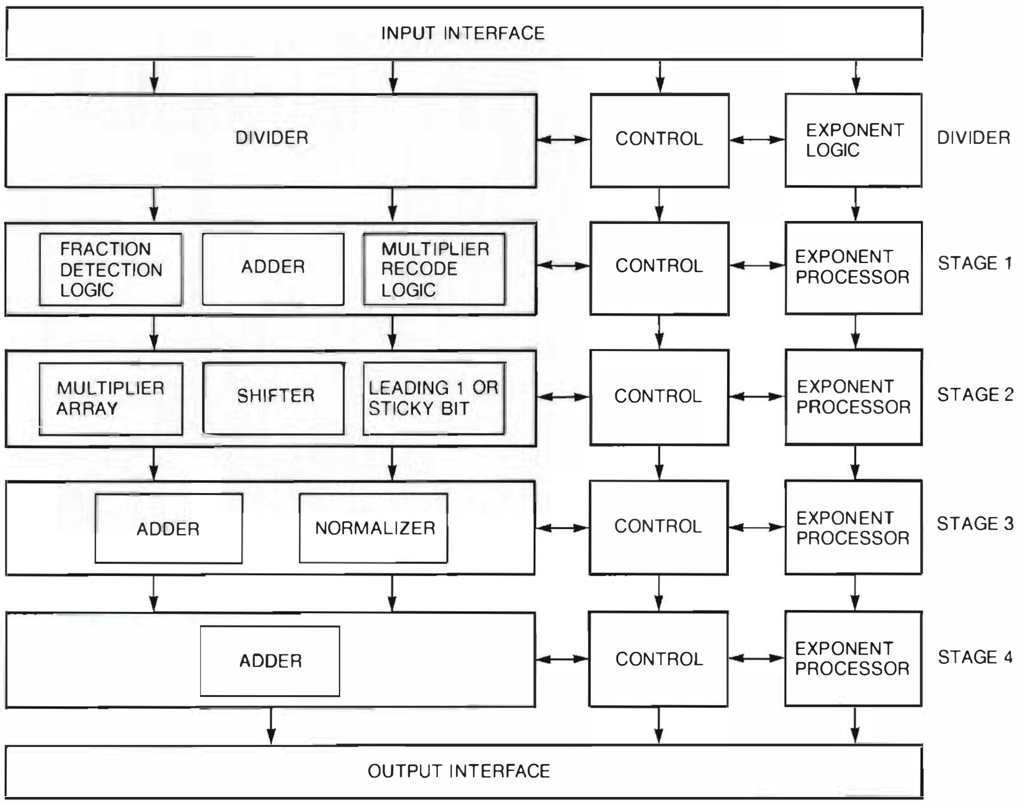


Figure 6 F-chip Execution Core

paths. The fraction data path, the most complex portion of the chip, is 60 bits wide.

The divider fraction data path consists of the hardware divider array and quotient logic. The divider array implements an iterative radix-2 SRT nonrestoring division scheme, and generates three signed quotient bits per cycle.[7] The quotient and remainder are driven on consecutive cycles to stage 1 of the pipeline.

The stage 1 fraction data path consists of a 60-bit adder, the multiplier recode logic, and the fraction detection logic. Stage 1 receives its inputs from the interface section, or from the divider for divide operations. In add and subtract operations, stage 1 is used primarily to compute the difference between the exponents of the two operands. In parallel, stage 1 is also used to determine the fraction difference for exponent differences of 0 and 1.

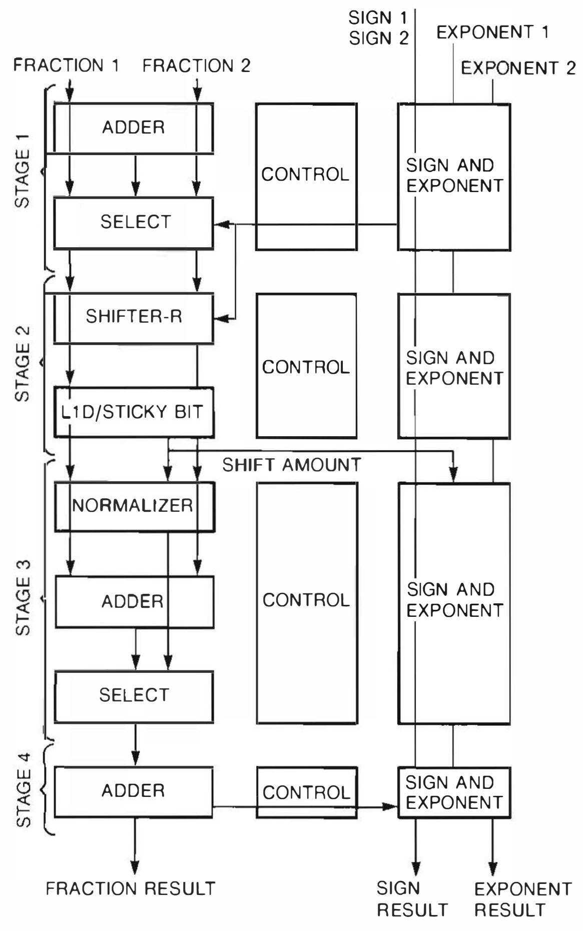


Figure 7 F-chip Hardware for Addition and Subtraction

(See Figure 7.) For the effective subtract operation, stage 1 uses an exponent difference prediction scheme in which the least significant two bits of the two exponents are examined to determine whether 0 or 1 bits of alignment are required.

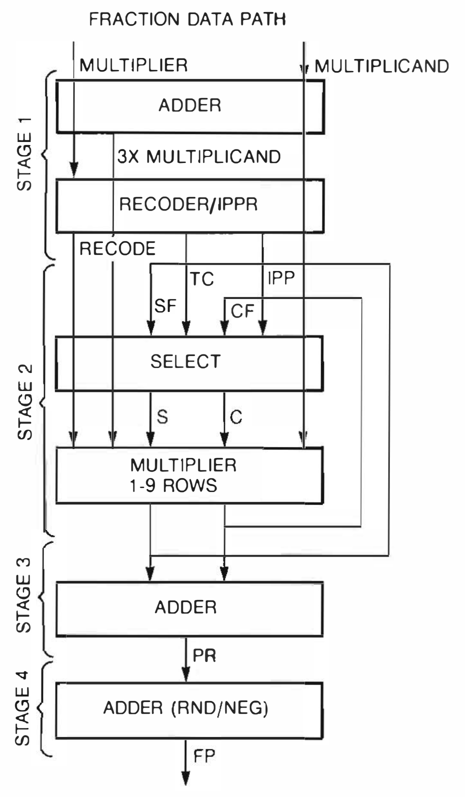
If the actual exponent difference is 0 or 1, stage 1 selects the adder result. Otherwise, stage 1 passes the original operands to the next stage for alignment. In multiplication flows, the stage 1 adder is used to compute three-times the multiplicand, and the recode logic generates the recoded multiplier bits for stage 2. In divide operations, stage 1 is used to assimilate the redundant quotient and remainder vectors into a two's complement form.

The stage 2 fraction data path consists of a dynamic 57-bit right shifter, unified leading one or sticky-bit detection logic, and a multiplier array. (The sticky-bit indicates whether a "1" was shifted out of the data path during alignment operations.) The shifter is used in add-like operations for aligning the fractions. The leading one or sticky-bit logic determines the normalization amount during subtract operations and integer-to-floating-point conversions. This logic also determines the sticky-bit during alignment in effective subtract flows. The multiplier array implements a radix-8 modified Booth algorithm and consists of nine carry save adder (CSA) rows. It is traversed once for single-precision formats and twice for integer and double-precision formats. The hardware organization for the multiply operation is shown in Figure 8.

Stage 3 of the pipeline consists of a 60-bit adder and a 57-bit dynamic left shifter for normalizing the intermediate results. The fraction adder is used in multiply operations to assimilate the product sum and carry vectors, and in add-like flows to add or subtract the aligned operands.

Stage 4 of the pipeline consists of a 60-bit adder used for rounding and negation of the final result, and the exception detection logic. The exception handling is done with a PLA which detects and signals overflow, underflow, zero results, and invalid operands.

The control is hardwired (no microcode is used) and distributed among the pipeline stages. Each stage has autonomous control implemented with control decoder PLAs. Each stage supplies control information to the next stage one cycle ahead of the data. This enables the data path control signals to set up before the data is propagated to the input of the stage. The distributed control scheme permits simultaneous execution of a divide and four other instructions in the pipeline.



KEY:

SF, CF	SUM AND CARRY OUTPUTS
TC	TWO'S COMPLEMENT VECTOR
IPP	INITIAL PARTIAL PRODUCT
S, C	SUM AND CARRY INPUTS
PR	PRODUCT
FP	FRACTION PRODUCT

Figure 8 F-chip Multiplication Fraction Data Path

The exponent data path in each stage is 13 bits wide. Each stage has a 13-bit adder and detection logic for detecting zero operands, exponent differences, and exception conditions. To compute the absolute value of the exponent difference in add, subtract, and convert operations, the stage 1 exponent data path has an additional 13-bit adder and selection logic.

The microarchitecture and the hardware organization of the F-chip were chosen to efficiently implement the three basic operations add/subtract, multiply, and divide. As an example of this efficiency, the adder in stage 1 is used in both the effective subtract flow and the multiplication and division flows. This flow overlap permits the execution of effective subtraction in only four steps, independent of the data, as opposed to five steps without the overlap.

For multiplication, this organization enabled the use of a radix-8 algorithm, which requires the computation of three-times the multiplicand. The implementation of all other operations was realized with the addition of minimal logic.

The VC Chip

The VC chip, or the backup cache controller and vector interface chip, implements the second level of a two-level cache structure and the interface to an optional vector processor. A photomicrograph of the VC chip is shown in Figure 9.

Cache Control Functions

The REX520 contains a 2KB primary cache data and tag store. The VC chip contains the tag store and control logic to perform reads, writes, and invalidates to a 128KB secondary, or backup, cache. The backup cache is direct-mapped and write-through.

The VC chip backup cache tag store is organized such that one tag and four valid bits correspond to every four-octaword (64-byte) block of the cache. Each valid bit corresponds to a one-octaword sub-block, as illustrated in Figure 10. When a cache tag miss occurs on a read, a block is allocated, a sub-block is filled, and the corresponding valid bit is set. When a cache tag compare is successful but the valid bit is not set, a sub-block is filled from memory and the corresponding valid bit is set.

The Backup Cache Data Store

The backup cache data store is built with off-the-shelf CMOS static RAMs, which are located on the module. The discrete cache data RAMs are organized as 8 bytes wide (the width of the D-bus) by 16,384 locations deep. In addition, there are eight bits of parity, with one bit corresponding to each data byte. Fourteen bits of the VAX physical address are needed to access the cache, as shown in Figure 11. When the backup cache returns data to the REX520 primary cache, it returns one quadword, the fill size of the primary cache.

The Invalidate Filter Bus, I-bus

As mentioned, the chip set implements a special-purpose bus, the I-bus, that allows the memory subsystem filter invalidates. In addition to this bus, the VC chip also implements a copy of the REX520 primary cache tag store for invalidate filtering.

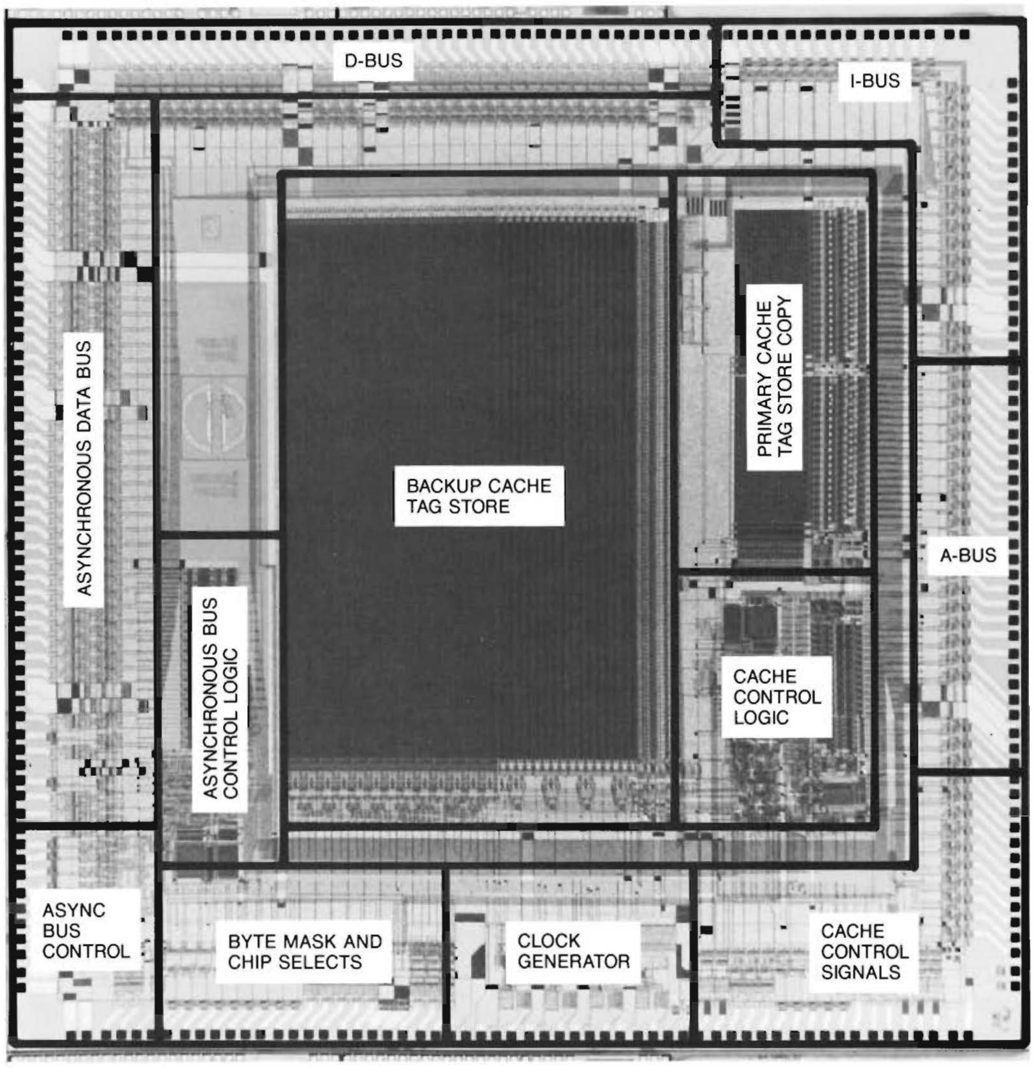


Figure 9 VC Chip Photomicrograph

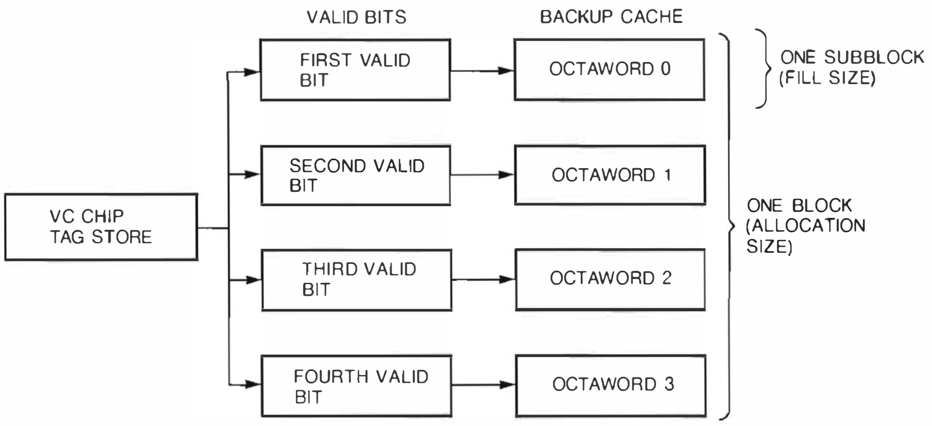


Figure 10 Tag, Valid Bits, and Backup Cache Store

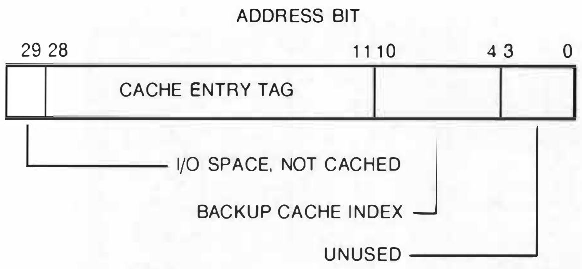


Figure 11 VAX Physical Address and Backup Cache RAMs

When an address is sent to the VC chip on the I-bus, the VC chip notifies the memory interface as to whether or not the address produced a hit in either cache. The memory subsystem broadcasts the invalidate onto the A-bus only if a match was detected. This mechanism significantly improves performance in multiprocessor systems by reducing the overall traffic on the processor buses.

The Vector Interface

The VC chip implements the interface logic that allows an optional vector unit to be connected to the scalar processor module. The vector unit has the potential to significantly increase the performance of many analytical applications.

If the vector unit is present in the system, the REX520 scalar CPU chip decodes vector instructions and passes operand and control information to the vector module through the VC chip vector interface. The vector unit, using its own memory interface, accesses vector data from memory.

The vector and scalar modules are connected through the vector interface bus (VIB). The VC chip is the sole master of the VIB. The VIB is asynchronous to the VC chip and synchronous to the vector module. The VC chip implements the asynchronous control logic required for transferring data between the two modules. The decision to operate the VIB asynchronously with respect to the VAX 6000 Model 400 module was made in order to simplify the design of the vector module.

The RSSC Chip

A variety of support logic is required to complete the functionality of a VAX CPU module. The VAX 6000 Model 400 system support chip (RSSC) integrates on a single chip the common core of functions necessary to support the VAX system environment. A photomicrograph of the RSSC is shown in Figure 12.

The RSSC provides the operating system with the hardware primitives needed to implement the boot and console routines, and with several necessary timing mechanisms. The RSSC is designed to interface directly with the VAX 6000 Model 400 chips, and is based on the system support chip, or SSC.[8]

The decision to base the RSSC chip on the previous SSC design was made because we could take advantage of an existing core of logic which already implemented the required functions. The SSC was fabricated in the CMOS-1 process and had been thoroughly debugged and qualified for the MicroVAX 3500/3600 system.[9] The challenge faced by the RSSC team was to design a new pad ring and bus interface unit (BIU) that would interface the existing SSC core to a much different and faster interchip environment.

RSSC Functions

The functions that the RSSC provides for the VAX 6000 Model 400 module can be grouped into two categories: boot and console code support, and timer functions to support the operating system.

Some of the important console and boot code support functions contained in the RSSC are the ROM/EEPROM interface, CPU halt-request protection, UARTs with programmable baud rates, 1KB of standby RAM, and input and output ports which interface to several console and module-level switches and LEDs. The RSSC also implements three timer functions: the bus timeout counter, a 10-millisecond interval timer, and a time-of-year clock.

The CLK Chip

Clock generation for the VAX 6000 Model 400 chip set was placed in a separate chip, the CLK chip, to better control interchip skew. By separating the clock, we eliminated power, pin count, area, and noise problems from the other possible home of the clocks, the REX520. Three main factors were taken into account in designing the CLK chip. These factors were the number of clock edges, clock skew, and signal integrity. A photomicrograph of the CLK chip is shown in Figure 13.

In determining the optimal configuration for the number of clock edges, the CMOS microarchitecture was examined closely. The number of clock edges per microcycle determines the granularity available to the designer. Increased granularity can simplify the design process by reducing the need for self-timed design techniques. However, the usefulness of partitioning the design cycle is limited. Granularity on the order of a gate delay is of

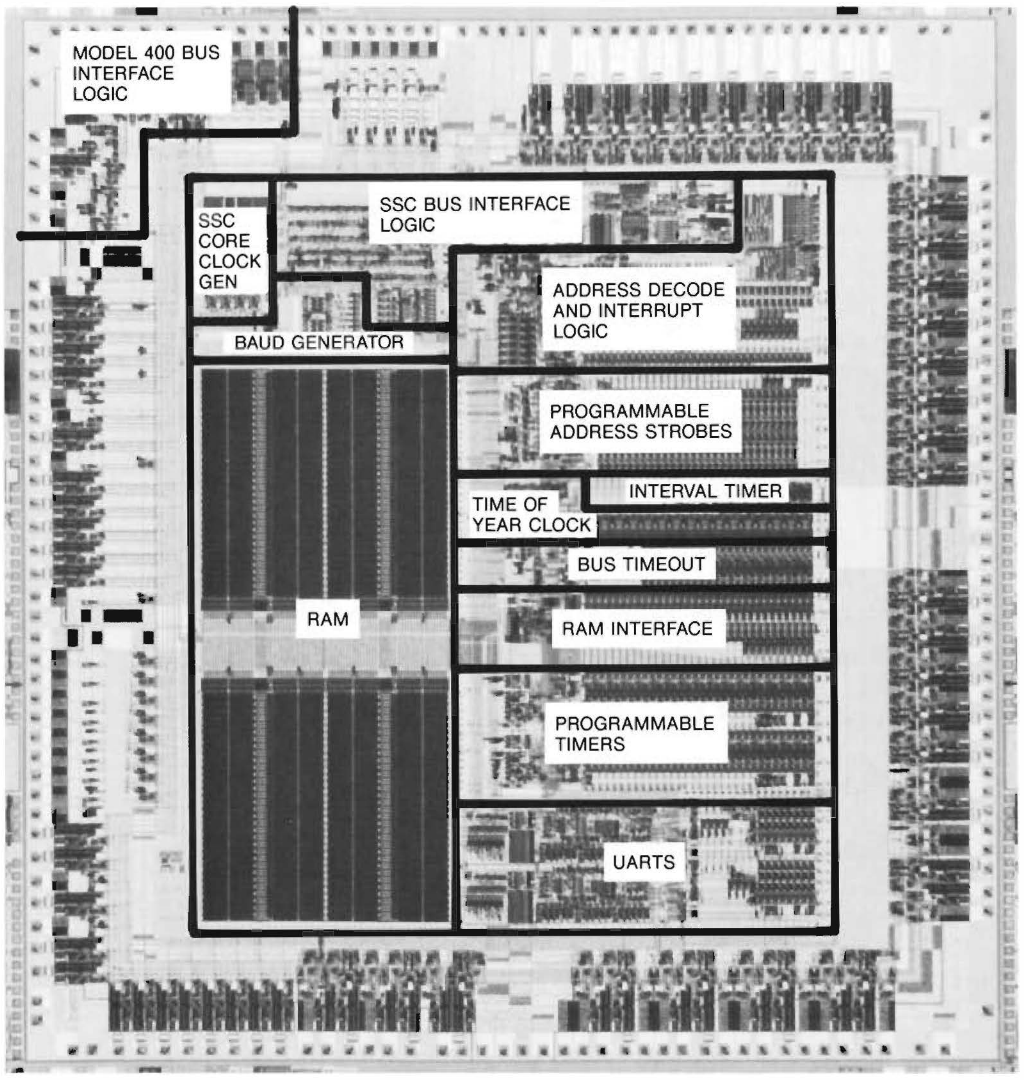


Figure 12 RSSC Photomicrograph

little use, whereas granularity on the order of a microcycle imposes the use of self-timed circuits. A decision was made to distribute four 14-ns pulses phased in time by 7 ns to facilitate the design of regular arrays, data path structures, and I/O functions. This distribution made master-slave clock generation and single-phase generation easy, and complementary clocks for latches more available.

Clock skew was minimized through logic and circuit techniques. The clocks are generated from a 143-MHz crystal oscillator. The frequency is immediately divided by two to generate an even duty cycle 71.5-MHz signal and its complement (PHI and PHI_BAR). The divide circuitry, which comprises four D-type flip-flops, is shown in Figure 14. CLK is the clock input, D is the data input, Q is the latched true data output, and QB is the latched complement data output.

CLK is the 143-MHz crystal oscillator clock. The circuit consists of a D-type flip-flop with one master, D1, and three slaves D2, D3, and D4. D2 provides local feedback for the divide-by-two function in order for D3 and D4 to match identically. D3 is the Q slave and D4 is the QB slave.

D3 allows the inversion in the feedback path to be absorbed and the Q and QB to see the same number of gate delays. This logic technique works for all dividers used in the design. However, the

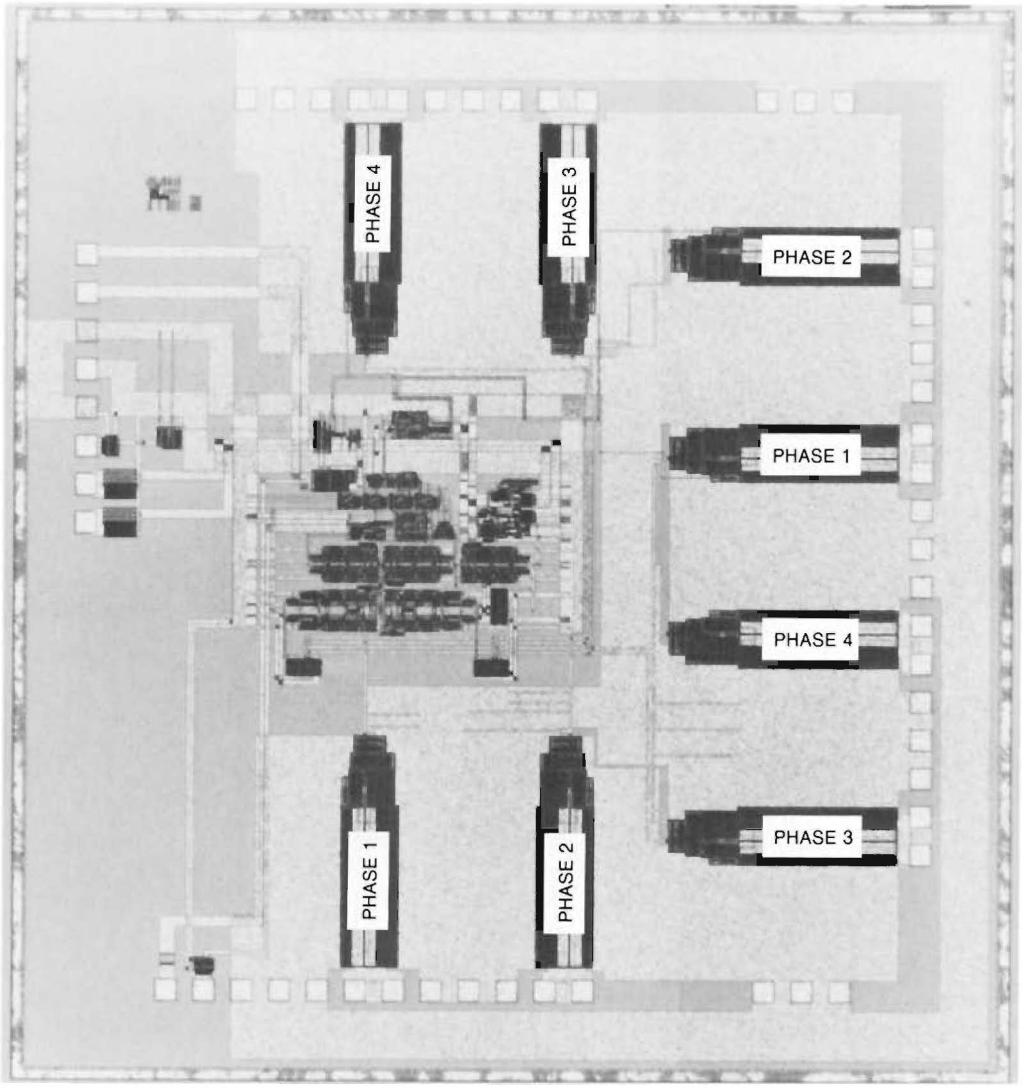


Figure 13 CLK Chip Photomicrograph

dividers feed a series of large output drivers in which skew could not be controlled through logic techniques. In these cases, skew was minimized by matching the capacitance on each node. The end result was a design in which the skew between any two edges generated by the CLK chip was not greater than 0.3 ns.

Signal integrity was carefully considered in the design. The clocks had to be distributed to eight different loads, and the total skew between any two edges could not exceed 0.5 ns. This design implied that the CLK chip package and the clock module interconnect could not contribute more than 0.2 ns of skew. To minimize the electrical impact of the package, three power pins, three ground pins, and two signal pins were used for each driver.

This pin design reduced the inductance seen by each driver by approximately a factor of two. In

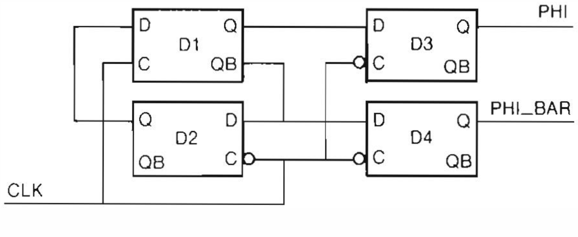


Figure 14 CLK Chip Divide Circuitry

addition, the clocks were radially distributed, and diode termination was used on each clock leg. The length of each leg was identical.

Conclusions

The VAX 6000 Model 400 chip set represents a successful mapping of the ECL-based VAX 8700 system microarchitecture into a CMOS-based VLSI chip set. The fully folded pipeline, on-chip cache and control store, and read and run/write and run strategies of the processor chip, combined with a high performance floating point processor and a second-level cache, enabled the VAX 6000 Model 400 to exceed the original performance goals for the system.

Acknowledgments

The authors would like to acknowledge the following people for their contributions to the VAX 6000 Model 400 chip set development effort: Randy Allmon, Brad Benschneider, Mary Jo Butler, Sandy Carroll, Gerry Cheney, Larry Commodore, Beth Cooper, Amnon Fisher, Nannette Fitzgerald, Annette Flohr, Moshe Gavrielov, Ed Gomes, Paul Gronowski, Bill Grundmann, Ellen Kagan, David Kravitz, Liam Madden, Vijay Maheshwari, Steve Martin, Karen McFadden, Roger Meeks, George Mills, Mike Minardi, Millind Mittal, Dave Morgan, Victor Peng, Jeff Pickholtz, Jim Reinschmidt, Doug Sanders, Katie Siegel, Rebecca Stamm, Pete Starvaski, Bob Supnik, Bill Wheeler, Jeff Winston, and Bill Upham.

References

1. P. Sullivan et al., "The VAX 6000 Model 400 Scalar Processor Module," *Digital Technical Journal,* vol. 2, no. 2 (Spring 1990, this issue): 27–35.
2. T. Leonard, ed., *VAX Architecture Reference Manual* (Bedford: Digital Press, Order No. EY-3459E-DP, 1987).
3. T. Fox et al., "The CVAX 78034 Chip, a 32-bit Second-generation VAX Microprocessor," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 95–108.
4. S. Mishra, "The VAX 8800 Microarchitecture," *Digital Technical Journal,* vol. 1, no. 4 (February 1987): 20–33.
5. J. Bartoszek et al., "VAX 6000 Model 400 Physical Technology," *Digital Technical Journal,* vol. 2, no. 2 (Spring 1990, this issue): 52–63.
6. B. Benschneider et al., "A Pipelined 50-MHz CMOS 64b Floating Point Arithmetic Processor," *IEEE Journal of Solid State Circuits,* vol. 24, no. 5 (October 1989): 1317–1323.
7. D. Atkins, "Higher Radix Division Using Estimates of the Divisor and Partial Remainders," *IEEE Transactions on Computers,* vol. C-17 (October 1968): 925–934.
8. J. Winston, "The System Support Chip, a Multifunction Chip for CVAX Systems," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 121–128.
9. G. Lidington, "Overview of the MicroVAX 3500/3600 Processor Module," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 79–86.

John T. Bartoszek
Robert J. Hannemann
Stephen P. Hansen
Robert J. McCarty
John C. Sweeney

VAX 6000 Model 400 Physical Technology

The physical realization of the VAX 6000 Model 400 microprocessor design offered a number of significant challenges at both the chip package and the module levels. In meeting the requirements for a robust and manufacturable midrange implementation, the VAX 6000 Model 400 physical technology approach broke new ground for Digital, and, in some cases, for the industry. New developments included the first tape-automated bonding (TAB) interconnected semiconductors, extensive board-level physical simulation, and the use of advanced testability features on a microprocessor-based midrange product. This paper provides details of the physical technology used in the VAX 6000 Model 400 project to achieve system-level product goals.

Introduction

As shipped beginning in July 1989, the VAX 6000 Model 400 microprocessor chip set is one of the fastest and highest performance complex instruction set computer (CISC) CPUs offered by the industry. Its microcycle time is 28 nanoseconds (ns). Such performance makes the chip set one of the most demanding in terms of physical technology design at the chip package and single-board-computer (SBC) level. This paper details the requirements that drove the VAX 6000 Model 400 physical technology and describes the resulting technology solutions. The range of solutions included the design process, module assembly, and advanced test features.

The VAX 6000 Model 400 project was the first Digital microprocessor-based system effort that required developers to use large-computer design tools and processes. During the project, a number of firsts, for either Digital or the industry, were achieved.

- The use of advanced tape-automated bonding (TAB) technology in a manufacturing environment
- The development of innovative surface-mountable chip packages that provide high lead count and a controlled electrical environment for the custom CMOS CPU and bus interface chips
- The design and development of an advanced printed wiring board (PWB) technology that allowed over 5000 inches of interconnecting wiring on only four routing layers of a 9-inch by 11-inch board, with two different controlled impedance levels
- The extensive use of electrical and thermal simulation at the chip package and module levels
- The employment of an advanced surface-mount technology (SMT) that allows the use of mixed component styles (50-mil SMT devices and fine-pitch, 25-mil devices, and a limited number of through-hole mounted devices); and attachment of both active and passive components on both sides of the board
- The use of advanced test techniques, most notably an innovative continuity transistor structure (CTS) for assembly verification and boundary scan design for the core CPU chips

All of the project's technology developments will be employed on follow-on products, in both midrange and entry-level systems. Ensuring this kind of technology extensibility was our explicit goal.

In the remainder of this paper, physical technology refers to the chip packaging and interconnection technologies at the individual device and module levels.

Technology Requirements

The physical technology requirements that drove the VAX 6000 Model 400 technology effort arose from two primary sources: architectural and performance requirements, and manufacturing and reliability related needs.

Architectural and Performance Requirements

The CMOS-2 semiconductor process technology and the microarchitecture were chosen to satisfy the system performance goal of at least six VAX units of performance (VUPs). That choice established the baseline physical technology performance and electrical requirements.

One processor design goal in particular had a significant effect on the physical interconnect technology. The processor was to operate at a CPU microcycle time of 28 ns at the SBC level. Although the nominal target of the chip set design was operation at 40 ns, early indications were that the CMOS-2 process would allow significantly faster operation. Therefore, the rest of the physical technology needed to support the 28-ns goal.

Table 1 summarizes the major features of CMOS-2 and the VAX 6000 Model 400 architecture that drove the physical technology requirements. Chip packaging and module technology requirements were affected in two ways:

- The wide data bus could result in many drivers switching simultaneously. Coupled with the relatively fast driver rise times, simultaneous switching significantly increased package electrical performance. Our major concern was the high current draw from local (within the package) power planes. The electrical performance required of the chip packages clearly indicated that simple cerquad packages would not work. The primary shortcoming of the cerquad packages was the high inductance of the signal and power connections. Performance levels required multilayer ceramic packages with features that were at the limit of available ceramic technology.

  A wireability estimation technique used early in the program indicated that four to six routing layers would be needed in the PWB, using 5-mil line widths and 5-mil spacing between lines, to route the core CPU chips with associated cache.

- The relatively high power dissipation requirements contained in the initial design specification posed considerable development difficulty.

  The core chip set was expected to be used in low-end products with limited air-flow capabilities. However, a junction temperature limit of 85 degrees Celsius at the DEC standard 102 Class B environmental limits had to be maintained.

How we addressed these problems is described in the later sections, Chip Packaging and Module Technology.

Table 1 VAX 6000 Model 400 CMOS-2 Features Affecting Interconnect Technology

Driver rise time	1 ns
Clock cycle - 4 phase cycle	28 ns
Data bus width	64 bits
Cache access time	7 clock phases (49 ns)
Max. chip pin count	224
Max. chip power	6 watts

Table 2 lists the key CMOS-2 chips developed for the VAX 6000 Model 400 processor and the key characteristics of these chips that affected the physical architecture of the design.

Table 2 VAX 6000 Model 400 Chip Set Characteristics

Chip	Lead Count (Actual)	Maximum Lines Switching (Approximate)	Power (W)
P-chip/DC520	224	81	6.1
F-chip/DC523	224	80	4.5
VC-chip/DC592	224	60	4.2
XDP/DC550	224	100	3.6
XCA/DC551	224	70	3.6

A more direct measure of the electrical performance required from the VAX 6000 Model 400 physical interconnect system can be derived by examining the timing budget allocated to some of the signal transfers that occur within the CPU. Three of the most critical are the clock distribution system, the cache access loop, and the data bus between the core CPU chips. The clock signals need to be valid and synchronized within 0.5 ns at the end points of all clock lines. The clock electrical performance relied heavily on the uniformity of the distribution system.

The cache access loop timing was slightly more complicated. The cache access loop timing budget that was established for the 28-ns version of the CPU is shown in Table 3. The interconnect system primarily affects the address bus (A-bus) settling time (20.5 ns).

Table 3 Cache Access Loop Timing Budget

Address bus settled	20.5 ns
Buffer	7.0 ns
RAM access	21.0 ns
Total	48.5 ns

Manufacturing and Reliability Requirements

Certain manufacturing and reliability goals established for the VAX 6000 Model 400 product also influenced the physical technology selection and design. The major goals were as follows:

- The use of TAB as the off-chip interconnect technology, which was viewed as an appropriate entry point for TAB
- An all surface-mount technology (SMT) module assembly approach
- A robust test strategy, including significant use of fault diagnosis using boundary scan and other test features
- Maintenance of the ability to perform engineering changes using etch cuts on the PWB by restricting signal routing to the second and ninth layers, with clock lines only on the remaining two signal layers
- Achievement of the required 85 degree Celsius junction temperatures

Most of these goals were met:

- The worst-case A-bus and data bus (D-bus) settling times were 19.6 ns and 20.2 ns, respectively. These settling times met the timing requirements as defined in Table 3.
- A nearly total surface mount assembly was achieved, with only connectors, oscillators, and erasable programmable read-only memories (EPROMs) in through-hole configurations.
- Boundary scan was designed into the core CPU chips. Observe-only scan latches were used at the chip boundaries as opposed to the original target of full scan at all logic boundaries.
- Test transistor structures for interconnect verification at module assembly were implemented for the custom chips.
- Separate, system-specific micro heat sink designs allow all chips to operate at a maximum of 85 degrees Celsius, with one exception: the 6.1 watt P-chip reaches 89 degrees Celsius under absolute worst-case conditions. Although not specifically meeting the defined goal, overall system reliability and operations were not felt to be impacted to any significant degree by allowing this exception.

Some goals were not met.

- The interchip routing included over 1000 signal nets and over 1800 routed nets within the PWB. This large number of nets required the use of the clock routing layer to complete the interconnect of the CPU module. Although the PWB technology supported 5-mil lines and spaces, the signal integrity constraints of the system required more than 5-mil spaces between signal lines. In addition, the ability to do etch cuts on a 5-mil line was determined to be too difficult and risky to pursue. Engineering changes were performed by doing cuts in the surface etch, which connects each surface-mount pad to its associated via, which in turn connects to an internal signal layer.
- TAB tape and packages were designed, proven, and sourced for six of the 224-pin CMOS-2 chips in the VAX 6000 Model 400 processor. However, for manufacturing logistics and line-loading reasons, TAB is currently used in only a subset of these devices. A series of wire-bonded backup packages was designed and is now in use for the remainder of the 224-pin devices.

Nonetheless, the physical interconnect technology used did result in package designs and a module design that supports 7 VUPs, 224 I/O, fine-pitch devices, with associated cache, operating at 28 ns.

The details of the chip packages, the PWB technology and module design are presented in the following sections. Also described are the test structures required to satisfy the performance needs and other established goals of the VAX 6000 Model 400 CPU product.

Chip Packaging

In March 1989, Digital's Semiconductor Interconnect Technology Group (SCIT) completed qualification of the Model 400 processor advanced chip packaging technology.

This qualification consisted of two primary elements, which are discussed in the sections below. The first element is the initial implementation of our internally developed perimeter tape-automated bonding (PTAB-1) process. The process serves as the interconnection medium from the semiconductor device to the chip package. The second element is the chip package itself, which is an advanced 224-lead multilayer ceramic (MLC) surface-mount component. The TAB/MLC combination provides an effective chip package solution for the VAX 6000 Model 400 products. Moreover, TAB/MLC serves as a technology springboard for further TAB-based packaging solutions currently in development for future CMOS systems.

A fundamental decision was made early in the TAB technology program. To reduce risk, we wanted to introduce TAB in an environment that allowed a convenient wire-bonded backup to be developed in parallel. The TAB/MLC solution proved ideal for this approach, and a set of wire-bonded packages was designed and qualified. Subsequently, several of the VAX 6000 Model 400 chips were moved to the backup wire-bonded packages for manufacturing logistics and line-loading reasons.

Wire-bonded technology is not described in this paper because the package designs are very similar to the TAB/MLC packages detailed below.

Tape-automated Bonding Technology

The connections from a semiconductor chip to the package have traditionally been made with wire-bonding technology. In this process, free-floating wires are placed individually from the aluminum chip termination pads to the package internal connections. Although this technology is versatile, it has some limitations in high pin count, high-density applications. Bonding becomes very time-consuming at high pin counts; package and manufacturing tolerances become critical; and package choices with wire bonding have increasingly narrowed to newer and more expensive technologies, such as precision thin films.

From 1980 to 1981, exploratory work was performed within Digital on alternative interconnection technologies. In 1982, the decision was made to pursue TAB technology. The rationale for this decision included TAB's easy testability, compatibility with a wide variety of packaging formats (both single chip and multichip), and improved pin density. The technology also had the potential for electrical enhancements with multiconductor TAB tape, and indications were that the industry as a whole would embrace TAB technology.

Unlike the wire-bonding process, TAB utilizes photolithographically defined copper conductors stabilized by a dielectric film made of polyamide.

As shown in Figure 1, the TAB tape consists of an inner set of connections for the chip-to-tape or inner lead bonds (ILB). External to the dielectric material is the outer lead bond (OLB) region where the tape-to-package connection is made. The leads then fan out to a set of test pads. These pads permit full electrical testing of the semiconductor device at this level of assembly. Sprocket holes serve as handling and alignment features.

Unlike wire bonding, in which the wires may be bonded directly to the aluminized bonding pad on the chip, the planar TAB tape requires a raised pillar, or bump, on each bond pad. The bump, usually made of gold, is typically 25 microns high. In addition to acting as a standoff, the bump provides a surface that is appropriate for the bonding of the gold-plated tape. Figure 2 shows a bumped and bonded device. The bump process may be thought of as an extension to the wafer fabrication process in which the construction of the bond pads is continued vertically.

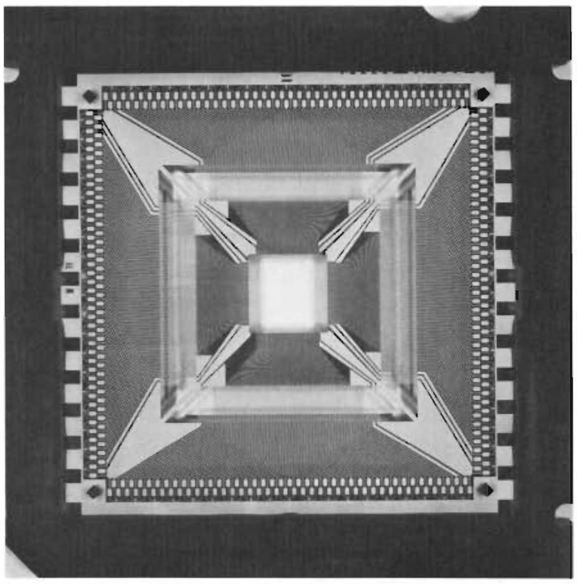


Figure 1 TAB Tape Configuration

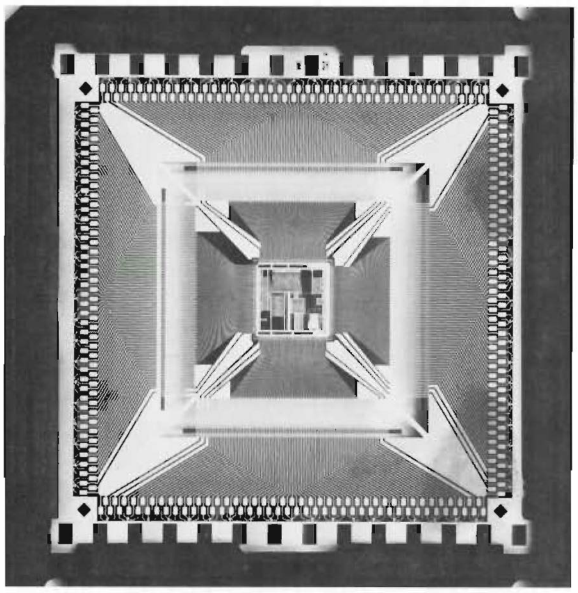


Figure 2 TAB Bumped and Bonded Device

As shown in Figure 3, after the semiconductor wafer has completed the standard diffusion, metallization, and passivation steps, the wafer is metallized again over the entire surface with a sputtered barrier metallization of titanium/tungsten. In the same process step, a top seed layer of gold is applied. The total thickness of these films is 10,000 angstroms. The barrier prevents diffusion between the aluminum pad and the gold bump. Such diffusion would, with time and temperature, seriously degrade the mechanical strength of the interface. The seed layer forms the base upon which the bump will be plated during a subsequent step.

Metallization is followed by the deposition of a thick layer, typically 30 microns of photo resist. This layer is then photolithographically patterned with a bump mask. Subsequent etching opens a hole over the bond pads down to the barrier and seed layer. At this point, the wafer is electroplated with pure gold. The gold is deposited only in the resist openings. Finally, the resist is stripped and a series of etches are performed to remove the seed and barrier metallizations from the areas between the bumps.

After the wafer processing steps are completed, the devices may be electrically tested by probing the bumps. Assembly of the functional die consists of sawing the wafer, selecting the good devices, and bonding the bumped chips to the TAB tape. The bonding process is very similar to wire bonding. Once the TAB tape has been aligned with the bumps, a tool supplies pressure and ultrasonic energy to the interface and creates a strong gold-gold metallurgical bond.

In 1985, the VAX 6000 Model 400 processor was identified as the first product that would incorporate TAB technology. Consequently, the TAB program's goals were set to meet the product's needs:

- CMOS-2 (1.5-micron technology) compatibility
- 150-micron pad pitch
- 224 pins
- 35-mm format TAB tape
- Assembly in ceramic package

The program was designated PTAB 1, the first implementation of TAB with chip pads in a perimeter or single row format. The PTAB 1 process was formally qualified in March 1989.

VAX 6000 Model 400 Multilayer Ceramic Packaging

The custom semiconductor packages implemented for SCIT's ZMOS- and CMOS-1-based systems have been either cofired multilayer ceramic (MLC) through-hole pin grid array (PGA) packages, or the simpler lead-frame-based cerquad surface-mount packages. The maximum pin counts in use up through CMOS-1 are 132 pins for PGA style packages and 164 pins for fine-pitch cerquads.

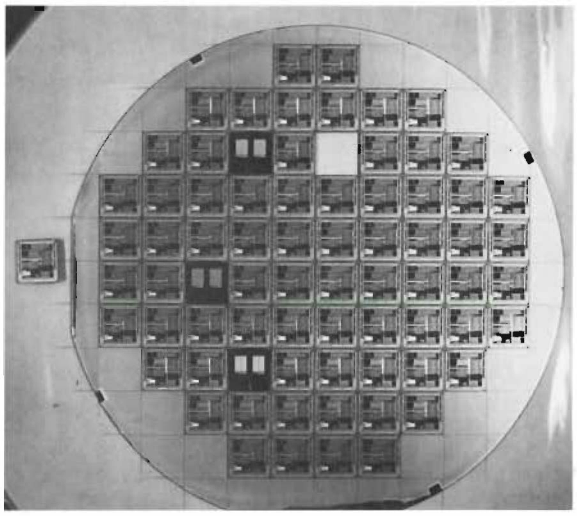


Figure 3 Semiconductor Wafer

The VAX 6000 Model 400 P, F, VC, XDP, and XCA chips require a packaging solution consistent with their electrical and thermal performance needs as well as compatibility with a lead count of 224 pins.

The solution chosen for these components was a blend of the PGA and cerquad technologies: a multilayer ceramic body with 25-mil pitch perimeter leads for surface mounting. The resulting 224-pin MLC is depicted in Figure 4.

There are several significant features in the 224 MLC family of packages. In wire-bond packages, maximum wire span constraints require a different layout for each chip configuration. The TAB format, however, has more routing flexibility and permits one basic package layout. The package has five internal interconnect layers with assignments, as shown in Figure 5.

In cofired MLC technology, the conductor traces are made with screen-printed, tungsten-filled vias. To specialize the package interconnect for each chip, a programmable approach was developed in which only one via layer would have to be changed for each package design. The combination of TAB interconnect and the programmable package concept considerably reduces design time, design complexity, tooling costs, and lead times.

Another feature of the 224 MLC is the provision for eight chip capacitors for power decoupling. The heat-sink design for the 224-pin MLC package is shown in Figure 6. The thermal performance of the package is shown in Figure 7.

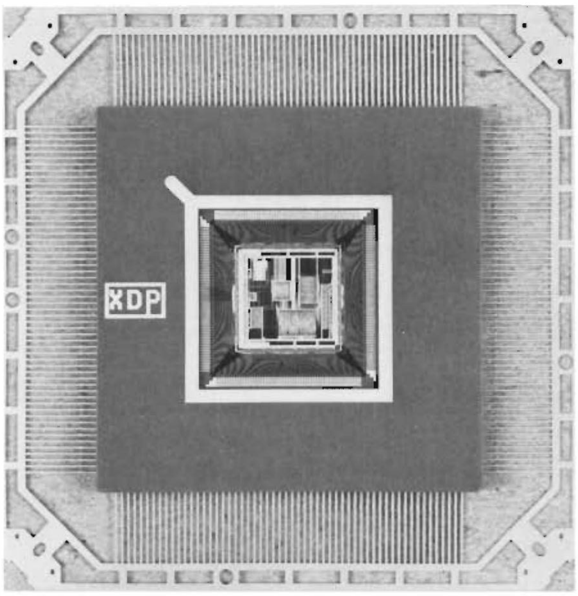
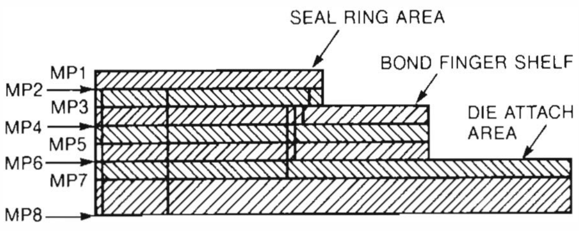


Figure 4 The 224-pin MLC



KEY:

MP1 - SEAL RING AND LOGO
MP2 - VSS EXTERNAL (GROUND)
MP3 - SIGNAL LAYER AND BOND SHELF
MP4 - VDD EXTERNAL (POWER)
MP5 - SPACER LAYER
MP6 - VSS INTERNAL (GROUND), DIE ATTACH AREA
MP7 - VDD INTERNAL (POWER)
MP8 - CHIP CAPACITOR SOLDER LANDS, LEAD BRAZE PADS, HEAT SINK ATTACH AREA

Figure 5 Package Itself with Internal Interconnect Layers with Assignment

The 224 MLC package is assembled by mounting the TAB chip into the package cavity with silver-filled die attach epoxy. Outer lead bonding of the TAB film to the gold-plated package pads is accomplished with the same pressure or ultrasonic process used for inner lead bonding. Final steps consist of lid seal, lead plating, chip cap and heat-sink attach, and lead trim. The part is then ready for electrical testing.

Module Technology

Printed Wiring Board Technology

The VAX 6000 Model 400 processor's printed wiring board (PWB) requirements offered a significant challenge to board fabricators. The primary difference between the Model 400 processor's boards and previous boards is 10-mil finished vias, resulting in a 7 to 1 aspect ratio. We were initially very concerned whether our vendors could produce boards

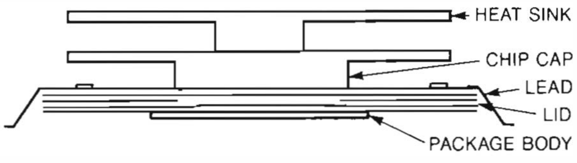


Figure 6 MLC Package Heat Sink

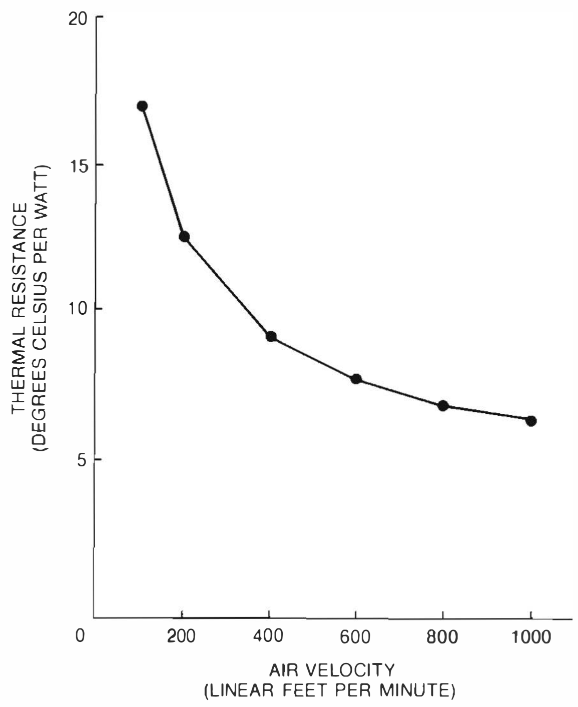


Figure 7 Thermal Performance of Package

that met our specification in the volumes we needed. Table 4 lists some of the primary features of the Model 400 PWB specification.

The most severe environment the boards would see was the assembly plant. Because the module used a variety of component types, it was put through several process steps, each step requiring general or localized heating to solder reflow temperatures. We wanted to ensure that the boards would survive multiple assembly and repair cycles and still be reliable. We established a "fit for use" plan that required the boards to undergo a series of thermal cycles. In these cycles, temperatures and times were set to match intended assembly process steps. The boards were then cross-sectioned and examined for defects. Once manufacturing began building modules, a few of these were also cross-sectioned. With this strategy, we could quickly determine the quality of each lot of boards. Moreover, we could begin to correlate board structure with board quality in each lot. This process allowed us to assess each vendor's capability to provide boards that would meet our specifications.

Because module assembly process temperatures typically exceed 200 degrees Celsius, questions arose over what was the appropriate board material. The more commonly used FR4 material, with an average glass transition temperature (tg) of 120 degrees Celsius, was suspected to be incapable of surviving the assembly process.

The alternative to FR4 was polyamide, which has a tg of 240 degrees Celsius. In response to the concern over FR4, a PWB material selection task force was convened, which was composed of board experts from throughout the company.

The task force discovered that board material actually is not the primary consideration. Either FR4 or polyamide is acceptable (both were eventually used in production). However, other board parameters become critical when FR4 is used. Primarily, minimum barrel copper plating thickness should be one mil for FR4 boards.

Variation in barrel copper should not exceed 50 percent. In addition, there can be no smearing or

Table 4 VAX 6000 Model 400 Printed Wiring Board Specifications

Parameter	Value
General:	
Board size	9.2 inch × 11.0 inch
Layer count	10 layers: 4 signal, 4 power/ground
Material options	FR4, polyamide
Copper foil type	Class III
Testing	100% data-driven
Electrical:	
Characteristic impedance	50 ohms ± 10% for 10-mil lines
DC resistance	4 ohms maximum
Physical:	
Via type	Through
Via size	10 mil finished 13.5 mil drilled
Maximum aspect ratio	7:1
SMT pads	50-mil pitch: 0.030 inch × 0.076 inch ± 0.001 inch 25-mil pitch: 0.016 inch × 0.076 inch ± 0.001 inch
Etch widths	Signal: 5 mils Clock: 10 mils
Proximity to next feature	5 mils
Solder requirement	0.15 mil minimum
Tin/lead alloy	63/37 ± 10%

other defects in the vias from the drilling process. To maintain tight control over these parameters, we included statistical process control as a vendor requirement. Table 5 summarizes the results of the task force findings.

Because of recent board quality problems and the aggressive nature of the Model 400 specification, we ran a product-specific board qualification. The goal was to verify that each vendor could consistently produce the Model 400 boards in volume before the vendor was placed on the qualified vendor list. The four key components of the plan were as follows:

- Incoming inspection
- Electrical test
- Cross-section analysis
- Assembly verification

Incoming inspection testing was nondestructive and covered plating thickness, plating composition, and characteristic impedance.

Using an internally developed tester, we could verify 100 percent connectivity between pads and, thus, detect both shorts and opens. We also used this test to verify prototype boards, ensuring that the boards were good before valuable prototype parts were committed to them.

Digital's Component Evaluation Laboratory performed bare board and assembled module cross-sectioning. This testing process uncovered inadequate or nonuniform plating, pad-to-copper-plating separations and misregistration. The information gained from this procedure was given to the vendor as a basis for corrective action on future board production.

Assembly verification was final proof that the board would make a reliable module. Each board went through the full assembly and test process, including burn-in, to ensure it could survive the process and pass all functional tests.

Table 5 VAX 6000 Model 400 PWB Materials Task Force Summary

Laminate:
Polyamide should be used for prototypes.
Polyamide and FR4 are acceptable for volume.
Long-term, vendor capability and cost may favor polyamide.
Copper:
Class 3 foil should be used.
For polyamide, standard plated-through-hole (PTH) copper is acceptable.
For FR4, minimum barrel plating thickness is 1 mil.
Barrel plating thickness variation maximum is 50 percent. Hole quality must be good.
Process Control:
Vendor should provide test coupon cross-sections with each lot.
Vendor should institute statistical process control.
Vendor site should be monitored regularly.

Surface-mount Assembly Technology

The surface-mount assembly technology (SMT3) used for the VAX 6000 Model 400 processor is the latest in a series of electronic assembly technologies developed by Digital since 1985. The SMT3 technology allows double-sided mounting of high lead-count and fine-pitch devices on a printed wiring board, with surface-mounted passives and mixed component styles. The VAX 6000 Model 400 processor uses essentially all of the SMT3 features.

High pin count and fine-pitch devices presented new problems to the surface-mount attach process team. The small, tightly spaced leads require a smaller pad and less solder than their 50-mil pitch predecessors. The smaller, more fragile pins can become misaligned and no longer coplanar where they meet the board surface.

Our primary goal was to find the correct pad size and solder volume for attaching 25-mil pitch components. Once the fine-pitch pad size and solder volume were determined, it became apparent that the correct solder amount could not be delivered to both fine-pitch and standard pads in a single operation using the standard solder-paste screen approach. The smaller, fine-pitch screen openings could not consistently pass the correct amount of solder. The solution was to use a laminated stepped stencil that places a thinner solder deposit on the fine-pitch pads than on the rest of the pads.

The initial approach for attaching fine-pitch devices was to use the existing vapor-phase mass-reflow process. However, if pin noncoplanarity exceeded 2 mils, some pins would not be soldered adequately. The best that could be guaranteed was 4 mils.

The development team turned to solder-in-place, which uses a thermode fixture to place the component, push the pins into the paste, and reflow the solder, forming a good solder connection for each pin. An advantage of this process is that it does not heat the entire board, and the previously attached components, to solder-reflow temperatures.

The surface-mount module process was by necessity developed concurrently with the Model 400 product design. As the design progressed, a problem with the primary assembly equipment developed. Suppliers for high-volume thermode pick, place, and solder equipment did not keep pace with our schedule requirements. We were then faced with the choice to develop equipment internally or switch the process to a more developed technology.

The Midrange System Manufacturing Group's most readily available backup process was vapor-phase mass-reflow. This process guards against coplanarity problems by including careful inspection of all fine-pitch components before committing them to a board. The pins that do not solder are manually repaired. Since it was not possible to develop solder-in-place equipment internally in time to meet our schedule, the program decided to use vapor-phase mass-reflow.

Module Design

Several board design requirements combined to make the design task challenging. These requirements included signal integrity constraints, finer layout and routing grids, and short dispersion etch. Figures 8 and 9 show the complete module. Most of the module area is composed of the six large 224-pin devices that form the processor core and interface to the XMI corner. These devices had critical placement requirements. Their 25-mil pin pitch forced very dense etch runs.

To attain maximum use of available routing area, the design used tenth-mil grid instead of the previously used one-mil grid. The new grid allowed optimum etch channel placement. Similarly, vias were placed on a 25-mil grid with 50-mil spacing. This allowed the designer more flexibility in placing vias so the available space was used most effectively.

The VAX 6000 Model 400 processor introduced new routing parameters and stringent signal integrity constraints. The density of the design required 10-mil vias. Signal integrity considerations

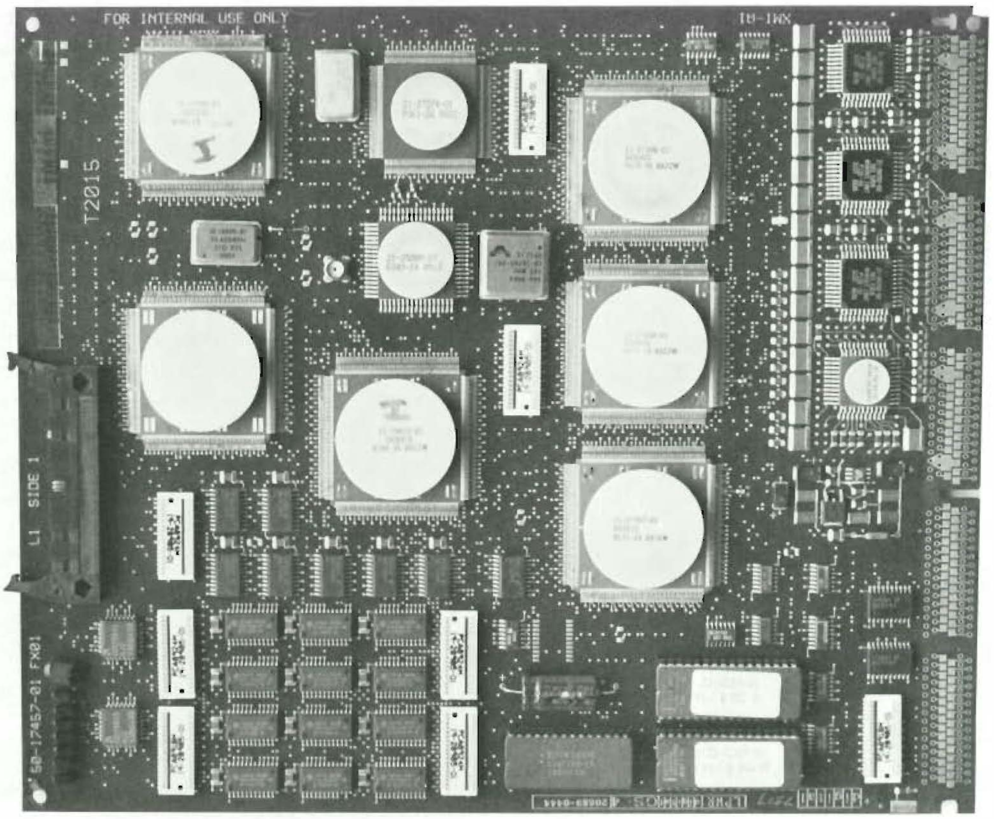


Figure 8 Processor Module — Side One

imposed component spacing and electrical connection requirements that resulted in components being closer than suggested by current standards. Surface etch was needed to keep connections short. Components on side two were placed underneath fine-pitch devices on side one for electrical proximity. New manufacturability rules were generated to cover these situations as the design progressed.

Because surface layers are not used to route signals, these layers were not designed to have good signal integrity characteristics. However, a signal has to travel a short distance from its pin pad to its dispersion via which connects to an inner routing layer. Therefore, it is essential to keep surface etch as short as possible to minimize the distance the signal travels outside a controlled impedance environment. To meet this distance requirement, each critical component was placed very precisely and its dispersion pattern was individually designed.

Signal integrity considerations placed other constraints on the design. Critical signals had to be kept less than a specified maximum to meet performance specifications. To avoid skew problems, clocks were routed equal lengths to within a tight tolerance. Table 6 details the resulting design parameters.

Test Technology

Assembly and Test Process Development

Assembly and test process issues were tracked throughout the development and selection of the Model 400 physical technology. The manufacturing impact of each physical technology choice was quantified in a spreadsheet analysis of cost and quality metrics.

These metrics were estimated by using integrated assembly and test process models for each possible physical technology implementation. Two issues became evident during the physical technology selection process.

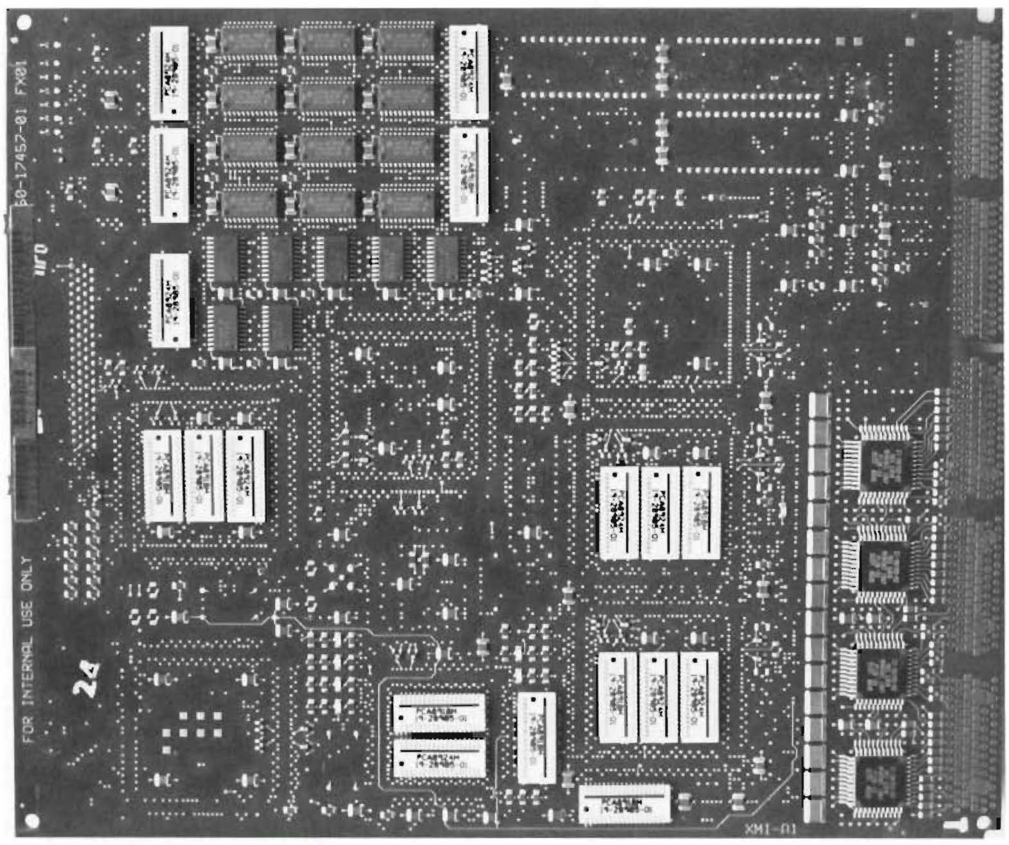


Figure 9 Processor Module — Side Two

Table 6 VAX 6000 Model 400 Processor Board Design Statistics

Parameter	Value
General:	
Board size (inches)	9.2 × 11.0
Board thickness (inches)	.093
Layers	10
Initial route area (square inches)	99
Routing vias	1699
Dispersion vias	3813
Total components	623
Total component pins	4976
Total used component pins	4218
Total networks	1005
Total etch length (inches)	5002

First, the high number of signals and the fine pitch of those signals in all of the possible product implementations significantly increased the risk of manufacturing defects, such as shorts between signals or open faults along a signal. A continuity transistor structure (CTS) is designed into each of the VLSI (very large scale integration) devices to help test and diagnose these open faults.

Second, high-speed operation and reduced physical access would make diagnosis of processor failures difficult. To alleviate this problem, test features and a test system were developed. The test feature was a form of boundary scan, called observe boundary scan (OBS). The test system, the VAX 6000 Model 400 scan monitor, utilized the OBS in a system test environment.

Continuity Transistor Structure

Because the devices are surface-mounted to the module, a large percentage of the manufacturing defects were expected to be open faults between the module and the chips. Typically, these open faults are difficult to detect and diagnose because they usually require the development of a set of complex test vectors that will be applied to the chip. To simplify the test for open faults, a continuity transistor structure (CTS) is designed into each chip. The CTS tests for open faults by using simple instruments such as voltage sources and current meters on an in-circuit tester (ICT).

The Module 400 CPU module is placed on a bed-of-nails fixture that gives the tester electrical access to at least one point on each internal module signal network. The tester then applies digital stimulus to the internal module and verifies correct contact by reading current flow.

The CTS is shown in Figure 10. Pins 1 through N represent all signal pins on the chip. The design uses minimum-sized transistors. The CTS design does not require any dedicated pins because the test pin is a normal device signal pin. There is no performance penalty because the transistors are placed in parallel to the normal system logic, which results in a negligible load on those signals.

The use of the CTS in the processor module manufacturing process has been very successful. The ICT very quickly isolates open connections to the device pin and differentiates them from device test pattern failures. This process allows the open connection to be repaired rather than replacing the device.

Further, CTS testing allows prototype modules to be fully tested for assembly defects, even if the VLSI in-circuit test patterns are not available. This advantage is possible because designers can fully develop CTS tests without any knowledge of the VLSI device internal structure or function.

Observe Boundary Scan

Modules that pass ICT testing are then tested at a "system like" test station. Self-tests and bootable diagnostics are run, and the VMS system is booted. If a module fails any of these tests, skilled technicians diagnose the failures by attaching logic analyzer probes to the module. Because of the fine-pitch surface-mount devices and the high-speed operation, it is very difficult to attach logic analyzer probes to many nodes on the processor module.

OBS allows the CPU module to be observed as it executes VAX macrocode on board self-test stimulus at the module's full clock rate. This additional observation is used by the scan monitor to help test and diagnose module-level faults in a system test environment in stage-one manufacturing.

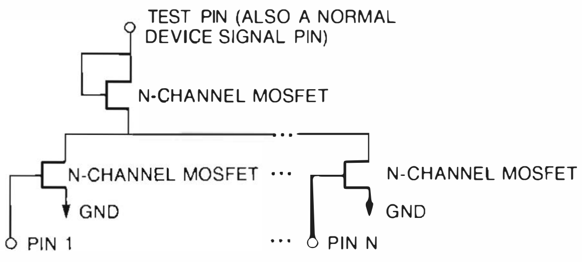


Figure 10 Continuity Transistor Structure

Designing OBS into a custom VLSI device is more complex than adding CTS, but is still relatively simple. OBS is simply a parallel-load, serial-shift register with one bit of the register on each device pin. Although it is not negligible, the OBS uses a relatively small amount of silicon area and also does not affect product performance. The total area used by both the CTS and OBS was estimated at about one to two percent of the chip area. Unlike CTS, which does not require any dedicated device pins, OBS uses two dedicated pins on each device in which it is implemented. To fully utilize the OBS test feature, the VAX 6000 Model 400 scan monitor was designed and built.

The scan monitor controls and reads the OBS on the Model 400 module. A host computer system interfaces with the monitor. The scan monitor control program (SMCP) operates the scan monitor, makes pass/fail decisions on the data received, and diagnoses failures. SMCP also includes many features that allow it to perform as a virtual logic analyzer, including waveform displays that highlight faulty behavior, as well as full triggering functionality.

Conclusion

The aggressive performance goals and advanced semiconductor technology used for the VAX 6000 Model 400 processor meant a significant development effort for packaging and interconnect technology. The technology requirements included high lead count, electrically tailored single-chip packages, very dense controlled impedance printed wiring boards, a state-of-the-art surface-mount assembly process, and advanced test features.

The physical technology achievements in the VAX 6000 Model 400 project represent an effort in the packaging and interconnect disciplines more akin to mainframe and supercomputer developments than to traditional microprocessor-based system approaches. The accomplishments of the efforts include:

- Development and implementation of an advanced TAB technology for the high lead-count custom chips
- Design of an innovative semicustomized ceramic single-chip package that combines the best features of surface-mount devices and traditional pin grid arrays
- Development, sourcing, and qualification of very dense printed wiring boards with multiple controlled impedances
- Achievement of a technology set capable of 28-ns clock cycles through the use of full electrical simulation at the device, package, and module levels
- Development and implementation in manufacturing of the SMT3 module assembly technology, which allows double-sided mounting, high lead-count fine-pitch surface mounting, surface-mounted passive components, and mixed component types
- Introduction of innovative testability features, including the continuity transistor structure for assembly verification and observe boundary scan for diagnosis in engineering debug and module manufacturing

As intended at the outset of the project, these technologies will be employed on a significant number of follow-on midrange and low-end products.

Acknowledgments

A large number of people contributed to the success of the VAX 6000 Model 400 physical technology program. It is not possible to name all these important contributors in this space. Therefore, the authors want to acknowledge that the VAX 6000 Model 400 project could not have been undertaken nor have succeeded without significant efforts by members of the SCIT Physical Technology Group, Semiconductor Assembly Group, Midrange Systems Engineering Group, Corporate PWB Group, Midrange Systems Integration Group, and the SCIT Semiconductor Engineering Group.

Richard E. Calcagni
Will Sherwood

VAX 6000 Model 400 CPU Chip Set Functional Design Verification

The VAX 6000 Model 400 system is Digital's first VLSI CPU to employ a fully micro-pipelined architecture. The CPU chip set for this system posed verification challenges far beyond those of previous designs. The major problem was the large number of complex control sequences and combinations that could exhibit design errors. A single verification strategy would not sufficiently handle this complexity. Therefore, verification engineers developed a multipronged approach for simulation modeling and functional design verification. They also employed CPU diagnostic programs, hand-generated tests, and directed pseudo-random techniques to verify that the design conformed to the VAX architecture. These techniques helped them find bugs prior to committing the design to masks. As a result, the first-pass versions of the CPU chip set successfully booted an operating system. Simulation also minimized chip rework and delays in bringing the product to market.

The Design Verification Project

The VAX 6000 Model 400 chip set verification project had two goals: find implementation bugs in the design and verify that the design performed as a VAX system. The design verification tasks involved about 25 person-years of effort in the areas of system microcode, custom VLSI (very large scale integration) chips, and the VAX 6000 Model 400 scalar processor module.

The chip set verification team coordinated a set of simulation models originally written by the chip logic designers. At various stages of the project, models were available at the gate/transistor, behavioral, and architectural performance levels. The verification team used these models to run a wide assortment of both basic and sophisticated tests. The use of simulation models is described in more detail in the next section.

The CPU design was partitioned into functional units, and one or more units were assigned to each member of the verification team. A list of specific tests or testing activities was produced for each section of the CPU chips' specifications. This list was augmented by project-wide brainstorming sessions. These sessions were used to analyze obscure or subtle combinations of events in the design. Often, the thinking process would identify that a bug existed before any testing had been done. Verification engineers used the list to create tests for the functional units of each chip. These tests were implemented in either microcode, macrocode, pin-stimulus, or some combination of the three. Tests were implemented in the priority determined by the design team's identification of the most complex areas of the design and those most susceptible to bugs.

Simulation Models

Functional design verification using software simulation is inherently slow in a design as large and complex as a VAX CPU.[1] To use resources most efficiently, the verification team specified and coordinated a project-wide modeling methodology that incorporated a number of different modeling levels, trading off detail versus other factors such as speed. These trade-offs allowed us to match the testing of each phase of the design to a model that met its specific needs and characteristics. Thus, simulations were only as detailed as necessary for a particular test situation, and the overall efficiency of the verification effort was increased. Most levels of modeling could contain different levels of description detail for different areas of the design, which further optimized simulation performance.

There were three major phases of the design: architecture, detailed block diagram, and logic/

transistor schematic. To match these design levels, architectural, behavioral, and structural models were written. Each model section was written by its designer and then integrated into a whole-chip model, a system model, or both models for chip and system testing. In addition the design method ensured that the overall organization of each modeling level was faithful to the hardware design level it represented.

Architectural Model

The architectural-level model is the highest level of modeling for this project.[2,3,4] This model describes only the control algorithms and abstract data paths in the microarchitecture. The VAX 6000 Model 400 CPU architects wrote this model in PASCAL for execution performance reasons. The PASCAL program avoids most of the simulation-oriented overhead because it is a standalone program.

In the architectural model, actual microcode is used. However, because much of the detail of the microarchitecture is abstracted in this model, crutches (additional simulation aids) are required to execute the microcode flows. Model simulations are driven from instruction traces. Opcode, operand, and address information are extracted from user programs and system software running on actual VAX systems. Special fields in each microword make use of the information from these traces and direct the flow of microcode execution accordingly. These special fields are only used for simulation and are not included in the actual microcode implemented in hardware. The use of trace data extracted from real VAX systems permits actual machine loads to be reproduced, and the architect can evaluate implementation trade-offs from these reproductions.

Signals in the architectural model are correct to within a machine cycle, which allows execution times to be accurately measured. Microarchitectural parameters, such as cache or translation buffer size, can be easily adjusted to analyze their effects on system performance. More complex design features, such as bus protocols and pipeline control algorithms, can also be modified relatively easily.

The model describes most of the hardware sections that will be in the final design. Therefore, it serves as a prototype system debugging tool to predict and tune system performance. The model also uncovers design flaws before implementation begins. About a hundred bugs were found at this preliminary stage by using the architectural model.

Behavioral Model

The behavioral level model describes each chip section's logic in detail. It is written in the DECSIM behavioral modeling language.[5,6,7] DECSIM provides a high-level computer-hardware description language that executes procedurally rather than using event-driven algorithms. Models written in such a language generally simplify both model and design debugging.

The modeling methodology requires that, in the behavioral model, every signal in the design be explicitly modeled, with its timing accurate to the clock-phase boundary. This methodology maximizes the probability that timing problems will be found at this level of simulation. (Note: No additional logic timing verification was done on this project.) Each designer writes the section's model in parallel with writing the chip's specification. The writing must be done before detailed schematics are started. This method ensures that the model accurately represents the real hardware behavior.

The model executes hierarchically. The system clocks are advanced at the beginning of each phase, then the top-level routines for each section are called. These routines, in turn, call the routines for each subsection. The subsection routines do the actual work.

The DECSIM behavioral model is the basis for many model variations. These variations range from the simplest single-CPU module to elaborate multiprocessor versions that include peripherals and I/O adapters.

The basic version of the behavioral model is nonported; i.e., the model is implemented as a single, self-contained hierarchy of procedures that does not use or connect to any other models. The nonported model contains detailed descriptions of the three custom VLSI chips: CPU processor, secondary cache controller, and floating point accelerator. This model also includes a representation for backup cache RAMs connected to the DAL (data and address lines). Also modeled are a system support chip and a simple memory that can return data as fast as the protocol allows.

This nonported model was used for extensive testing of the microcode, microarchitecture, and logic design of the VLSI chips and their interfaces. A majority of the verification tests did not need detailed memory timing or I/O. Tests could be run faster on the nonported model than on one that simulated the actual memory access delays. These delays would have increased the testing time without adding value to the testing process.

Another version of the behavioral model is ported. This model was constructed to test the DAL and the interaction of the core chips with the system support chip (RSSC) and bus interface standard cell (REXMI) chips.[8] Ported models have ports at the boundaries of procedural model components. Ports are used where chip pins or similar boundaries appear in the design. Ported model components are integrated for simulation through a structural wire list derived from module (printed wiring board) schematics. The main ported model consists of the core CPU chips with a ported representation of the DAL. This model connects to ported models of the other chips on the VAX 6000 Model 400 module, which in turn connect to memory and I/O models. Several different combinations of these ported models were used for various specific verification test applications as shown in Table 1. The combined ported models run several times slower than the less complex nonported model. The slower time is offset by increased testing granularity. The ported models also allow asynchronous behavior, which in turn allows chip- and module-level interactions to be tested.

Table 1 lists more usages of the ported core chip set behavior model as compared to the nonported architectural performance model.

Structural Models

The structural models were derived automatically from the designers' transistor-level schematics. The wirelists, or network descriptions, were translated for the two simulator systems: DECSIM MOS and the ZYCAD simulation engine.[9,10,11] DECSIM MOS is a transistor-level simulator based on RSIM and ESIM that models R-C delays, undefined-state initialization, and charge sharing.[12,13,14] The hardware-accelerated ZYCAD system abstracts transistors into a three-state, gate-level model. The DECSIM MOS model was used for standalone chip sections and whole chip simulations to find initialization and charge-sharing bugs. Both DECSIM MOS and ZYCAD were used to find logic and schematic bugs. Both systems used pin-level pattern stimulus that was

Table 1 Combinations of Behavioral Model Configurations

Ported	Number of Modules	Module Abstraction	Number and Level of Memory	Peripherals	Applications	Performance (Microcycles/ Second)
No	1	Architectural level; per-cycle detail	1 Abstract memory module	None	Architecture debugging and performance tuning	600
No	1	Behavioral level for support chips; per-phase detail	1 Abstract memory module	None	CPU verification and generating chip test patterns	7
Yes	1	Behavioral level for support chips	1 Detailed memory module	None	Self-test code debugging and bus interface verification	2.5
Yes	2	Behavioral level for support chips	2 Detailed memory modules	None	Multiprocessor verification	1
Yes	1	Behavioral support chips	1 Detailed memory module with gate-level bus interface	1 RL02 disk (high level)	Booting VMS system	2
Yes	1	Gate level for bus interface chips	1 Detailed memory module with gate-level bus interface	None	Bus interface verification and generating chip test patterns	1
Yes	1	Gate level for bus interface chips	1 Detailed memory module with gate-level bus interface	None	Module verification	1
Yes	2	Gate level for bus interface chips	2 Detailed memory modules with gate-level bus interface	Bus adapter	System verification in multiprocessor mode	0.5

generated from the nonported behavioral model. Signals traced in the behavioral model matched the boundary of the section of logic or chip being simulated at the gate level. Test results were compared on a cycle-by-cycle basis. The tests uncovered many bugs in the logic design implementation.

Gate-level Fault Simulation

In addition to ZYCAD true-value simulation, single stuck-at fault simulation was done. Fault simulation measured verification and manufacturing test coverage, and provided guidance for verification engineers to enhance tests. The fault simulation effort for the CPU processor chip alone was almost six months long. As a result of this effort, five new tests were written, and manufacturing fault coverage was subsequently increased from 83 percent to 94 percent.

Verification Strategies

CPU chip set verification engineers had two explicitly stated and somewhat overlapping goals. We had to prove that the hardware design intent adhered to the VAX architecture standard in every respect, and that the logic implementation adhered to the intent.[15] We strongly believed that any bugs in prototype hardware (first-pass silicon for the custom VLSI chips) would negatively impact our ability to meet time-to-market for the product. Bugs found at a later stage of the design process are more expensive to fix for custom VLSI chips. It is expensive because we are severely restricted in our ability to isolate and work around bugs in the hardware. Therefore, for custom VLSI chips, verification explicitly meant proving the design and finding the bugs in simulation. No single verification strategy or technique can find all of the bugs in something as complex as a VAX CPU. Therefore, a breadth of verification strategies were flexibly applied.

In addition to technical strategies, the verification team cultivated a "bugs are good" philosophy throughout the project.[16] Past experience has shown us that bugs will always creep into the design of something as complex as a VAX CPU. Instead of being viewed as mistakes or failures, bugs were celebrated because a bug found in simulation was a bug that didn't make it into prototype hardware. This subtle shift in how the finding of a bug was regarded had, we believe, a strong motivational impact on members of the design and verification teams and increased the probability of finding bugs during verification.

Existing Design Verification Tests

The VAX architecture has undergone a number of implementations since the first VAX-11/780 system was designed in the mid-1970s. Over the years, a substantial body of knowledge regarding the key areas and problems associated with designing a VAX processor has been accumulated from various VAX implementations. We put these past lessons to use in the VAX 6000 Model 400 verification effort. We actively sought out bug lists, test plans, and actual test code used by previous VAX system design teams. One key example of this is HCORE, a self-checking VAX macrocoded diagnostic program. HCORE specifically focuses on the high-risk areas that are common across VAX designs. The HCORE test program was originally developed from another basic field-diagnostic program. It was later modified many times, throughout several projects, to focus on testing potential high-risk instructions and functional areas that had been identified in past designs. Existing design verification tests (DVT) such as this are almost always VAX macrocoded tests. Macrocoded tests transport more easily across implementations than microcoded tests because the microword formats are usually different from implementation to implementation.

We derived three benefits from using existing DVTs. They provided a strong level of confidence in the basic functional operation of the design. Second, they found any functional bugs that might be hiding in obscure or seldom used areas of the VAX instruction set. Third, when used with demons (explained later), they were useful in finding bugs in very implementation-specific areas, such as error recovery logic.

Custom Design Verification Tests

Although all VAX CPU designs implement the same architecture and run the same software, the hardware and firmware implementation details of each are unique. Therefore, generic VAX diagnostic tests did not necessarily cover the specific critical paths and functions in the VAX 6000 Model 400 design. Existing DVTs often could not provide a clear picture of what had and had not been covered. To solve these problems, we used custom DVTs to test specific, obscure, and hard-to-get-at areas of the design. There were several techniques for implementing custom DVTs:

- Handwritten macrocode
- Handwritten microcode

- Manipulation of pins and internal signals under simulation control

These techniques could be used individually and in combination, within a single custom DVT. For example, although custom DVTs for the instruction fetch and parse logic (I-box) were written primarily in microcode, a custom macrocoded instruction stream was written to give the I-box something to parse. Explicit manipulation of pins in simulation was used to generate asynchronous events, such as interrupts, when necessary.

Custom DVTs provided confidence in areas of the design that could not easily be tested with existing DVTs. We made the tests as focused and efficient as possible. However, in doing so, the generation of such tests required large amounts of development time and people resources. Although these tests uncovered several bugs in all areas of the design, we now believe that many of these same bugs could have been found with less labor-intensive methods, such as pseudo-random tests. At the time, the primary advantage of custom DVTs was the clear indication they gave that specific functional areas of the design had been tested and were working as specified.

Pseudo-random Design Verification Tests

Each new VAX CPU design aspires to improve on the price or performance of the previous design. Improvements are sought by pushing the limits of available technology to package hardware into smaller and, if possible, less expensive spaces. At the same time, a decrease in the cycle time or an increase in the work done per cycle in the functional design is also sought. In particular, this last item has substantially increased design complexity by introducing techniques such as pipelining and special-case hardware. As a result of this complexity, we often encounter very obscure bugs when debugging new VAX implementations. These bugs involve unanticipated interactions in the logic, between seemingly unrelated functional areas, and interactions dependent on intricate combinations and sequences of events. We were concerned about these types of bugs because it is extremely difficult to write tests for unanticipated problems. The method we chose to address these problems was pseudo-random testing.

The intent of pseudo-random testing is to exercise the design in ways that are likely to find bugs without necessarily knowing in advance what those bugs are or where they might be. Pseudo-random testing implies simulating many cycles and trading off test efficiency to address the problem of unanticipated bugs.

It is absolutely necessary to automate the pseudo-random test process, both test generation and test scoring, as much as possible since pseudo-random tests are much longer than focused tests.

A powerful tool already available for pseudo-random testing VAX designs is the VAX architectural exerciser tool suite (AXE and MAX).[17] Originally intended as hardware prototype verification tools, AXE and MAX have proven to be even more effective as design verification tools in a simulation environment. They provide a virtually inexhaustible source of unique, interesting macrocode test cases, and require a minimum of intervention and effort by the user.

Although AXE and MAX provide some control over test case parameters, they still aspire to be general, architecturally focused exercisers. We also wanted pseudo-random test case generation that could be targeted at specific, risky areas of the implementation. These areas were the most likely locations for unanticipated bugs. Custom pseudo-random exercisers were developed for these areas. These exercisers provided very detailed control over test case parameters, yet retained many of the features and advantages of AXE and MAX.

A powerful technique for pseudo-random test is the use of demons.[16] A demon is any automated intervention of a simulation model's normal execution behavior. For example, a bus demon can interject one or more bus commands, error conditions, or interrupts at random intervals in order to aggravate normal system operation. By doing this, a dense environment of unusual or uncommon event combinations can be created to stimulate the design with worst-case situations. Demons typically slowed model execution by a factor of ten, but they often found bugs that had not been considered by the designers or architects. Without demons, it could have taken many months of field testing to find and characterize these bugs, if they would have been found at all.

Pseudo-random testing with AXE, MAX, custom exercisers, and demons was used throughout the development cycle of the CPU chip set. All four uncovered obscure interaction bugs, as expected. Unfortunately, they did not find them all. Some unanticipated bugs slipped through verification and into the first-pass silicon stage. These bugs were eventually found after running many more cycles in real prototype hardware. The failure to find all unanticipated bugs in the simulation stage illustrates a fundamental problem in the use of pseudo-random testing. The effectiveness of testing is closely coupled to the number of cycles run, and simulation speed severely restricts this number.

Our application of pseudo-random testing to the problem of unanticipated bugs was largely successful for this project. However, we learned that we must do more in the future to increase the efficiency and scope of these tests. To provide this increase, we are looking toward more directed pseudo-random testing.

Booting the VMS Operating System

A major milestone in the development of any new VAX CPU is booting the VMS operating system on prototype hardware. Not only does this demonstrate significant functionality in the design, but it also provides a platform from which further testing can proceed. As previously stated, increased complexity in these designs can produce very subtle bugs. Often, such bugs do not even appear until the hardware is run under a heavy system load in a large multiprocessing or I/O-intensive environment. Successfully booting VMS on prototype hardware is necessary before any such system load testing can begin. The sooner such testing begins, the better the chance of finding subtle bugs. For this reason, booting the VMS system in simulation was an important goal of the VAX 6000 Model 400 chip development and verification effort.

At first glance, it would seem impossible to boot the VMS system on a simulation model in a reasonable amount of time. Simulation speeds on the fastest of our models were many orders of magnitude slower than actual hardware. However, careful analysis of the macrocode modules involved in the boot process revealed that by optimizing or removing large, iterative sections of code, we could substantially reduce the number of cycles in the boot path without losing significant coverage for verification. For example, the primary bootstrap module contains code that creates a bit map of all physical memory in the system. The code tests each memory location for errors. By reducing this code to test fewer memory locations, the number of cycles executed is vastly reduced.

Another key optimization involved speeding up simulated transfers from disk. At several points during the boot process, code and data are pulled into memory from a mass storage device (usually a disk). For simulation, a special "turbo" disk model was written by the verification team. This model artificially processes model requests for block data transfers and performs the transfers instantly. This technique eliminated wasting simulated CPU cycles while waiting for transfers to complete.

With model and code optimizations in place, booting the VMS system to the point of printing the VMS banner and starting the process scheduler was actually achieved in simulation. It took approximately seven CPU days on a host VAX 8800 system. One model bug and one real design bug were found during this effort. Booting the VMS system in simulation required a large amount of verification resources. The process was worthwhile in terms of the bugs that were found and the confidence it gave us that we could boot VMS on first-pass hardware.

Analysis of the Functional Bugs Found

To demonstrate where the verification project was successful and where it needed improvement, we discuss here the bugs that were found both before and after the chip set first-pass design was committed to masks. (Note: The milestone of this commitment is called PG, for mask data preparation's pattern generation.) Only the bugs pertaining to architecture, microcode, functional design, and logic design are discussed. Layout and circuit problems, as well as modeling and tool bugs, are outside the scope of this paper.

During the design period, several hundred bugs, with a variety of complexity levels, were found in all sections of the design. These bugs were found through the verification techniques described in previous sections and are detailed in Table 2.

The prototype chips were first tested on a Takeda 3381 chip tester. This tester allowed prototype chips to be tested in a standalone environment. The chips were then inserted into a prototype CPU module, which was part of a custom-designed engineering tester. The prototype module provided a

Table 2 Methods Used to Find Design Bugs before First-pass Silicon

Verification Process	Number of Bugs Found
Custom DVTs, DVT reviews, microcode assertions	273
Existing macrocode test programs	171
Pseudo-random macrocode tests	90
Boot VMS operating system on the behavioral model	1
Total of bugs found prior to first-pass silicon	535

true system environment for the chip set.[18] During prototype debugging, 11 bugs were found. The characteristics of these bugs are shown in Table 3.

None of these bugs was a "show-stopper" in terms of prototype debugging or field testing. In fact, most of them were so obscure that the probability they would appear during normal use was very low. Several generalizations and conclusions can be drawn from the types of bugs found and how they were found, particularly whether they could have been found earlier.

- Bugs 1, 2, 6, and 9 were simple, but were missed because of overlooked test coverage. The simplest test, if identified and written, would have found them. We learned that we needed more discipline and thoroughness in generating lists of tests as guided by test coverage indicators.
- Bugs 4, 5, and 11 were found on the simulator after prototypes were available. It is questionable whether these bugs would ever have been noticed, much less isolated, on shipped systems because the conditions triggering them were obscure. For example, six conditions, including a parity error interrupt, had to happen simultaneously to reveal bug 4. It was acceptable to find these bugs after PG because the impact from such bugs on prototype debugging was minimal.
- Bug 3 was found using prototype hardware rather than simulation. A new version of MAX was released during the prototype debugging time frame. This version had some new test coverage features, and we decided to run them on the hardware at-speed as opposed to using simulation. One of the new features stimulated the conditions for bug 3. Had this MAX feature been available prior to PG, we would have encountered the bug in simulation within the first few tests. Typically, 90 percent of the bugs uncovered by using the MAX tool are found with the first 100 tests generated.
- The verification methodology for bug 10 was correct, but had not been followed. Appropriate gate-level comparison testing would have found this bug prior to PG.
- Bugs 7 and 8 were extremely obscure and difficult to find. The first symptom of these bugs was a series of unexplained system crashes over a period of several weeks. The situation was finally resolved by attaching a logic analyzer to a system and waiting for days for the right combination of events to trigger the failure. Unfortunately, it is highly unlikely that we would have found either bug in simulation, even with a broader scope of directed-random verification, given the limited amount of simulation that was done. The architectural, microcode, and pipeline combinations required to trigger these bugs were just too complex.

Conclusions

Although there were 11 bugs in the first-pass chip set, the verification efforts were considered quite successful. Prototype debugging was never stopped because of a bug, and system field testing was able

Table 3 Bugs Found after Pattern Generation

Bug Number	Chip Type	Bug Type	Complexity	How Found
1	CPU	Microcode	First order	Inspection
2	CPU	Microcode	First order	Inspection
3	CPU	Logic	Sequential	MAX on prototype
4	CPU	Microcode	Second order	Pseudo-random on simulator
5	CPU	Microcode	Second order	Pseudo-random on simulator
6	CPU	Microcode	First order	Self-test on prototype
7	CPU	Logic/microcode	Second order	Field test system crash
8	CPU	Microcode	Second order	Field test system crash
9	CPU	Logic	First order	Prototype debugging
10	Floating point accelerator	Logic	Sequential	Chip tester comparison
11	Cache controller	Logic	Second order	System verification test on simulator

to complete on schedule. The modeling and verification methodologies contributed to this success.

- All logic, functional, and microcode bugs could be reproduced in the behavioral model. Therefore, the bugs found after PG could have been found with simulation had the appropriate combination of events been tested.
- Complex pipeline activity, an area of concern from the beginning, was the primary problem area in the design. Compromising design complexity to make thorough verification more achievable should be considered.
- In general, the tool or test that first exercised a chip section or functional area with a bug found the bug. We believe this indicates that it is more productive to first debug using existing macrocode tests or automatically generated tests before writing new tests.
- Although the effort to boot the VMS operating system on the simulator was comparatively large, one bug was found that would not have been found through other means. We had not considered the combination of events that caused the bug. Finding this bug clearly showed the benefit of running this application on the simulator. Further, it gave us confidence that the VMS operating system would boot on the first-pass prototype hardware to ensure that prototype debugging could proceed unimpeded.
- The flexible and wide-ranging modeling methodology served the design team well. The source code of the CPU chip set model has been reused in system verification and chip set application development projects at least ten times. The corporate standard DECSIM logic simulator made this modeling effort savings possible.

Acknowledgments

The verification of the CPU chip set design was a team effort performed by the Semiconductor Engineering Group's microprocessor verification team. Members of this team included David Asher, Rick Calcagni, Liza Hudepohl, George Mills, Rick Pekkala, Ed Rocha, Will Sherwood, and Chris Spear. Part of our success is attributed to our CAD tool development groups, including the SEGCAD DECSIM team and the VAX Architecture Group AXE/MAX team. We would also like to thank Mike Uhler for his technical guidance throughout the project.

References

1. J. Basmaji et al., "The Role of Computer-aided Engineering in the Design of the VAX 6200 System," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 47–56.
2. C. Wiecek, "The Simulation of Processor Performance for the VAX 8800 Family," *Digital Technical Journal,* vol. 1, no. 4 (February 1987): 100–110.
3. C. Wiecek and S. Steely, "Performance Simulation as a Tool in Central Processing Unit Design," *Performance Evaluation Review,* vol. 11, no. 1 (August 1979): 41–47.
4. B. Moses and K. DeGregory, "Performance Evaluation of the VAX 6200 Systems," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 64–78.
5. M. Kearney, "DECSIM: A Multilevel Simulation System for Digital Design," *Proceedings of ICCAD* (October 1984).
6. R. Gries and J. Woodward, "Software Tools Used in the Development of a VLSI VAX Microcomputer," *Proceedings of the MICRO-17 Conference* (October 1984).
7. W. Sherwood, "An Interactive Simulation Debugging Interface," *Computer Hardware Description Languages and Their Applications,* Breuer and Hartenstein, editors, (Amsterdam: North-Holland Publishing, 1981): 137–144.
8. P. Sullivan et al., "The VAX 6000 Model 400 Scalar Processor Module," *Digital Technical Journal,* vol. 2, no. 2 (Spring 1990, this issue): 27–35.
9. B. Milne, "Put the Pedal to the Metal with Simulation Accelerators," *Electronic Design,* vol. 35, no. 21 (September 1987): 39–52.
10. M. McMahon, "Accelerators for Faster Logic Simulation: The ZYCAD Approach," *Proceedings of VLSI and Computers, First International Conference on Computer Technology, Systems and Applications* (COMPEURO '87 Cat. No. 87CH2417-4, 1987): 981.
11. D. Jenkins and S. Morton, "Transistor-Level Logic Simulation Using the ZYCAD Logic Evaluator," *Proceedings of the Conference on Automated Design and Engineering for Electronics* (1985): 154–163.

12. C. Terman, "Timing Simulation for Large Digital MOS Circuits," *Computer-aided Design of VLSI Circuits and Systems,* (Greenwich: JAI Press, 1986).
13. C. Terman, "RSIM—A Logic-Level Timing Simulator," *International Conference on Computers and Design* (1983).
14. C. Terman, "Simulation Tools for Digital LSI Design," *Technical Report TR-304,* MIT/LCS (Cambridge: MIT Laboratory for Computer Science, 1983).
15. T. Leonard, ed., *VAX Architecture Reference Manual* (Bedford: Digital Press, Order No. EY-3459E-DP, 1987).
16. D. Clark, "Bugs are Good: A Problem-oriented Approach to the Management of Design Engineering," *Research — Technology Management* (forthcoming 1990).
17. D. Bhandarkar, "Architecture Management for Ensuring Software Compatibility in the VAX Family of Computers," *IEEE Computer* (February 1982): 87–93.
18. W. Sherwood, "The VLSI VAX Chip Set Microarchitecture," *Microarchitecture of VLSI Computers,* P. Antognetti, ed. (1985): 103–126.

John W. Croll
Larry T. Camilli
Anthony J. Vaccaro

Test and Qualification of the VAX 6000 Model 400 System

Computer-aided design simulation, which is used in the design of the VAX 6000 family, finds most problems during the hardware design phase. Simulation, however, cannot test a complex system running under system software control. For the VAX 6000 Model 400 system, a qualification process was designed to completely test the interaction of the system's hardware and software components. The benefit of such a process is clearly shown in the results. Nearly all the problems found in the qualification stage could not have been found in the simulation process. The testing and qualification of the Model 400 was a multigroup effort. This paper describes the methods and tools of three Midrange Systems Engineering groups who were involved in the project.

The VAX 6000 Model 400 system is the third in the VAX 6000 series. The Model 400 is designed to enhance CPU performance using the same platform, that is cabinetry, buses, and power systems, as all models in the VAX 6000 family. The basic architecture of the VAX 6000 Model 400 is unchanged from that of the earlier Model 200 and 300 systems.[1,2,3] The Model 400 is distinguished from earlier models primarily by two additions: a new processor design, which offers over twice the performance of the original VAX 6000 Model 200 processor, and by the addition of a vector coprocessor.[4,5] The interfaces in the Model 400 to the common platform remain the same. However, the processor is an entirely new design, and is the first Digital system to use semiconductors designed with the CMOS-2 process.[6]

All VAX 6000 systems use a common design process that relies heavily upon simulation to detect and correct design errors. This simulation process is designed to ensure that first-pass hardware, or the initial engineering prototypes, will run the operating system at speed.[7] The VAX 6000 Model 400 processor is an excellent example of the effectiveness of simulation techniques. The elapsed time from power-on of the first prototype to reliable login under both the VMS and ULTRIX operating systems was less than six weeks, which is substantially less time than has been seen for previous VAX processors.

Current computer systems are very complex, especially when hardware and software interactions are considered. Simulation cannot adequately test hardware executing under operating system control. A companion system test and qualification process, executed on hardware prototypes, is required to thoroughly test the complete system and ensure its reliability.

A key goal of the VAX 6000 common-platform design strategy was to allow new processor technology to be brought to the market quickly. To achieve this goal, we had to minimize the time required for system test and qualification without compromising the quality or reliability of the final product. By reusing common platform components, we could primarily focus on testing the new components.

This paper addresses the system test and qualification process used for the VAX 6000 Model 400 system. This process was designed to maximize test effectiveness and minimize test time. The first customer shipments of the VAX 6000 Model 400 system occurred only six months after the introduction of its predecessor, the VAX 6000 Model 300 system.

System Test and Qualification Process

The VAX 6000 Model 400 System Integration Group is responsible for the overall design and management of the system test and qualification process. This group comprises engineers who reside within the hardware design group, but have not directly participated in the design of the components being tested. Therefore, any problems found during the test period can be rapidly communicated and resolved, whereas the possibility of "testing to implementation" versus "testing to specification" is avoided.

A distributed test process was developed for the VAX 6000 Model 400 system that used the resources and expertise of a variety of groups within Digital. The test process also allowed many tests to be executed in parallel to minimize time. Additionally, some of these groups have specialized test facilities that are required to satisfy standards imposed by Digital and various government regulatory bodies, e.g., the FCC and UL.

The groups that participated in testing the VAX 6000 Model 400 system, together with a brief description of each group's function, are shown in Table 1. Each group played a valuable role in testing a particular aspect of the system. Initially, a very aggressive test schedule of six months for the entire test and qualification process was planned. However, due to some delays in planned prototype availability, all testing had to be completed in about five months to allow systems to ship as scheduled in July 1989. The expertise of all the groups involved was required to meet this schedule. However, space limitations prohibit an in-depth discussion of each function. Therefore, we will examine in-depth the roles of three of the Midrange Systems Engineering Groups—the VAX 6000 Model 400 System Integration Group; the VAX Architecture Verification Group; and the Midrange Systems Evaluation Engineering Group. We will also describe the tests and tools used in the qualification of the VAX 6000 Model 400 system.

System Integration Group

Planning for the system test and qualification of the VAX 6000 Model 400 system began approximately one year prior to scheduled availability of the first prototype systems. During this period, the overall qualification plan was developed, and individual test plans were solicited from each group that would participate in the testing. Each plan was reviewed by the System Integration Group for test coverage, as well as for minimization of overlaps and duplication in the component plans. In parallel, the System Integration Group developed plans for hardware-specific system design verification tests

Table 1 Organizations Involved in System Qualification

Organization	Function
System Integration	Manage qualification process; perform DVT, system test, reliability confidence test; manage field test
VAX System Architecture	Exercise new system to find discrepancies with the VAX architecture
Midrange Systems Evaluation Engineering	Test a variety of system configurations for proper operation, concentrating on I/O configurations
VAXcluster Validation	Test complex VAXcluster environments with new products to find problems in new or existing VAXcluster components
Mechanical Technology	Demonstrate successful product operation while exposed to specific environmental conditions (e.g., vibration, humidity, altitude)
Electromagnetic Compatibility Engineering	Test new products for electromagnetic compliance to various government regulatory requirements (e.g., FCC, VDE)
Product Safety Laboratory	Test various aspects of new product safety and ensure compliance to safety standards (e.g., UL, CSA)
Diagnostic Quality Assurance	Test various diagnostic programs to ensure correct operation
Manufacturing Product Verification	Provide the necessary data to verify and improve the manufacturing process to enable consistent production of a quality product
Central Characterization Group	Characterize performance of industry and application-specific workloads
System Performance Analysis Group	Model and measure performance of midrange products
Customer Services Systems Engineering	Develop service delivery and training programs, test diagnostic and repair features of products
Software Quality Maintenance	Test software-layered product operability with new hardware and operating system versions
Systems Reliability Engineering	Predict, test, and analyze hardware and system reliability

(DVTs) and for external field test. Prototype plans specified the number, distribution, and cost of the prototyped systems that would be required during the test period.

The System Integration Group also acted as the problem-reporting center. In implementing a distributed test process, two functions are essential. There must be a central focus that disseminates information regarding observed problems to all test groups. Second, an established method for tracking status and resolution of these problems must be maintained. An internally designed system maintained a complete audit history of each problem. There were 121 problems reported during VAX 6000 Model 400 testing. Each step in the resolution process was tracked for each problem in the problem database. This database was available to all test groups. The database was supplemented by weekly cross-functional meetings, at which representatives from engineering, manufacturing, and customer services reviewed and updated each problem's status.

The complex VAX 6000 Model 400 project schedule was developed and tracked by the group. Due to the number of groups involved in testing and the delays involved when problems were found, the critical path tended to be very dynamic during the test phase. A project management tool, which was developed within Digital and which used PERT, tracked status against milestones and modeled different scenarios to prevent overall schedule slippage as changes occurred.

The System Integration Group performed four major types of test. These were system test, reliability confidence test, design verification tests, and field test. The following sections describe each of these tests.

System Test

There are two forms of system testing, directed and random. Most testing groups use directed tests, which test hardware or software features, or follow a strict test sequence. Directed tests seek specific results and are well defined.

The System Integration Group performed many directed tests on the Model 400 system. Some of these tests were done to satisfy the requirements of external regulatory agencies, or internal Digital development standards. Other directed tests include system DVT tests, which are discussed in more detail later in this paper.

Many aspects of complex systems cannot be adequately tested in a directed fashion. For example, an operating system and a processor can operate in a nearly infinite number of states. It is impossible to design a series of tests to verify each of these states.

Random tests exercise the system in more comprehensive ways than directed tests. They do not seek specific results. Instead, random tests attempt to push the system into as many different states as possible, as quickly as possible. Greater test coverage results from these tests, but problem diagnosis and isolation are more difficult.

Because random testing does not look for specific results, it is effective only if done for extended periods of time. Even if identical test scripts are run repeatedly, system activity becomes unpredictable over time, due to events such as network activity or disk fragmentation. This unpredictability is important because it means more system states are being exercised.

The System Integration Group developed a random test package, called the Systems Integration Test Package (SITP), to test the VAX 6000 Model 400 system. This package consists of a comprehensive collection of test programs and a script-driven mechanism that controls their execution. SITP is diverse and flexible. The test programs were obtained from many sources. The System Integration Group also wrote some test programs to exercise specific aspects of the Model 400 system that were not fully exercised by other tests.

The test programs used with SITP are high level. Each high-level test uses many lower level functions within the system. Many of these programs are run together, with varying test parameters and runtimes. The programs are self-checking. If an action does not complete properly, the program notes the error immediately. The program does not attempt to identify the cause of an error; rather, it gathers as much information about the error as possible. This information is later examined by a test engineer.

SITP is easy to use, restarts automatically after system crashes or power failures, and includes monitoring tools. Periodic reports, with details about system activity and error log data, are generated by the test package. With this information, the test engineer can gauge the effectiveness of the tests and adjust them as necessary. The test engineer can also control and monitor tests on many different machines. Machines can be located locally and remotely.

A number of SITP scripts were developed to provide different workloads for testing the Model 400 system. Each set of scripts emphasized a different type of system activity. Some were compute intensive, some I/O intensive, and some stressed parallel and multiprocessing activity. The scripts were modified to suit system configurations as needed.

SITP and the test scripts were installed on all the Model 400 system prototypes in the system integration lab. Tests under the control of SITP were run on the prototypes as prototypes were available. Because the prototypes were heavily used during daytime hours for various debugging tasks, SITP tests were run overnight and on weekends. The test scripts were designed to run for a specific number of hours and then stop. The prototype was then available for the next user. This procedure allowed otherwise idle prototype hours to be used in system testing and ensured a clean shutdown of the tests. In this way, test data could be retrieved without interference from other prototype users.

SITP was used on the earliest Model 400 system prototypes and was continually used throughout the qualification period, as prototype time was available. Scripts were tailored to cause test concentration in specific areas and were modified as necessary to suit various prototype configurations. Typical SITP runs would last for 16 or 19 hours (overnight), or 58 or 60 hours (over weekends). Processor, memory, and I/O configurations varied from run to run, and depended on test needs and equipment availability.

The overall results from system testing were very positive. Over 6700 CPU hours were accumulated on various prototypes and configurations. Many errors were encountered during this period, but most were due to SITP bugs (SITP was still under development for most of this period) or to errors in setting up test scripts. Hardware errors occurred in peripheral devices, principally disks and communications devices, and were corrected as they occurred.

Of the more serious problems found, one was a hardware problem that would cause a system hang. The problem was identified as a bug in a bus interface chip on the CPU module, which was operating in an untested mode. It was resolved by modifying the system console to ensure that this mode was never used. An error was found in the VMS machine check handler, which was corrected in a subsequent release of the VMS operating system.

Five other serious bugs were found in the new CPU modules. Although none of these bugs were found by the System Integration Group's testing, each took time to investigate, resolve, and test the fixes. As a result, there was less time available on prototype machines for other testing. Two of these bugs were fixed by modifications to the CPU module. The other three required changes to the processor chip. As corrected processor chips became available, SITP was used to ensure the fixes had not introduced further bugs.

It is interesting to note that four of these five problems occurred in system areas not simulated during hardware design. Of these four, two occurred in the handling of external system events. One was in system reset handling. The other was in handling "control/P" interrupts. Control/P is the standard method an operator uses to get the attention of the system console on VAX systems.[8] Two bugs were caused by interactions between the new processor and other system components. These interactions were not simulated during hardware design. The fifth bug was not found during simulation because of a deficiency in a simulation test tool.

Reliability Confidence Test

To accumulate uninterrupted run-time on the Model 400 system, five identically configured systems were set up in an isolated area. The machines were isolated to protect them from outside interference while the confidence test was running.

The purpose of this test was to determine the actual reliability parameters of the Model 400 system hardware and to compare the results to the system's actual reliability requirements. A secondary goal of the test was to determine the long-term system reliability, both for the hardware and operating system software.

The duration of the test was planned for six weeks, which was sufficient to show the hardware reliability. Once this six-week period was over, we planned to continue to run the machines in the same environment with the same workload for as long as possible to accumulate further system run-time.

The test started at the beginning of May 1989, when enough CPU modules became available to populate the five machines. The formal test period ended two months later, in late June. Three of the machines continued to run for two and a half months, until mid-September. Of the other two machines, one machine was needed for other purposes, and another's CPU modules were removed to change the configurations of the remaining three.

The systems ran identical SITP scripts that concentrated on exercising the new CPUs. Tests included compute-intensive programs, programs that explicitly tested various aspects of the new

CPUs (e.g., multiprocessor cache coherency), some decomposed parallel applications, and tests that generated many VMS processes.

The overall results of these tests were very good. The systems demonstrated a hardware reliability of over a year between hardware failures.

Only two module failures occurred. One had a bad cache tag store, which was discovered very early in the test. As a result of this discovery, the test process for the CPU module's cache control chip was changed. The other module failure was a floating point chip failure. One specific test program began generating wrong answers late in the formal period of the confidence test. This module was removed for repair.

Of the other failures that occurred during this test, all were attributable either to test script or set-up errors, or to software or hardware problems which were corrected prior to shipment to customers.

Once the formal test period was completed, the three machines that continued to run until mid-September exhibited no new failures and eventually accumulated a year's run-time. The reliability of the VAX 6000 platform and the Model 400 CPU was successfully demonstrated.

Design Verification Tests

Part of the System Integration Group's responsibility is to ensure that parts of the system not covered by tests from other groups are tested. In general, these parts are specific either to the new hardware or to how the new hardware fits into the existing system. These tests are called design verification tests (DVTs).

A complete test plan for the Model 400 system DVTs was written and reviewed by the group. The list of DVTs performed is shown in Table 2.

The Model 400 DVTs complemented those tests performed on the new hardware components. The parts of the system tested were those in which other testing was weak or nonexistent. These tests were conducted in a formal manner, with a written sequence of events and formal reports of results. Any problems found were noted and reported to the relevant development groups for analysis and eventual correction.

The DVTs executed for the Model 400 uncovered two system bugs and some minor documentation problems. Both bugs were related to power failure recovery. One was in the console and one in the VMS operating system. Both bugs were eventually fixed. The minor documentation problems in the system installation guide were also fixed. All other design verification tests found no problems with the system.

Field Test

Field tests are made on prototypes of new products provided to customers. The purpose of field test is to gain experience with the new products before production; new products are actually used as opposed to tested.

Table 2 VAX 6000 Model 400 System Design Verification Tests

Test	Function
Keyswitch	Verifies front panel functions with new hardware/software
Voltage margin	Verifies proper operation over allowable voltage range
Thermal	Verifies proper operation over allowable temperature range
Power-fail/battery backup	Tests battery backup operation with hardware and software
Interlocks	Tests memory interlock functions
XMI saturation	Tests system operation under very heavy bus loads
Queue contention	Tests proper operation under very heavy worst-case memory loads
Multiprocessing	Tests proper operation of all multiprocessing features
Load test	Runs system for extended period under heavy compute and I/O load
System installation	Verifies overall manufacturing process, and installation documentation
Boot	Verifies that system boots all operating systems from all devices
System configuration	Verifies that all allowable configurations work properly
Remote services console	Verifies that remote services console hardware and process works with new system
Console input	Verifies that console terminal hardware and software work properly

For VAX system products, field test historically has lasted a minimum of four months. This period was determined from tracking problem reports from field test sites. Generally, a month was required for the new product to be installed and for usage to reach a level where meaningful testing occurred. The next three months provided useful data about the new system. After three months, the amount of useful data declined.

In field testing the VAX 6000 Model 400 system, we shortened this four-month period to three. The plan was to check on field test results two months into the test. If field test was not progressing well at this point, we were prepared to extend the test period.

The System Integration Group did two things to eliminate the field test startup time. First, because the Model 400 system was an upgrade from earlier VAX 6000 systems, Model 200 systems were shipped to each field test site in advance of the Model 400 field test start. These systems were installed and running approximately a week before the official field test started. A considerable amount of time was saved in site preparation and system installation.

Second, the startup of the field test sites was performed by system integration engineers, who brought Model 400 CPU modules and new VMS software to each site. These engineers supervised the installation of the new CPUs in the previously installed systems, installed the VMS operating system, and ensured that the systems were available to the customer before leaving the site.

A total of seven field test sites were started up in early April 1989. Six of these sites were located in the United States. Five of these sites were installed and turned over to the customers within two days. The sixth site was ready in four days. The seventh site was located in Europe, and was started up by mid-April.

Once the site systems were running, the System Integration Group maintained regular contact with each site. Each site was assigned a "captain" (a system integration engineer), who polled the site weekly, talked to the users, and received first-hand information about machine usage. This method was used instead of the traditional dial-up problem reporting method for two reasons. First, technical problems existed in making reliable connection to the dial-up system. Second, many people are reluctant to report problems, unless the problem is so large that work stops or is severely impaired.

Overall, field test went smoothly. Most of the problems that occurred were minor and easily corrected. Two problems arose because the Model 400 system was different from the other VAX systems in use at two of the sites. One site reported minor differences in results from a benchmark program, which was due to differences between run-time libraries used for full VAX architecture implementations and subset implementations.[8] The other problem resulted because a customer program was referencing an internal processor register that does not exist in subset implementations.

Field test progress was assessed two months into the test, at the beginning of June. The results of field test were then examined, together with the data obtained from other qualification testing. Since no major problems had been found, we decided to proceed with plans to ship the Model 400 system as scheduled in mid-July.

Some improvements could have been made in the field test process. First, site audits prior to installation of the prototypes were not very thorough. Many of the sites were not running the necessary software revision levels. Therefore, the new machines could not be immediately put into VAXcluster environments with existing machines. Second, some pieces of hardware were missing and had to be supplied later. Better communications with the site prior to shipping prototypes would have reduced these problems.

The System Integration Group wrote some monitoring software that was to be installed on each field test machine. Because this was written at the last minute, it was not properly tested and did not work properly. The problems were fixed, but the monitoring software was not run at all the sites. Finally, how to get data from the monitoring software back to the engineering groups was not well defined. Thus, usage data obtained from field test machines was spotty. These tools and methods are being improved for use in the field test of future products.

VAX Architecture Verification Group

The Architecture Verification Group ensured that the Model 400 CPU conformed to the VAX architecture specification.[8]

Test Process Overview

The architectural verification process consists of running two programs — AXE (architecture exerciser) and MAX (multi-instruction architecture exerciser) — in various modes for a given number of test cases. The tests are simple to run. However, the test programs are quite complex and required many years to develop. The Architecture Verification

Group maintains and enhances these test programs and the databases used to verify the architecture.

When one of the tests fails, the group identifies the problem and helps resolve it. Once the problem is fixed, the group repeats the tests.

AXE and MAX

The VAX instruction set consists of over 360 instructions and 21 addressing modes. Most modes are valid for up to six operands per instruction. Conceptually, both AXE and MAX divide an instruction's context into several components. These components include opcode, operand specifiers, operands, page protection and validity, and processor status long word (PSL). For each component, valid and invalid values are pseudo-randomly selected to create a test case. The exerciser continues to create unique cases for as long as it is run.

The VAX architecture has a clearly defined exception and instruction restart structure. Much of the VAX architecture's complexity is in those operations. Therefore, both AXE and MAX favor values that cause faults. Each program establishes a situation with faults, starts the instruction or sequence, verifies that the fault occurs, fixes the fault, restarts the instruction, and verifies that it completes correctly. Upon completion, AXE or MAX compares the results from the unit under test to a known good reference, and reports any differences. The known good references contain the correct results of each test case. These references have been accumulated over the years of testing VAX systems and have changed as the VAX architecture changed.

AXE AXE is the older and simpler exerciser. It creates test cases that consist of a single instruction. A typical instruction stream would be:

```
ADDF3 R1, R2, S^#1 R1=00008000, R2=46246811
```

When this instruction is first executed, either a reserved operand fault (on R1) or a reserved addressing mode fault (short literal destination) should be reported. AXE will fix whichever fault is reported and restart the instruction.

Assuming the reserved addressing mode fault was reported, the instruction might then look like this:

```
ADDF3 R1, R2, R3 R1=00008000, R2=46246811
```

If the reserved operand fault is reported, AXE will change R1 to a valid floating point value and restart.

When the instruction completes, AXE compares the instruction stream and the relevant data to reference data. In this example, the relevant data includes the three general purpose registers.

Both AXE and MAX ignore a machine state defined to be unpredictable for a given condition. Therefore, allowable differences between implementations do not cause erroneous failure reports.

Limiting AXE's testing to single instructions precludes meaningful testing of the pipelining that is common in today's CPU designs. MAX overcomes this limitation. MAX currently acts as an adjunct to AXE. However, it will eventually replace AXE.

MAX MAX is similar to a compiler in that it creates complete instruction streams. However, MAX does not have source code to ensure that the resultant machine code is logically consistent.

To test how a CPU handles inter-instruction data dependencies, MAX must create test cases with instructions that share registers and memory locations. Creating sensible instruction streams can be difficult. For instance, the result of one instruction could be used as part of an address calculation for a subsequent instruction. However, the likelihood is slim that the result of a randomly selected arithmetic calculation will be used within the test case's virtual address space.

MAX first creates and executes sensible cases in logical steps. It then assembles and executes each case as a whole.

After selecting the first instruction, MAX executes the instruction, including fault restarts, and saves the final results. MAX next selects and places the second instruction in memory following the first instruction. Where possible, it uses the results of the first instruction for operands and operand specifiers of the second instruction. MAX selects new values for operands and specifiers for which old values cannot be used.

MAX includes the new values in the initial state of the entire case. It then executes *only* the second instruction. This process repeats until an instruction stream of the desired length is created. At this point, the entire stream is executed. Once the stream is run, the results of all of the instructions are compared. This comparison is made against the results of the single-step execution and the results of the known good reference.

Test Process

The minimum testing requirements for shipment of VAX systems were developed from experience gained from testing all previous VAX systems. The requirements are a compromise between the number of bugs likely to be found and the time required for the tests. The testing program consists of over 200 million test cases.

The use of AXE and MAX in the system qualification process is a continuation from the CPU design process. Both exercisers were used extensively to test simulations of the CPU design. However, this testing is limited by simulation speed. Testing on hardware at speed is necessary for more thorough coverage.

Testing of the VAX 6000 Model 400 started in late April. To complete verification in less than two months, four machines were used. The machines executed different test cases in parallel.

The VAX 6000 Model 400 CPU was the first VAX CPU for which *no* hardware bugs were found by AXE or MAX during final qualification. This result is a testament to the careful design and extensive testing during the simulation and chip debug phases of the CPU design.

However, a bug was found in the VMS operating system's floating point emulation code that calculates the POLYx instructions.[8] Because the Model 400 CPU is the first that did not include the POLYx instruction in the CPU microcode, it was the first CPU that required use of the VMS emulator for this instruction. For this reason, the bug had never been seen before.

The Architecture Verification Group traced the bug and confirmed that a patch to the emulator fixed the bug. The group then reran all of the test cases that exercised the floating point emulator to confirm that no new bugs were introduced. The patch was included in the VMS version 5.2 mandatory update.

Midrange System Evaluation Engineering Group

The Midrange System Evaluation Engineering Group tests new products to isolate design faults and configuration incompatibilities. This group examines various system configurations, tests interaction between components, and tests special situations, such as power failure and recovery.

Test Process Overview

System evaluation typically begins when the first hardware and software prototypes become available and continues through to product shipment. Evaluation planning often begins six to nine months prior to the actual process with the creation and review of a test plan. The test plan is based on product specifications and information from development groups. It identifies the test tools, configurations, and strategy the System Evaluation Group will use.

The System Evaluation Group exposes new products to a wide range of hardware and software configurations. This exposure is achieved by combining a stable, well-equipped, and versatile laboratory with specialized test software and test procedures. These tests are done before system shipment to customers, when modifications can be made at minimal expense. The group complements the qualification efforts of the product development groups by allowing these groups to focus attention on product-specific engineering issues.

Test Strategy and Test Software

The overall VAX 6000 Model 400 test strategy involved the installation of the new processors in a wide range of system configurations. The group subjected these configurations to a series of interactive load experiments, which used internally designed and developed software tools. These tools concurrently stressed all configured processors, memory arrays, I/O adapters, and communication devices. Stress in this context means heavy loads in terms of I/O bandwidth used for a given data bus, and minimal idle CPU time for processors. This test strategy has proven successful since it was first used in the mid-1970s for PDP-11 systems.

The processor and memory tests exercise VAX instructions, the floating point processor, and the cache subsystem. Configurations with large memories are tested with a specially developed memory exerciser. This program references memory in the least efficient manner to force high page fault and low cache hit rates.

I/O adapter tests exercise specific devices. The tests include a system exerciser that can generate various I/O rates to disk, tape, and terminal devices, while verifying data integrity.

Communication tests include a DECnet exerciser and an Ethernet local area network program. These tests generate a high level of network activity and check data integrity.

Configuration Selection and Test

The System Evaluation Group selects configurations according to many factors, including

- VMS operating system restrictions
- Bus architectures and slot placement limitations

- Power and packaging restrictions
- Marketing requirements

The maximum number of each supported option is tested within laboratory resource limits. Because the Model 400 CPU is a higher performance processor for the existing VAX 6000 platform, it was tested within established VAX 6000 family configuration guidelines. The System Evaluation Group chose nine Model 400 system configurations to test from one up to six processors, 32 to 256 megabytes of memory, and from two to six VAXBI channels.

An important part of configuration testing for a new processor involves verification of proper system initialization and operating system boot using various load paths. With the Model 400 system, this testing meant loading the operating system through different VAXBI channels, disk or CI adapters, and load devices. Several problems were noted with bus adapter initialization and self-test while testing certain VAXBI channel configurations. These configuration-dependent problems were corrected by modifying console and operating system software.

As shown in Figure 1, VAXBI channels on the Model 400 were also configured with all currently supported VAX 6000 system I/O adapters. Although testing with large and diverse system configurations poses logistical challenges, these environments will often succeed in exposing device compatibility problems.

The device compatibility problems found during the Model 400 evaluation occurred either during operating system initialization or when interactive workloads concurrently exercised all system devices. One such problem resulted from the increased processing speed of the Model 400 processor and would occur only when an adapter was tested in a specific configuration under a certain workload.

More specifically, this problem was due to a race condition between a VMS application level program issuing I/O requests and the adapter hardware pro-

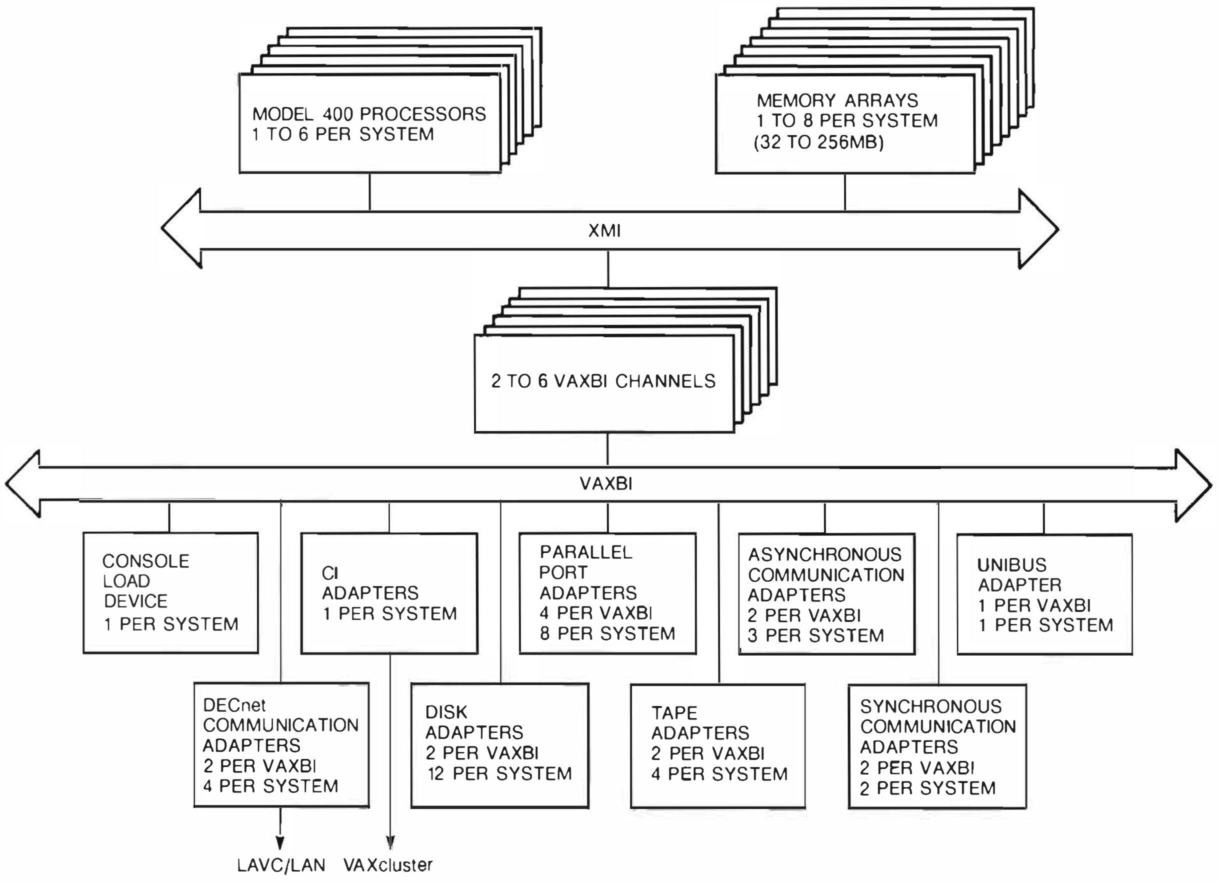


Figure 1 Adapters/Options Tested during VAX 6000 Model 400 Evaluation

cessing and returning a response. With slower machines, the adapter hardware won the race; but with the faster Model 400 processor, the host won by issuing commands faster than the adapter hardware could process them. The problem was corrected with a modification to the device driver software.

Complete configuration test coverage for the Model 400 processor also required VAXcluster/local area VAXcluster (LAVC) and DECnet/local area network (LAN) evaluation using each of the supported CI and DECnet communication adapters. To accomplish these tests, a 13-node VAXcluster was established composed of eight VAX host systems, four HSC mass storage servers, and the VAX 6000 Model 400 system under test.

Primarily, the VAXcluster/LAVC and DECnet/LAN testing verified the functional compatibility of the Model 400 processor with the VAX 6000 series CI and NI adapters. Cluster and LAN activity were used simply as adapter loads. Interactive experiments were designed to emphasize stress at the local system level. Cluster-level verification was deferred to another group.

Evaluation of the VAX 6000 Model 400 system also verified the Model 400 system-level power-fail/warm-restart capability in large configurations with high compute and I/O loads. These tests ensured that battery backup units would maintain supply voltages for the guaranteed duration. Further, these tests ensured that error reporting and recovery procedures operated properly.

Interactive Test Method

The interactive test method first selects test software. It then modifies parameters such as VMS queue I/O (QIO) request size, number of outstanding commands, and device mode of operation. Using this method, three system workloads were generated.

The first workload used small VMS QIO request sizes and maximum QIO request queue lengths to achieve high I/O rates. This workload minimized idle CPU time and maximized time on the interrupt stack.

The second workload used large VMS QIO request sizes with sequential disk accesses and device loopback to generate high bus utilization rates (bytes per second). This workload saturates I/O buses and interconnects by generating large amounts of direct memory access activity.

The third workload combined CPU/memory and I/O adapter test software with a distribution of small, moderate, and large VMS QIO request sizes. Memory arrays were also configured in a non-interleaved mode to degrade memory access time and aggravate potential bus timeout conditions.

The System Evaluation Group ran interactive test experiments for a minimum of four hours and a maximum of three days. A typical experiment lasted 18 hours. Most long duration experiments were performed using the combined CPU/memory and I/O adapter workload. The longer run-times provided adequate time for adapter exercisers to step through a preprogrammed range of transfer sizes.

Following each test, VMS error counters, test software status reports, and VMS error log entries were examined for system or device errors. The group then characterized problems in terms of their frequency, repeatability, and the system environment in which they occurred. The system environment included detailed information regarding hardware configuration, the software test tools and parameters used, and module, operating system, device driver, and firmware versions.

Although some of the problems noted during the Model 400 evaluation were easy to reproduce and occurred frequently, others were intermittent in nature and not so easily induced. For example, the configuration-dependent problems occurred each time a specific configuration was tested, whereas a particular device compatibility problem was intermittent and required long duration runs before appearing.

The detailed problem descriptions, along with pertinent error log or crash dump data, were provided to the appropriate development groups for analysis. Also, descriptions were logged in the project-specific problem reporting databases to ensure that problems were properly tracked and resolved. The System Evaluation Group then worked with the development group to fix the problem.

From a total of six problems noted during the Model 400 evaluation, four were configuration-dependent and corrected through modifications to the VMS operating system or console microcode. Two problems were device compatibility bugs that occurred during operating system initialization and interactive testing. These problems were fixed through modifications to the VMS operating system or device driver software.

When the VAX 6000 Model 400 system evaluation was completed, a final report was distributed. The report summarized the configurations and test tools used, the specific experiments performed, and the current status of all problems that were identified.

Summary

System qualification is the last stage in the system development process. The qualification process for the Model 400 system was designed specifically to take into account computer-aided design and simulation used by the hardware design process.

As our experience with the Model 400 system has shown, nearly all of the problems found during system qualification were in areas of the system that could not be simulated. As a result, the qualification process is most effective when focusing on testing those parts of the system that cannot be simulated because of their complexity, which presents both a challenge and an opportunity.

The challenge is to design test processes and tools that can adequately test a complex system in a reasonable time. With SITP and other test tools, such as those used by Midrange System Evaluation Engineering, we have made a significant start in developing these tests. However, there is still much room for improvement, and work is continuing in this area.

The opportunity is to shorten the qualification process. Because first-pass hardware is more robust, more system testing can occur earlier. Also, better test tools enable us to provide more test coverage in less time and with fewer resources, both in prototypes and number of people. We took advantage of this opportunity in the Model 400 system qualification to cut the length of field test by 25 percent, thereby bringing the Model 400 system's new technology to market faster.

References

1. B. Allison, "An Overview of the VAX 6200 Family of Systems," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 10–18.
2. B. Allison, "The Architectural Definition Process of the VAX 6200 Family," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 19–27.
3. R. Gillett, "Interfacing a VAX Microprocessor to a High-speed Multiprocessing Bus," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 28–46.
4. P. Sullivan et al., "The VAX 6000 Model 400 Scalar Processor Module," *Digital Technical Journal,* vol. 2, no. 2 (Spring 1990, this issue): 27–35.
5. D. Slater et al., "Vector Processing on the VAX 6000 Model 400 System," *Digital Technical Journal,* vol. 2, no. 2 (Spring 1990, this issue): 11–26.
6. H. Durdan et al., "An Overview of the VAX 6000 Model 400 Chip Set," *Digital Technical Journal,* vol. 2, no. 2 (Spring 1990, this issue): 36–51.
7. J. Basmaji et al., "The Role of Computer-aided Engineering in the Design of the VAX 6200 System," *Digital Technical Journal,* vol. 1, no. 7 (August 1988): 47–56.
8. T. Leonard, ed., *VAX Architecture Reference Manual* (Bedford: Digital Press, Order No. EY-3459E-DP, 1987).

Thomas C. Furlong
Michael J. K. Nielsen
Neil C. Wilhelm

Development of the DECstation 3100

The DECstation 3100 is the first member of Digital's family of high-performance ULTRIX workstations. Built with the R2000 chip set from MIPS Computer Systems, Inc., and highly integrated I/O and graphics subsystems, the DECstation 3100 implements 12 mips of RISC-based computing, workstation I/O, and excellent bit-map graphics on a single module. The DECstation 3100 workstation runs Digital's ULTRIX operating system (compatible with UNIX software) as well as DECwindows software, TCP/IP, DECnet software, and Network File Services (NFS). The workstation can be configured with 8MB to 24MB of parity-protected memory, monochrome or 8-plane color graphics, 15-inch or 19-inch monitors, and SCSI disk and tape devices. This paper describes the DECstation 3100 product, the design effort, details of the system, and measured benchmark performance.

System Overview

Packaged in a desktop system box, the DECstation 3100 workstation is implemented as a single module that contains CPU/FPU, separate instruction and data caches, memory control logic, Ethernet and small computer systems interconnect (SCSI) controllers, four serial lines, and video display logic. Connectors on the module accept as many as 12 memory modules of 2 megabytes (MB) apiece as well as a single monochrome or color frame buffer module. The box optionally contains 3.5-inch, 104MB SCSI disk drives. An SCSI connector on the back of the system box supports the attachment of additional SCSI devices, such as the 332MB disk drive, the 95MB tape drive, and the 600MB CDROM disk drive.

Background and Project Goals

Digital's research and development groups in Palo Alto had used a UNIX operating system in a client/server computing environment for more than three years. The clients were various VAX workstations. The servers were VAX systems and RISC-technology research machines, called Titans, which were developed by Digital's Western Research Laboratory.

The major frustration in this environment was the lack of processing power at the workstation. Computer room servers delivered up to 12 million instructions per second (mips), but most office workstations delivered only 1 mips. This disparity caused many workstations to be used only as rather expensive terminal emulators for the larger machines. The slow workstations also meant that window applications were often bogged down in screen update activity, and that NFS performance was limited not by device or network speeds but by the workstation's processing power.

The primary goal of the DECstation 3100 project team was to produce a fast RISC ULTRIX workstation that would bring processing and windowing performance to the user's desk at a competitive price. The product would run the ULTRIX operating system, DECwindows products, and network software for both TCP/IP and DECnet networks.

In early May of 1988, we received approval to build an ULTRIX workstation that featured increased processing power. The remaining design goals were time to market, packaging, reuse of existing designs, and system cost and price. The aggressive schedule called for the first workstation to ship in mid-January 1989. We were asked to use the new desktop system box designed for product to be later announced as the VAXstation 3100. We were also asked to reuse any hardware or software elements of the VAXstation 3100 that we possibly could. Our own view of the marketplace caused us to choose an entry-level price of approximately $10K.

At the end of the project, we achieved all of our goals. We shipped the first system in the desired box on the very day we had promised. We reused most of the existing ULTRIX, windowing, and network software, and supported the same internal disk drives as the VAXstation 3100. We held the entry-level price within 20 percent of the $10K goal.

System Cost Issues

We needed to control the high-cost items of processor, caches, and memory. We also had to resist the tendency to add things, either because other groups requested them or because we had ourselves wanted them.

The processor choice was based more on cost per mips than the absolute cost of the processor chip set itself. The cost per mips consideration led us to select a RISC processor instead of a VAX processor in order to obtain at least a two-to-one performance advantage. Since we were building a product for users of the ULTRIX operating system, the lack of VAX instruction compatibility was not an issue. We chose the R2000 chip set from MIPS Computer Systems as the best CMOS RISC technology on the market.

Even though caches contribute significantly to system performance, we still considered the use of small caches to reduce cost. Simulations of system designs indicated that the higher expense of large caches was necessary to achieve fast desktop performance.

Our most difficult challenge was determining how to implement memory. We did not want to burden the entry-level systems with more memory than was necessary, but we did want the opportunity to add memory to systems that could use it. We considered memory on the system module, memory on daughter cards, memory on commodity single in-line memory modules (SIMMs), and finally memory on custom-designed SIMMs. In the end, we chose the custom-designed SIMMs because of their density, cost, and configurability.

Basic Project Rules

To succeed at this project we kept the size of the team to the minimum and isolated the team from outside influences. We settled on a minimal product focus team and a minimal design team. The five-person product focus team would manage the project while the three-person hardware design team would build the machine. Everyone would work from the Palo Alto base.

The first machine was running two months after the project start date. By that time the design team had expanded to about twenty people developing the electronics package, diagnostics, software, and documentation. Many designers, particularly software engineers, temporarily relocated from New Hampshire to participate in the project. Researchers from Digital's two Palo Alto research laboratories gave generously of their time in reviewing the design, developing new software, and testing prototype systems.

The product focus team built extended support groups for functions such as manufacturing, marketing, sales, and application development.

Basic Design Rules

From the beginning we agreed upon some basic design guidelines. We strictly adhered to these decisions throughout the design phase of the project. We would develop all functions on one system module. Anything that did not fit would not be part of the product.

We would do no ASIC or other IC design; the schedule did not allow for it. So we would use lots of random, low-cost, standard logic functions, PALs in particular. Opportunities for lower cost integration would be saved for follow-on products.

We would build dumb I/O controllers. This decision eliminated the use of secondary processors, microcode, and hardware coordination of intelligent devices. Various-sized buffers on every controller would allow devices to run at their own speeds independent of general processor activity. Software drivers running on the general CPU would interact with industry- or Digital-standard controller chips for the I/O subsystems.

We would build dumb graphics — no pipeline, no graphics processors, no rendering chips, but simply a frame buffer configured as part of the main memory address space. The only exception was to add a color plane mask for use by the color software.

We would aim for ease of manufacturing by keeping the option choices low. The only choices to be made in manufacturing the system box were how many memory modules to insert, which frame buffer module to insert, and whether to add one or two internal disks. This decision kept the total manufacturing permutations down to 30. (Earlier Digital workstations were up around 1000.)

We would try to let experienced computer users upgrade and service their own workstations. The choice of system box limited this capability since the VAXstation 3100 box was not originally designed for user access. We simplified the original box design and left much of the box empty to make the electronics more accessible.

Conflicts and Resolutions

At project start, the design team resisted the idea of offering both monochrome and color graphics options. We believed in the value of providing color wherever and whenever possible on workstations: customers prefer color, and it provides additional functionality. However, we were not confident we could develop both graphics options and still remain on schedule. We decided to use a simple frame buffer along with a single, configurable video output design. We would bear the cost of an unnecessary VDAC in the monochrome system, but we only had to design one system.

Although the hardware design team was confident that cost/performance trade-offs were the right ones, other project members voiced their concern. Some issues were memory size and I/O and graphics performance. Would the maximum memory size of 24MB be large enough for applications such as computer-aided design (CAD) and modeling? How would Ethernet and disk performance compare to VAXstation and competitors' workstations? Would the choice of frame buffer matched with a fast RISC processor deliver adequate graphics performance, particularly in color systems?

These questions were answered once the software development was complete and performance measurements could be made on late prototype systems. See the Product Qualification section of this paper.

Another topic we debated was whether to allow peripheral devices in the box, which was clearly designed for such devices. Various combinations of disks and tapes in the box presented three problems: more complicated options in manufacturing, a heavy drive plate and complex cabling for the user to remove during system upgrade, and a power problem if an internal tape drive was present. Eventually we permitted only the internal 3.5-inch disk drives and simplified the drive mounting plate to be easier to remove.

Product Qualification

Since the entire project time was only eight months, we needed to maximize the test time of the DECstation 3100. The hardware was an entirely new design. The software was a port from a VAX base to a RISC base, and much of the lower level graphics software was completely new code.

To test modules and the hardware system, we sent many early prototypes to a local testing laboratory for stress testing. While running both diagnostics and the ULTRIX operating system, we shook the systems, power-cycled them, temperature-cycled them, even submitted them to rainfall due to an environmental failure. We recorded every failure and traced it back to its source. Many of these failures led to changes in diagnostics, components, placement, and mechanical solutions. This early stress testing did not uncover any problems not seen elsewhere, but with a small number of machines, it validated all problems seen in the other test situations.

To test the total system with its software, we invented a qualification team nicknamed the "wrecking crew," a group of about twenty-five senior engineering researchers and developers. They agreed to accept early prototypes and to subject them to heavy use for a period of three months. Their goal was to break the systems, as often and in as many ways as they could. During the wrecking period, we constantly installed the latest software changes, replaced diagnostic ROMs, added hardware, and moved systems from person to person to allow everyone to try different configurations. Each crew member was responsible for an exhaustive test of a subset of the ULTRIX commands and utilities.

The wrecking crew was a huge success. Team members reported 785 problems, complained mightily and usefully, ported CAD tools, window applications, and compilers in their spare time. Their constant demand for the performance they expected exposed many bugs that were artificially limiting performance. Best of all, the wreckers almost doubled the number of software experts knowledgeable about the workstation from an early stage and thus contributed significantly to system quality.

Processor Subsystem Details

The DECstation 3100 CPU consists of the MIPS R2000 integer processor, the R2010 floating point coprocessor, and four R2020 write buffers. The chip set operates at 16.67 megahertz.

In the DECstation 3100, the R2000 chip set runs in "Little Endian" mode. In other words, bits within bytes and words are counted from right to left, and the low-order bit is the rightmost bit in a word. "Little Endian" mode means that the integer data format of the DECstation 3100 is identical to the integer data format of any VAX processor. The floating point data format is compliant with IEEE standards.

The R2000 CPU implements the instruction set, processor registers, virtual memory, and interrupt system as defined by the R2000 architecture. The CPU maintains the direct-mapped, write-through data cache. Each cache is 64 kilobytes (KB) in capacity with a 4-byte line size. The tag and data stores of each cache are byte-parity protected, and cache parity errors transparently generate cache misses to reload the cache from memory.

The R2010 floating point coprocessor implements the IEEE arithmetic functions and coprocessor registers defined by the R2000 architecture.

The R2020 write buffer implements a four-stage write buffer for the CPU. This write buffer allows the CPU to write to its write-through cache without stalling the CPU as long as the write buffer is not full.

Graphics

Graphics on the DECstation 3100 is implemented in a tightly integrated subsystem. Frame buffer memory is a region of memory in the processor address space — 256KB in a monochrome system and 1MB in a color system. Less than half of the monochrome frame buffer and three quarters of the color frame buffer are displayed on the workstation monitor. The remaining frame buffer memory may be used for storage of graphics data structures such as fonts. The frame buffer memory is not parity protected.

At boot time, the ULTRIX operating system detects the size of frame buffer memory and whether the system is monochrome or color. Because frame buffer memory is cacheable and addressable in the same way as the dynamic random access memory (DRAM), the software is able to achieve extremely high performance without any special-purpose graphics hardware.

A color plane mask allows processor writes to the color frame buffer to affect only specific bits of a pixel. This design allows modification of a given plane of the color frame buffer using only write cycles, which increases performance significantly.

The graphics programmable cursor supports a 16-by-16 pixel, two-plane cursor. The cursor can take two forms: a 16-by-16 bit pattern or a crosshair whose lines may extend to the edges of the visible raster or may be clipped to a programmed region. The cursor in a color system may have up to three colors, and the cursor in a monochrome system may have up to three gray-scale values.

Memory

The DECstation 3100 supports 8MB to 24MB of byte-parity protected memory in 4MB increments. The memory system includes both the DRAM array and a video random access memory (VRAM) frame buffer. The video frame buffer has the same memory access characteristics as memory and may be cached if desired. The memory system supports byte, halfword, word writes, and word reads.

The memory system control logic is optimized for minimum memory read latency, at a slight cost in memory write latency. On a memory read, the CPU incurs a five-cycle stall in the absence of memory refresh contention. The memory system can sustain five-cycle reads, which results in a peak read bandwidth of 13.3MB per second.

Memory writes to an empty write buffer complete in eight cycles, but do not stall the CPU. Successive memory writes complete at the rate of six cycles, and the CPU stalls whenever the write buffer is full. The memory system can sustain six-cycle writes, which results in a peak write bandwidth of 11.1MB per second.

The DRAM and VRAM arrays are implemented with SIMMs. Each DRAM array contains 2MB of memory on a double-sided module. The VRAM arrays contain either 1 megabit (Mb) (monochrome) or 8Mb (color) of frame buffer memory on single-sided modules.

Ethernet

The Ethernet interface on the DECstation 3100 consists of a CMOS controller chip and a 64KB buffer. The controller chip manages transmission and reception of packets through ring descriptors and packet buffers located in the Ethernet buffer. The buffer is time-multiplexed between the controller chip and the workstation CPU.

Connection to the Ethernet is by a thick-wire or ThinWire cable. A push-button switch on the rear of the system box selects the appropriate connector.

SCSI

The DECstation 3100 supplies a small computer system interconnect (SCSI) as the interconnect for storage peripherals. The workstation's SCSI interface consists of a gate array controller chip and a 128KB buffer. The controller chip manages the SCSI bus through selection, DMA data transfer, and disconnect commands. The interface supports command disconnect/reconnect and synchronous data transfers at 4MB per second on the SCSI bus. The buffer is time-multiplexed between the controller chip and the workstation CPU.

An SCSI connector on the rear of the DECstation 3100 system box allows connection of external SCSI peripheral devices. Digital offers a 332MB disk, a 95MB tape, and a 600MB CDROM reader. Each of these devices is packaged with power in its own sidecar box.

Table 1 Comparison of RISC System Performance

	DECstation 3100	Sun 4/110	Sun 4/260	MIPS M/120-5
Dhrystones/second	22.7K	12.8K	16.7K	24.7K
Linpack single precision (MFLOPs)	3.7	0.95	1.3	4.0
Linpack double precision (MFLOPs)	1.6	0.57	0.89	2.1
Stanford small integer benchmark (seconds)	0.115	0.220	0.150	0.118
Digital Review's CPU 2 benchmark suites (seconds)	6.91	18.99	13.71	NA
XLIB graphics performance rate	4.9K	NA	0.7K	NA

Serial Lines

Four serial line interfaces are present and are programmable from 50 to 9600 bits per second. The serial transmitters are double buffered, and the receivers share a 64-entry FIFO. The workstation uses one serial line for the keyboard and another for the mouse. One serial line, designed for modem use, supports data-terminal-ready/data-set-ready (DTR/DSR) control signals.

Software

The DECstation 3100 runs the standard software expected by users of UNIX operating systems as well as software that allows easy networking and windowing of VAX and 3100 systems. Digital's ULTRIX operating system is compatible with Berkeley V4.3, AT&T System V, and is compliant with POSIX standards. DECwindows software runs on the DECstation 3100 in both the monochrome and color configurations and integrates seamlessly with DECwindows running on VAX systems with VMS operating systems. ULTRIX supports DECnet, TCP/IP, and NFS. Compilers include the C, FORTRAN, and PASCAL compilers adapted from the compilers from MIPS Computer Systems.

Performance

Table 1 lists key performance measures of the DECstation 3100 workstation. For comparison purposes, the table also lists the performance of other RISC systems, namely the Sun 4/110 and 4/260 and the MIPS M/120-5.

Table 2 lists the SCSI and network subsystem peripherals I/O performance of the DECstation 3100.

Table 2 DECstation 3100 I/O Performance

Subsystem Peripheral	I/O Performance
RZ55 file reads (with read-ahead microcode)*	930KB per second
RZ55 file reads	330KB per second
RZ55 writes	320KB per second
NFS file reads (with read-ahead RZ55 microcode)	450KB per second
NFS file reads	330KB per second
NFS file writes	320KB per second
IP/UDP 1500-byte packets	800KB per second
IP/UDP 64-byte packets	1000 packets per second
IP/TCP end-to-end	350KB per second

*RZ55 read-ahead microcode has not yet been released by Digital Equipment Corporation.

Conclusion

One of the delights of the DECstation 3100 project was that we were building the machine that we ourselves had wanted for a very long time. By using a small, focused core design team and resisting incremental additions, we achieved the aggressive time-to-market goal.

Larry B. Weber

Compiler Optimization in RISC Systems

Compiler optimization determines the level of RISC system performance. The architectural design of compilers from MIPS Computer Systems, Inc., combined with support tools facilitates compiler optimization and overall system throughput. The compiler design takes advantage of small and high-speed cache memory to enhance performance. The cord tool positions the program in memory to ensure that the most frequently used memory locations never compete for the same cache locations. Portability is crucial to compiler effectiveness. MIPS compilers implement many industry-wide extensions to the standard languages to make them compatible with other implementations.

RISC (reduced instruction set computer) system performance embodies many components. In addition to the performance of individual instructions, the processor architect must consider how the compiler combines the instructions, how system vendors construct the memory system, and how the user writes programs. Of particular importance is how well the compiler optimizes programs for a given hardware architecture. In addition, program portability is essential to ease the burden of moving applications to new systems. By considering all such aspects of system performance, the processor architect can use the full potential of a RISC system.

Traditional CISC (complex instruction set computer) processors were developed without significant information on how high-level programming languages would use them. In contrast, RISC architects make trade-offs between microprocessor structures and compiler complexity, with the goal being overall system performance.

The compiler is the key link between the architect and the system user. Therefore, it is essential that the processor architect completely understand the compiler and its capabilities.

Compilation is the process of converting high-level source code written in a programming language into machine code for a target machine. This process must consider translating the instructions correctly into machine language, as well as into optimal machine language. Often, the high-level language masks the primitive level of the target machine by providing programming tools that do not directly correspond to the machine's features. Compilers must also deal with programs that do not take best advantage of these features. The optimizer is the portion of the compiler that deals with performance issues.

Reprinted with the permission of *ESD Magazine*, December 1989, Digital Design Publishing, Westborough, MA 01581.

Optimization Boosts Performance

Optimization occurs at many stages within the compiler. Some optimizations are best done at the front end of the compilation process when first processing the source program. These optimizations are called language-dependent optimizations because they rely on features unique to a specific programming language. Other optimizations, called machine-dependent optimizations, are performed late in the process because they require detailed information about the target machine and how the program actually uses that machine. Still, other optimizations are independent of both the source language and the target machine.

Compilers from MIPS Computer Systems consist of several independent front ends that convert individual languages into a common intermediate code called ucode. (See Figure 1.) MIPS currently supports six programming languages: ADA, C, COBOL, FORTRAN, PASCAL, and PLI; ANSI C and C++ will be available in 1990. The common back end performs the bulk of the optimization and generates machine code.

The common back end of the compiler uses a variety of optimization techniques that require varying amounts of information. The compiler must gather the information from the source code and analyze it. Peephole optimizations require the

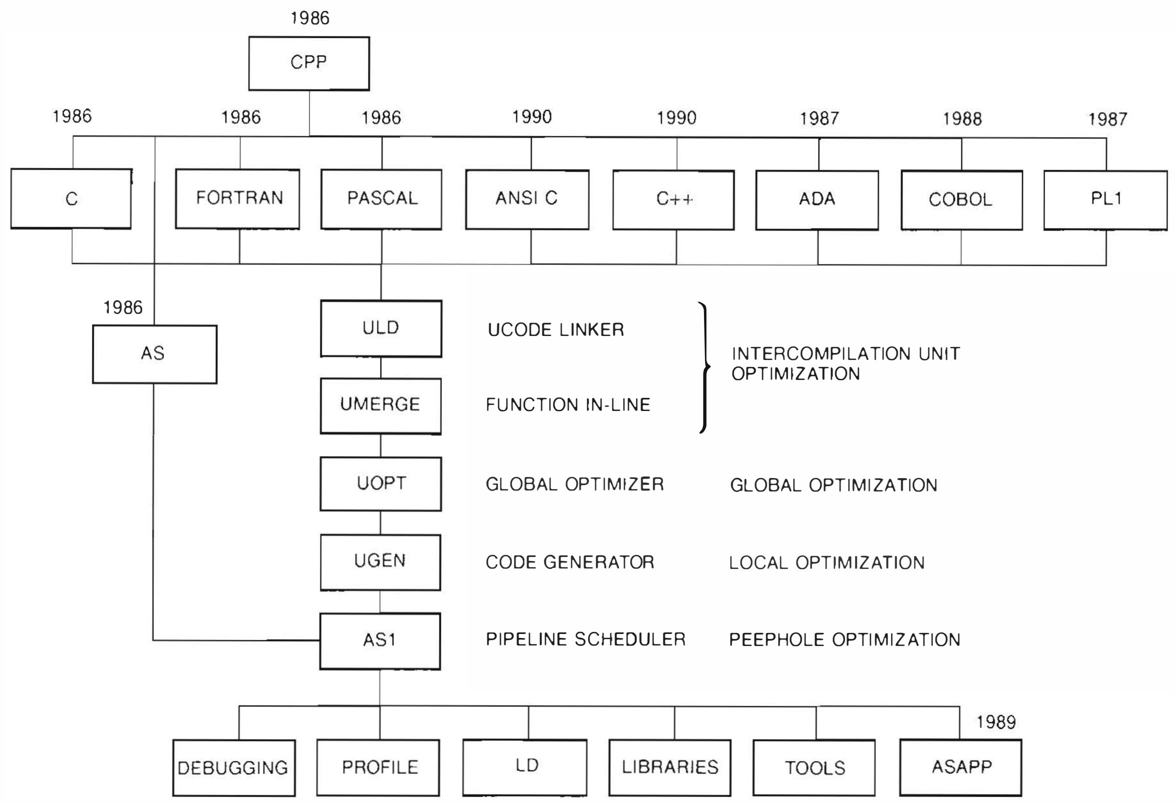


Figure 1 Compiler Structure

least amount of information—usually only an instruction or two. Global optimization requires the most information; it must take into account control flow (the branching and looping structure of a program) and data flow (the data usage within each section of the program). Intercompilation unit optimizations represent an extreme form of global optimization that occurs between independent source files. Local optimization requires an intermediate amount of information—usually data usage within a group of consecutive statements. Table 1 lists optimizations shared in the MIPS compilers.

One of four optimization levels (-O0, -O1, -O2, or -O3) can invoke the MIPS compiler. The levels indicate the relative compilation speed—not the importance—of the various optimization classes that the compiler can implement. For example, a program compiled at the -O0 level, which specifies no optimization, would compile faster than a program compiled at the -O3 level, which offers the full range of optimizations. These optimization levels also correspond closely to the components invoked during the compilation process.

The -O0 option disables the optimization normally performed by the code generator and assembler. The -O1 option (the default) designates minimal and fast optimization. Under this option, the code generator and assembler perform local optimizations within basic blocks. Apart from traditional local optimizations such as local common subexpression, expression simplification, constant folding, dead code elimination, and peephole optimizations, the code generator performs branch and label optimization, and the assembler performs architecture-dependent pipeline scheduling. Compilation speed does not lengthen noticeably between -O0 and -O1. Thus, -O0 is seldom needed and is used mostly for comparison studies.

Option -O2 adds the uopt phase to the compilation stream to perform global optimization and register allocation within the full range of individual procedures. Compilation time might lengthen substantially because global data-flow analysis and coloring register-allocation algorithms are invoked.

Supporting optimizations across multiple source files is the -O3 option. This option adds the uld

Table 1 Optimization Methods

Peephole optimization	Instruction scheduling Instruction selection Instruction substitution
Local optimization	Call/return selection Branch-to-branch optimization Local subexpression elimination Constant folding Expression simplification Dead code elimination Local register assignment Short-circuit evaluation
Global optimization	Invariant code removal Strength reduction Global register assignment Global subexpression elimination Shrink-wrapping register saving Linear test replacement Loop unrolling Tail recursion Copy propagation Redundant store elimination
Intercompilation unit optimization	Interprocedural register allocation In-line expansion of procedures

phase, which combines separate compilation units into a single file at the ucode level. Thus the option enables multimodule programs to achieve the same degree of optimization as single-module programs. The umerge phase selectively expands procedure calls by in-line substitution. The resulting ucode object is then sent into normal back-end optimization and compilation stream starting with uopt. The -O3 option causes uopt to perform interprocedural register allocation; uopt also benefits from the complete information in the linked ucode file to perform the other global optimizations normally associated with this phase.

Local Hardware Architecture

MIPS' architectural design facilitates compiler optimization and overall system throughput. Important to system performance is the memory hierarchy. A split cache provides independent access to both instructions and data in a single cycle. A single combined cache would limit the processor to obtaining only a single instruction or data item each cycle.

The compiler design takes into consideration the effect of cache memory. On average, instruction references cache miss less frequently than data references; this observation allows the compiler to prefer slightly longer instruction sequences if they avoid extra data references. The instruction cache miss rate is lower because the hardware loads multiple contiguous words into the cache on a miss, and the sequential nature of instruction execution takes advantage of this locality.

Register optimization makes the most significant architecture/compiler performance enhancement. Because the optimizer knows that data allocated in registers can be accessed without delay, it places the most frequently used variables into registers. The optimizer computes the lifetime of individual data items and replaces the memory usage with a register usage. The relatively large number of registers makes it likely that the optimizer can successfully promote the most frequently referenced variables into a register. (There are thirty two 32-bit integer registers and sixteen 64-bit floating-point registers.)

One of the most important architectural features used to improve performance is the instruction pipeline. Although several cycles are required to actually complete the instruction, the processor can be viewed as if each instruction takes only one cycle because a new instruction is started each cycle. The compiler is aware of several exceptions; for example, load instructions require one instruction between the load and the use of the data loaded. This load-delay slot is used by the compiler for another instruction—effectively, the cache memory access of the load executes in parallel with the other instruction. This technique is called instruction scheduling. Other examples of parallel instruction execution include

- Overlapping a branch instruction with another operation
- Testing for division by zero while the divide is in progress
- Executing several different floating point operations simultaneously (The MIPS processor has separate floating point add, multiply, and divide units.)

The MIPS architecture does not have condition codes. Although this seems unusual compared with many other machines, this design actually improves performance. The architecture provides branch instructions that both test a condition and then branch. Thus, the compiler must generate only one instruction for conditional branches rather than the two instructions usually required (one to test for the condition, then another to perform the branch). Only comparisons that compare larger (or smaller) values between two registers, or a register and immediate value, cannot be handled this way. The compiler restructures most comparisons to avoid this case, thus decreasing average test time.

Instructions in a CISC processor often have widely varying execution times. This difference makes it hard to determine which of several alternative sequences is actually fastest. Because almost all RISC instructions take the same time, the optimizer can select the fastest sequence with relative ease.

Program Portability

To make a system useful, programs must be ported onto the processor. The use of UNIX operating systems and standardized languages such as FORTRAN has tremendously improved program portability. However, incorrect programs may have latent bugs that are masked by a naive compiler. Thus, an optimizing compiler tends to expose more of these problems. A naive FORTRAN compiler may assign all variables to memory locations, giving variables predictable initial values. An incorrect program that relies on these initial values will fail when an optimizing compiler assigns a variable to a fast register that tends to have unpredictable values.

An optimizer converts the program (as written) into one that is identical except that it executes faster. To do this, the optimizer must make assumptions about what the programmer intended. Often the programmer depends on experience with previous compiler implementations rather than the rules of the language.

The MIPS compiler suite provides a number of options to allow a program to run without modification in the presence of such common errors. This permits a program to be ported quickly, giving the programmer a choice as to when to correct the problem.

Traditionally, system vendors have added unique extensions to their language implementations. While these may be a boon to a programmer when writing the program, they can be a bane when it comes time to port the program. The MIPS compilers implement many industry-wide extensions to the standard languages to make them compatible with other implementations. An important set of extensions is the support of Digital's VAX FORTRAN extensions. Another is the inclusion of IBM's PL1 extensions to permit an independent software vendor to port a 1.8-million-line PL1 program to the MIPS architecture.

Wherever extensions are required, MIPS chooses proposed extensions for similar functions that are being considered by the standards committees. An example of this last situation is the need to represent hardware I/O structures when being referenced by an optimizing compiler. Consider an I/O device register that is used to define the status of the I/O device. The optimizing compiler would see multiple references to the address without intervening assignments. The compiler would cleverly (but incorrectly) optimize all references to a processor register. MIPS has added the key word volatile to indicate to the compiler that this variable changes in ways that the compiler cannot detect. This extension was recently incorporated into the current ANSI standard for C, but it was added to the MIPS C compiler four years ago.

Tools to Development

Fundamental to system development is a tool set that aids in compiling, debugging, performance tuning, and system construction (bring-up). MIPS' tool set includes those tools traditionally found in UNIX operating systems, as well as tools unique to MIPS.

The multiple front-end, common back-end construction of the MIPS compilers provides a consistent set of languages to the developer. Options and flags are the same across all languages. In fact, *cc* can usually be used to compile programs in the other languages. All languages share a common linkage convention that makes it easy to write or port programs written in two or more languages.

The UNIX tool *make* provides a convenient method of program development. This tool identifies which source modules of a program changed and recompiles only those modules. In recompiling, *make* provides the correct compilation options. It also provides the complete set of compilation options; for example, the debugging option can be enabled through a *make* target to troubleshoot a program. Later, the debugging options can be disabled, and higher levels of optimization can be specified in compiling the production version of the program. The *make* rules file, which accompanies the source program modules, determines how the modules are to be combined into the final run program. This prior determination eliminates the need for detailed written documentation or programmer support, making it simpler for developers to exchange source programs.

Crucial to efficient program development is a source-oriented debugger. MIPS provides an extended version of dbx which supports all programming languages. (ADA has a special debugger.) The debugger provides features such as printing variables as they change, or replaying a debugging session to a certain point before continuing the debug session. The user can view and edit the source, as well as see each statement as it is executed.

The debugger also lets the user view the program in both the high-level language in which it was written and in the generated machine language. Thus, the user can set a breakpoint on a specified statement or instruction. Full debugging facilities are available only when the debugging option is specified. However, the compiler (by default) maintains the line number tables in a compressed format in the load module even when the option is not specified. Retention of the line number tables permits partial debugging without the need to recompile the program using the debugging option. Line number information is kept for each instruction, permitting the instruction scheduler to move the instruction while keeping track of the line that originated it.

Specification of both debugging and optimization options can create conflicts. For example, it is possible for a bug to appear only when a higher level of optimization is specified. Moreover, optimizations such as register allocation can confuse the debugger, because variables are not at the locations that it assumes. To avoid this situation, the MIPS compilers disable any optimization that interferes with debugging when both debugging and optimization options are specified. To debug the problem that shows up only in optimized code, a special option permits both the debugger and optimization to be enabled. This technique requires the developer to use caution when inquiring about the contents of a variable.

To tune a program for optimal performance, the developer must learn where in a program the time is spent and why. Traditional UNIX systems provide a method called pc sampling. With this technique, the system must interrupt a program at regular intervals (usually 60 or 100 times per second) and increment a per-location counter. After enough sample points are taken, a pattern of execution time emerges. This method has a severe flaw because modern processors execute 10 to 20 million instructions per second; this means the number of samples is less than 1 in 100,000. The method requires a very long execution time to collect a statistically meaningful sample.

Although MIPS provides the pc sampling technique, it also furnishes a more exacting method. *Pixie* is a tool that takes an executable load module and prepares it for measurement by inserting a counter in every basic block. After running the instrumented program, the counters are dumped to a file for analysis by several programs. *Prof* displays tables of interesting information such as the following:

- Number of CPU cycles for each source line
- Number of times each function is called
- Average number of cycles in each call to a function

Figure 2 shows examples of *prof* output listings. A second program, *pixstats*, takes the same counts and displays information about the program in architectural terms, such as:

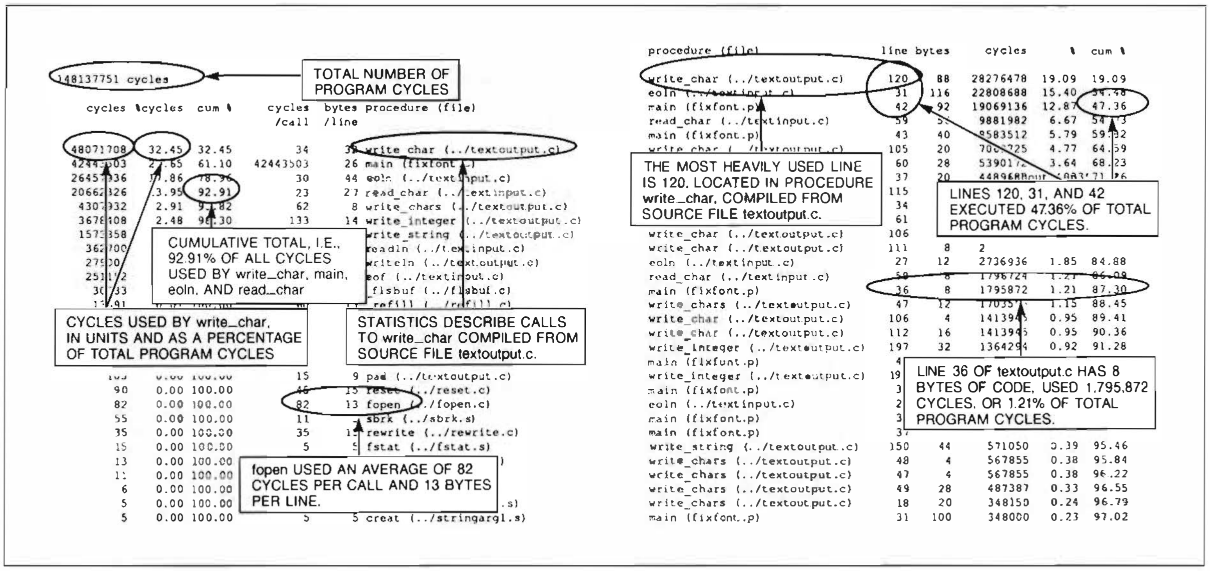
- Number of cycles used for each instruction
- Number of unused delay slots
- Number of FLOPS
- A general indication of cache locality

These two programs assume a perfect memory system, that is, no effect due to cache misses.

For a complete analysis, a cache simulator is also available that uses *pixie* to provide a memory address trace to the simulator to model the memory system. MIPS uses this technique to plan new machine designs. Each proposed change to the system requires a detailed simulation to exhibit its effect. When a MIPS designer is convinced that a balanced and optimal point has been found, implementation begins. Experience with this technique has shown an accuracy of better than 4 percent when comparing predicted performance to actual performance.

An Optimization to Improve Cache Performance

An area seldom addressed in compilers is the optimization of programs in memory to improve cache hit rates. Modern microprocessor performance has been increasing faster than the supporting memory systems. Taking advantage of this higher performance without introducing costly memories has required the use of small and high-speed cache memory. Cache memory contains recently used instructions and data. Hardware substitutes cache memory if the desired word is in the cache. This substitution is invisible (other than performance improvements) to the program. Each main memory location must share a cache location with other memory locations because the cache is smaller than main memory. For a cache to be effective, it must contain enough of the program or data to ensure multiple reuse of the instructions or data.


TOTAL NUMBER OF PROGRAM CYCLES
CUMULATIVE TOTAL, I.E., 92.91% OF ALL CYCLES USED BY write_char, main, eoln, AND read_char
CYCLES USED BY write_char, IN UNITS AND AS A PERCENTAGE OF TOTAL PROGRAM CYCLES
STATISTICS DESCRIBE CALLS TO write_char COMPILED FROM SOURCE FILE textoutput.c.
fopen USED AN AVERAGE OF 82 CYCLES PER CALL AND 13 BYTES PER LINE.
THE MOST HEAVILY USED LINE IS 120, LOCATED IN PROCEDURE write_char, COMPILED FROM SOURCE FILE textoutput.c.
LINES 120, 31, AND 42 EXECUTED 47.36% OF TOTAL PROGRAM CYCLES.
LINE 36 OF textoutput.c HAS 8 BYTES OF CODE, USED 1,795,872 CYCLES, OR 1.21% OF TOTAL PROGRAM CYCLES.

```
148137751 cycles

  cycles %cycles  cum %     cycles  bytes procedure (file)
                             /call  /line

48071708   32.45  32.45         34     38 write_char (../textoutput.c)
42443503   28.65  61.10   42443503     26 main (fixfont.p)
26457336   17.86  78.96         30     44 eoln (../textinput.c)
20662326   13.95  92.91         23     27 read_char (../textinput.c)
 4307932    2.91  95.82         62      8 write_chars (../textoutput.c)
 3678408    2.48  98.30        133     14 write_integer (../textoutput.c)
 1573358                                  write_string (../textoutput.c)
  362700                                  readln (../textinput.c)
   27900                                  writeln (../textoutput.c)
   25112                                  eof (../textinput.c)
    3033                                  _flsbuf (../flsbuf.c)
      91                                  _refill (../refill.c)
                                15      9 pad (../textoutput.c)
      90    0.00 100.00         46     15 reset (../reset.c)
      82    0.00 100.00         82     13 fopen (../fopen.c)
      55    0.00 100.00         11        sbrk (../sbrk.s)
      35    0.00 100.00         35     1  rewrite (../rewrite.c)
      15    0.00 100.00          5      5 fstat (../fstat.s)
      13    0.00 100.00
      11    0.00 100.00
       6    0.00 100.00
       5    0.00 100.00                   .s)
       5    0.00 100.00          5      5 creat (../stringarg1.s)
```

```
procedure (file)                 line bytes    cycles      %    cum %

write_char (../textoutput.c)      120    88  28276478  19.09  19.09
eoln (../textinput.c)              31   116  22808688  15.40  34.48
main (fixfont.p)                   42    92  19069136  12.87  47.36
read_char (../textinput.c)         59        9881982   6.67  54.03
main (fixfont.p)                   43    40   8583512   5.79  59.82
write_char (../textoutput.c)      105    20   7060725   4.77  64.59
                                   60    28   5390172   3.64  68.23
                                   37    20   4489688
                                  115
                                   34
                                   61
write_char (../textoutput.c)      106
write_char (../textoutput.c)      111     8   2
eoln (../textinput.c)              27    12   2736936   1.85  84.88
read_char (../textinput.c)         59     8   1796724   1.21  86.09
main (fixfont.p)                   36     8   1795872   1.21  87.30
write_chars (../textoutput.c)      47    12   1703572   1.15  88.45
write_char (../textoutput.c)      106     4   1413945   0.95  89.41
write_char (../textoutput.c)      112    16   1413945   0.95  90.36
write_integer (../textoutput.c)   197    32   1364294   0.92  91.28
main (fixfunt.p)                    4
write_integer (../textoutput.c)    19
main (fixfont.p)                    3
eoln (../textinput.c)               2
main (fixfont.p)                    3
main (fixfont.p)                   37
write_string (../textoutput.c)    150    44    571050   0.39  95.46
write_chars (../textoutput.c)      48     4    567855   0.38  95.84
write_chars (../textoutput.c)      47     4    567855   0.38  96.22
write_chars (../textoutput.c)      49    28    487387   0.33  96.55
write_chars (../textoutput.c)      18    20    348150   0.24  96.79
main (fixfont.p)                   31   100    348000   0.23  97.02
```

Figure 2 Examples of prof *Output Listings*

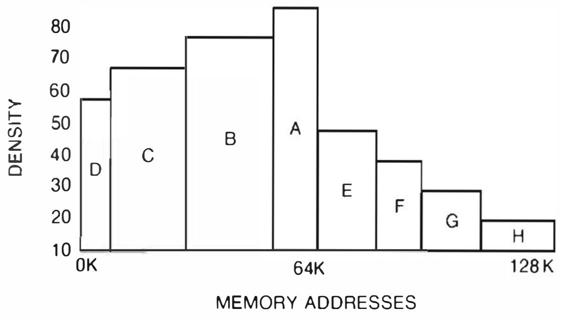
A valuable optimization is to position the program in memory so that the most frequently used memory locations never compete for the same cache locations. MIPS has built a tool called *cord* that rearranges the program to improve instruction cache utilization. This tool is made possible through the existence of precise profiling tools.

To use *cord,* the programmer compiles the program in the usual way. *Pixie* is used to add counters to the program for each basic block. After executing the instrumented program, *prof* is run with an option that creates a file containing dynamic execution information. That file is given to *cord,* along with the original executable module.

Cord computes the density for each function (procedure). The density is defined as the average number of cycles executed by each instruction in the function. Figure 3 is an example of eight functions, their sizes, cycle counts, and density. *Cord* then creates a new executable module after sorting the functions according to density. Figure 3 also shows the order of the functions in the rearranged program.

This sort improves cache hit rate because it places the functions that use the most cycles in memory so they do not compete for the same cache location as other frequently executing functions. The effectiveness of sort is helped by two other features in the MIPS architecture. First, the caches are direct-mapped to memory so that each memory location corresponds to a single cache location. Second, the MIPS operating system places virtual pages in physical memory so that adjacent virtual pages map to adjacent cache pages. As a result, the placement of functions by *cord* has very predictable effects on the cache.

Figure 3 also shows the arrangement in an unexpected way. Rather than placing the densest function (A) at the beginning of memory, it is placed farthest from the start of the cache. This arrange-



KEY

NAME	SIZE	CYCLES	DENSITY
A	12K	960K	80
B	24K	1680K	70
C	20K	1200K	60
D	8K	400K	50
E	16K	640K	40
F	12K	360K	30
G	16K	320K	20
H	20K	200K	10

Figure 3 Cache Performance Improved by cord

ment has the effect of making the densest function share cache locations with the function 128 kilobytes (twice the cache size) away (H). *Cord* improves performance by as much as 20 percent to 30 percent on programs exceeding the size cache. It works solely by improving the efficiency of the instruction cache. Methods to improve data cache accesses are not available yet because of the more random nature of data accesses and difficulty in getting accurate data reference information.

An advanced architectural simulator, Sable models (in C) the processor, including TLB, pipeline, register set, and system design, incorporating the cache subsystem, main memory, and I/O interface. Developers can customize Sable for a unique system design. Sable has been used routinely at MIPS to bring up the UNIX operating system before hardware is available. Simulation with Sable assures that the software is reliable and performs optimally when the hardware is actually available. Sable can be used with the debugger to provide full symbolic debugging in the simulation environment. Also, Sable can provide the same address traces that *pixie* provides to analyze operating system performance. After a system has been brought up using Sable, other tools can assist in constructing the system on the real hardware. A simple debug monitor is available to work with the symbolic debugger to provide a symbolic debugging environment on the real hardware.

Trends in Compilers

Compilers are taking better advantage of the parallelism in today's RISC processors. Evidence of this can be seen in the scheduling of instructions to capitalize on load and branch delays and multiple floating point units. This trend will continue as it becomes feasible to build effective multiprocessor systems. In this area, compilers that will partition a problem across multiple processors — each performing a portion of the iterations of a loop — will be seen. A major challenge will be to find ways to use this kind of parallelism in nonengineering problems. These problems tend not to be loop intensive and will require a breakthrough in compiler technology for automatic parallelism.

Over the past five to ten years, the programming language C has come to the forefront as a major systems language. While C offers many advantages, it requires a user to deal with fairly primitive structures rather than abstractions. C++ will offer much of the flexibility of C with the added capability of data abstraction.

It is expected that future compilers will take advantage of optimizations that reduce cache misses. These optimizations include loop interchange, which reorders the accesses to an array to improve locality of data references, and software pipelining, which takes better advantage of overlapping memory accesses and computation.

Further Readings

The Digital Technical Journal *publishes papers that explore the technological foundations of Digital's major products. Each Journal focuses on at least one product area and presents a compilation of papers written by the engineers who developed the product. The content for the Journal is selected by the Journal Advisory Board.*

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

VAX 8600 Processor
Vol. 1, No. 1, August 1985

MicroVAX II System
Vol. 1, No. 2, March 1986

Networking Products
Vol. 1, No. 3, September 1986

VAX 8800 Family
Vol. 1, No. 4, February 1987

VAXcluster Systems
Vol. 1, No. 5, September 1987

Software Productivity Tools
Vol. 1, No. 6, February 1988

CVAX-based Systems
Vol. 1, No. 7, August 1988

Storage Technology
Vol. 1, No. 8, February 1989

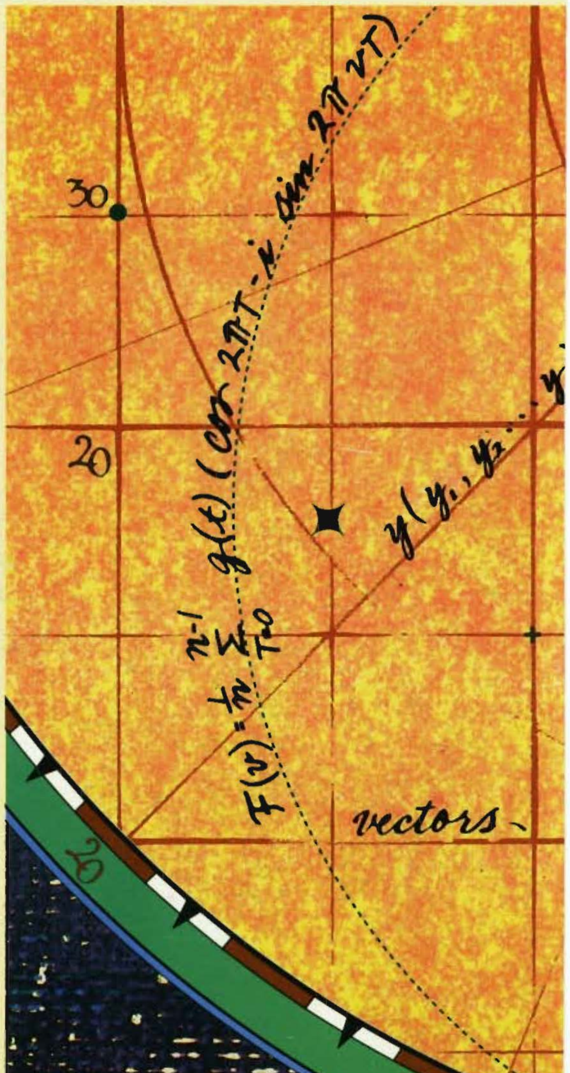
Distributed Systems
Vol. 1, No. 9, June 1989

Compound Document Architecture
Vol. 2, No. 1, Winter 1990

See the inside front cover of this issue for subscription information.

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

digital™



ISSN 0898-901X