Logical Verification of the NVAX CPU Chip Design

By Walker Anderson

1 Abstract

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

2 Introduction

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

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

3 Team Organization and Approach

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

Logical Verification of the NVAX CPU Chip Design

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

1. Plan tests for a function.

2. Review those plans with the design and architecture teams.

3. Implement the tests.

4. Review the actual testing with the design and architecture teams.

5. Implement any additional testing that was deemed necessary.

4 Simulation and Modeling Methodology

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

Behavioral Models

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

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

The box-level models were then integrated into a full-chip behavioral model, which also included a backup cache model, a main memory model, and models to emulate the effects of several system configurations. The pseudosystem models did not model any one specific target system configuration but could be set up to operate effectively like any target

system configuration or in a way that exercised the chip more intensely than any target system would. Available early in the project, this model was the primary vehicle for logical verification until the schematics-derived, full-chip, in-house CHANGO model was created. The full-chip behavioral model could simulate approximately 13 cycles per VAX VMS CPU second and was used to simulate about one billion CPU cycles.

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

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

Schematics-derived Models

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

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

5 Pseudorandom Exercisers

Early in the project, it became apparent that, given the limited number of engineers, the short schedule, and the complexity of the NVAX chip design, a strategy of developing and simulating all conceivable implementation-specific test cases would be ineffective. This strategy would have required the engineers to implement tedious, handcrafted tests. Instead, the verification team adopted a strategy that depended heavily on the use of directed, pseudorandom tests referred to as exercisers. This strategy generated and ran many more interesting test cases than would ever have been conceived by the verification and design engineers themselves.

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

1. Set up the test case.

2. Simulate the test case on either the behavioral or the CHANGO model.

3. Execute the test program on a VAX reference machine.

4. Analyze the simulation and accumulate data.

5. Check the results for failure.

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

Setup

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

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

SEGUE provides a notation that allows the user to specify sets of possible text expansions. Elements of these sets can be selected either pseudorandomly or exhaustively, and the user can specify the weighting desired for the selection process. For example, a hierarchy of SEGUE templates typically comprised three levels. At the lowest level, a SEGUE template was created to select pseudorandomly a VAX opcode, and another template was created to select a specifier, i.e., operand. At an intermediate level, the verification engineers created templates

that called the lowest-level templates to generate short sequences of instructions to cause various events to occur, e.g., a cache miss, an invalidate from the system model, or a copy of register file contents to memory. At the highest level, these intermediate-level templates were selected pseudorandomly with varied weighting to generate a complete test program.

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

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

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

Simulation and VAX Reference Execution

After assembling and linking the test program, it was loaded into modeled memory, and its execution was simulated on either the behavioral or the CHANGO model. As the test program simulation took place, a simulation log file and a binary-format file, which contained a trace of the state of the pins and various internal signals, were created. As the exerciser test programs executed, various VAX architectural state information was written periodically to a region of modeled memory referred to as the

dump area. When the simulated execution of the test program completed, the contents of the dump area were stored in a file. Also, the test program was executed under the VMS operating system running on a VAX computer used

as a reference machine. At the end of execution, the contents of the memory dump area were stored in another file.

Analysis

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

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

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

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

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

- To indicate significant areas of the design where coverage may have been deficient

- To determine how the exercisers might be adjusted to become more effective or thorough, or to focus on a particular low-level function of the chip design

Pass/Fail Checking

Several checking mechanisms were employed to determine whether tests passed or failed. The SAVES tool was used to check for correct behavior of the

design, especially where correct behavior was difficult to observe at a VAX architectural level. For example, the verification engineers used SAVES to check the proper functioning of performance-enhancing features such as branch prediction logic, pipelines, and caches.

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

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

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

Other Aspects of the Exercisers

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

6 Focused Tests

Several focused tests were generated to supplement the exercisers. These were necessary to test implementation-specific aspects of the design that could not be checked by comparing results against a VAX reference machine. In some cases, an exerciser could have been used to test a particular function, but the verification engineer judged it easier to hand-code a focused test program than to control an exerciser in order to accomplish the testing. Focused tests were necessary and particularly challenging to

create and maintain when very precise timing of events was required to test a certain scenario of chip operation. This timing could be achieved only by handcrafting an assembly language test and running it under carefully controlled simulation conditions.

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

7 Other Tests

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

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

Small portions of the HCORE suite were used as a nightly model regression test. In general, very little regression testing of the NVAX models took place; the team believed that using computing resources to run pseudorandom exercisers and other new tests was of more value to the verification effort than consuming resources with extensive, frequent regression testing. Consequently, the entire HCORE suite was run at only a few key checkpoints during the project.

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

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

8 Schematic Verification

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

Box-level CHANGO Simulation

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

Full-chip CHANGO Simulation

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

Simulation of the VMS Boot Process

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

was created on an existing VAX system. Each block of the disk was copied to a VMS file, which was then used as the system disk image during simulation.

A disk model with a simple programming interface and a direct memory access (DMA) capability was added to the simulation environment of the full-chip CHANGO model. The disk model read blocks from the system disk image file, and wrote data to a small cache of internally maintained disk blocks. To accelerate disk transfers, the disk model would examine cache state and use the system bus for the disk transfers only when the data was present in the cache and required a cache invalidate or write back. In other cases, the data was transferred directly into the memory subsystem in zero simulated time.

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

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

9 Prototype Chip Verification

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

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

The rigorous testing that continued on prototype systems revealed a few logical bugs which had gone undetected during simulated verification. Typically, information about a bug was collected on the prototype system, and then the failing scenario was reproduced on the behavioral model, where

the scenario could be analyzed and better understood. The chip-internal signals were extremely difficult to observe, but a 12-bit, parallel port allowed access to one of eight sets of signals from various sections of the

chip. The ability to monitor the control store address bus by means of this parallel port proved to be an essential debugging feature.

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

10 Bug Tracking and Design Release

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

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

11 Results and Conclusions

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

- Little or no testing of the function had been performed.

- Testing of the function was performed just before release, in a hurried manner.

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

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

1. One simple bug was detected by running the HCORE test suite on the prototype system with the floating-point unit (F-box) disabled. This bug could have been found in the same way through simulation, but the test suite was not run as a final regression test with the F-box disabled. In general, focused tests like HCORE were not run with varied chip/system configurations. The verification team concluded that all focused tests should be run with different chip/system configurations. At the minimum, a configuration that disables all possible functions should be tested.

2. Another bug was discovered because the CPU chip generated spurious writes to memory in the prototype system. The exercisers probably did generate the conditions necessary to evoke this bug; however, the spurious writes went unnoticed. It is extremely difficult to verify that a machine does everything it is supposed to do and nothing more. Additional assertion checkers or monitors in the models might detect such bugs in the future.

3. A third bug was evoked when a prototype system exerciser executed a translation buffer invalidate all (TBIA) instruction under certain conditions. On a real system, the TBIA instruction is used only by the operating system. In our verification effort, the TBIA instruction was little used by the exercisers that were simulated. Operations that are performed only by the operating system should not be underemphasized in exercisers.

4. One first-pass bug was related to the halt interrupt, which is used only during debugging operations. The halt interrupt received minimal testing and was not tested at all in any type of exerciser. Discovering this bug was especially annoying because a similar bug had escaped detection by the initial logical verification effort for a previous

VAX implementation. This turn of events reinforces the belief that there is value in reviewing the escaped bug lists from other projects. Also, during the verification effort, there seemed to be a natural, but erroneous, tendency to undertest functions used infrequently or not at all during normal system operation. Such functions sometimes require

extra attention, because they may be quite complex and may have been given less careful thought during the design process.

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

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

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

12 Acknowledgements

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

13 References

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

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

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

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

14 Biography

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

15 Trademarks

The following are trademarks of Digital Equipment Corporation: Digital, VAX, VAX 6000, and VMS.

Copyright 1992 Digital Equipment Corporation. Forwarding and copying of this article is permitted for personal and educational purposes without fee provided that Digital Equipment Corporation's copyright is retained with the article and that the content is not modified. This article is not to be distributed for commercial advantage. Abstracting with credit of Digital Equipment Corporation's authorship is permitted. All rights reserved.