Design Methodologies for the PA 7100LC Microprocessor

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

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

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

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

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

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

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

Synthesis and Routing Methodology

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

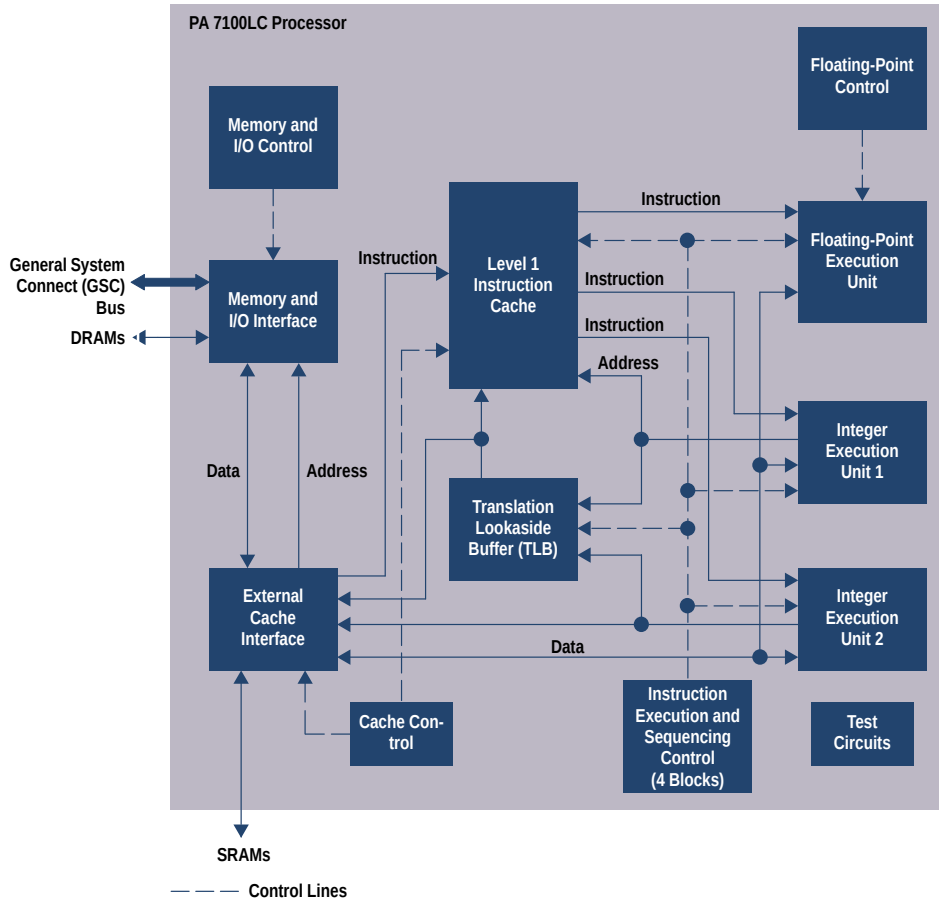

Fig. 1. A simplified block diagram of the PA 7100LC showing the relationship between the control blocks and the other major blocks in the processor. The instruction execution and pipeline sequencing control block consists of four separate blocks that are physically distinct but highly interconnected.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

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

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

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

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

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

Verification Methodology

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

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

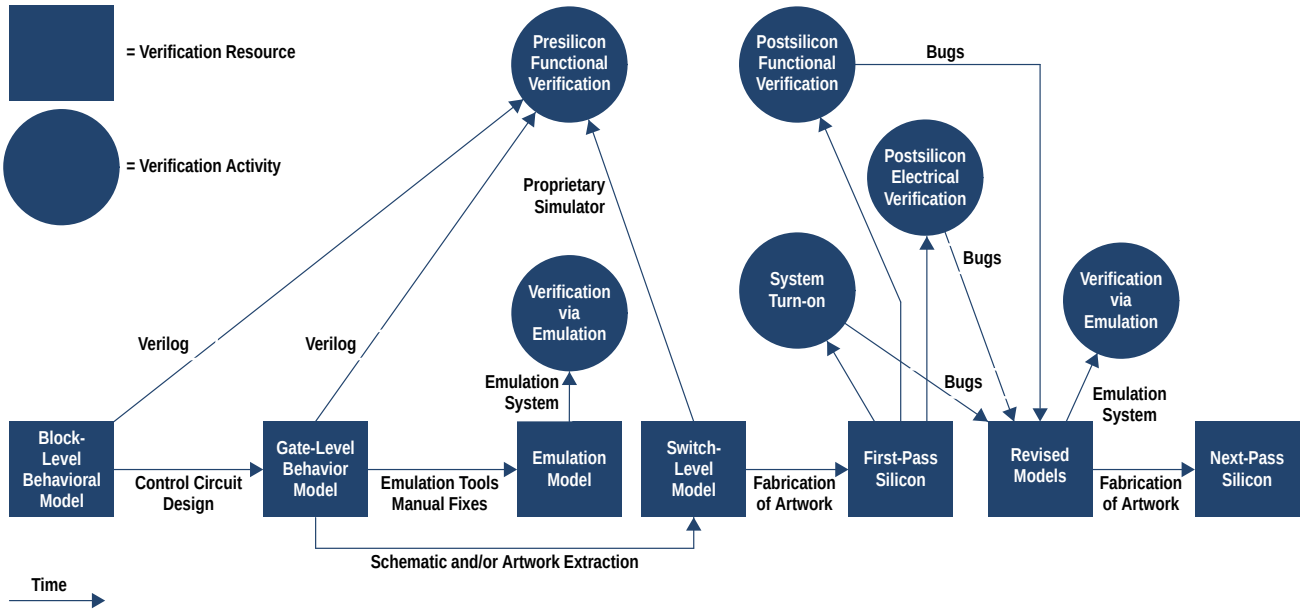

Fig. 2. Overview of the functional verification process.

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

Presilicon verification activities included:

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

Postsilicon verification activities included:

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

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

A New Strategy

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Presilicon Functional Verification

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Postsilicon Functional Verification

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In-Circuit Emulation

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

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

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

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

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

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

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

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

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

Emulation Methodology

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

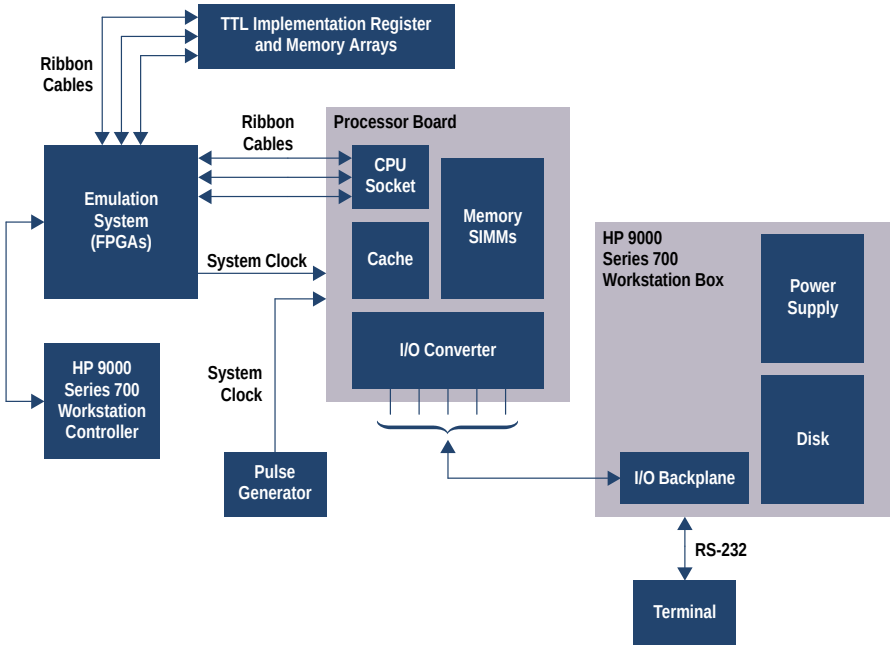

Fig. 3. Emulation setup for the PA 7100LC.

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

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

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

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

From our emulation efforts we learned the following:

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

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

Postsilicon Electrical Verification

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

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

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

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

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

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

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

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

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

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

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

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

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

Debug and Test

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

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

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

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

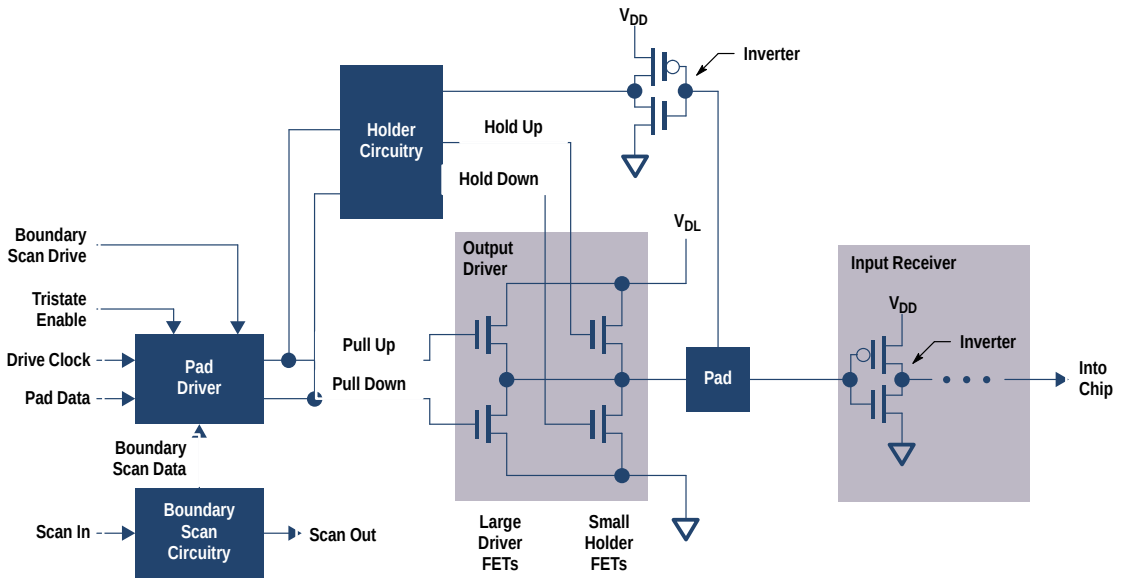

Fig. 4. Simplified diagram of a PA 7100LC I/O driver. Static current can flow from V_{DD} to ground in the inverters if the pad is not driven to V_{DD} or ground. For example, if the pad driver drives a one, the pad would be driven to 3.3V (V_{DL}), which would cause static current to flow, invalidating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

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

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

I_{DDQ} Implementation

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

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

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

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

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

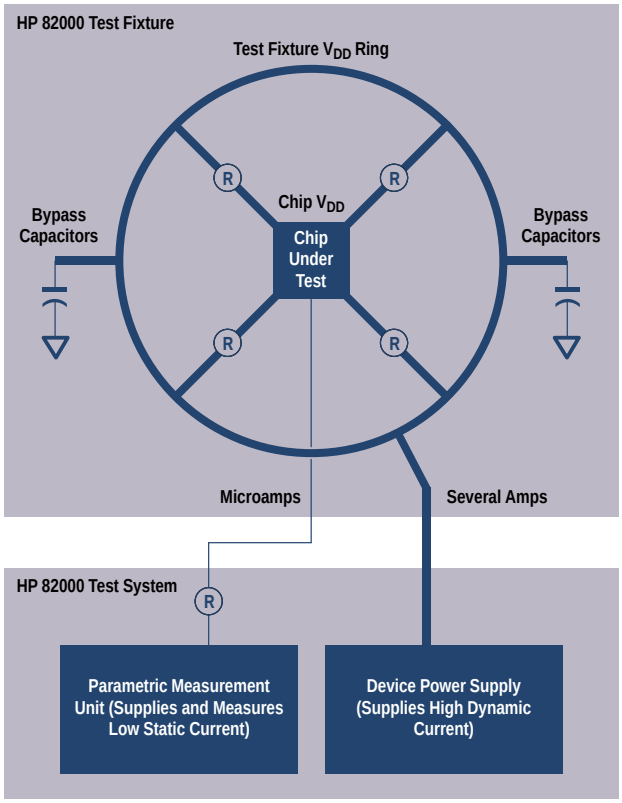

Fig. 5. I_{DDQ} measurement setup.

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

I_{DDQ} Measurement

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

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

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

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

Sample-on-the-Fly Testing

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

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

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

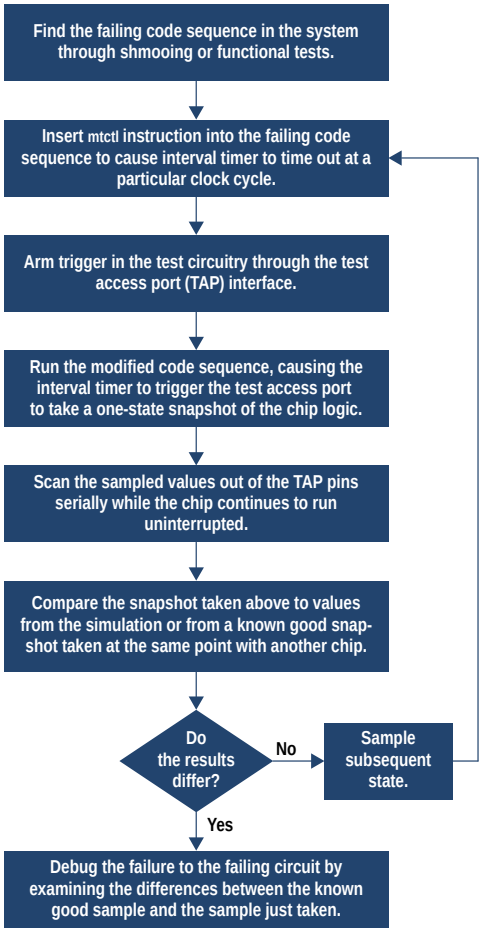

Fig. 6. Sample-on-the-fly testing process.

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

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

Debug Mode

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

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

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

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

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

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

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

Conclusion

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

References

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

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

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

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