Verification, Characterization, and Debugging of the HP PA 7200 Processor

To guarantee a high-quality product the HP PA 7200 CPU chip was subjected to functional and electrical verification. This article describes the testing methods, the debugging tools and approaches, and the impact of the interactions between the chip design and the IC fabrication process.

by Thomas B. Alexander, Kent A. Dickey, David N. Goldberg, Ross V. La Fetra, James R. McGee, Nazeem Noordeen, and Akshya Prakash

The complexity of digital VLSI chips has grown dramatically in recent years. Rapid advances in integrated circuit process technology have led to ever-increasing densities, which have enabled designers to design more and more functionality into a single chip. Electrically, the operating frequency of these VLSI chips has also gone up significantly. This has been a result of the increased speed of the transistors (CMOS transistors are commonly called FETs, for field effect transistors) and the fact that the circuits are closer to each other than before. All this has had tremendous benefits in terms of performance, size, and reliability.

The increased complexity of the VLSI chips has created new and more complicated problems. Many sophisticated techniques and tools are being developed to deal with this new set of problems. Nowhere is this better illustrated than with CPUs, especially in design verification, both functional and electrical. While design has always been the focus of attention, verification has now become a very challenging and critical task. In fact, verification activities now consume more time and resources than design and are the real limiters of time to market.

On the functional side, for many years now it has been impossible to come even close to a complete check of all possible states of the chip. The challenge is to do intelligent verification (both presilicon and postsilicon) that gives very high confidence that the design is correct and that the final customer will not see any problem. On the electrical side, the challenge has been to find the weak links in the design by creating the right set of environments and tests that are most likely to expose failures. The increased complexity of the VLSI chips has also made isolation of a failure down to the exact FET or signal an increasingly difficult task.

This paper presents the verification methodology, techniques, and tools that were used on the HP PA 7200 CPU to guarantee a high-quality product. Fig. 1 shows the PA 7200 CPU in its pin-grid array package. The paper describes the functional and electrical verification of the PA 7200 as well as the testing methods, the debugging tools and approaches, and the impact of the interactions between the chip design and the IC fabrication process.

Functional Verification

The PA 7200 CPU underwent intensive design verification efforts to ensure the quality and correctness of its functionality. These verification efforts were an integral part of the CPU design process. Verification was performed at all stages in the design, and each stage imposed its own constraints on the testing possible. There were two main stages of functional verification: the presilicon implementation stage and the postsilicon prototyping stage.

Presilicon Functional Verification

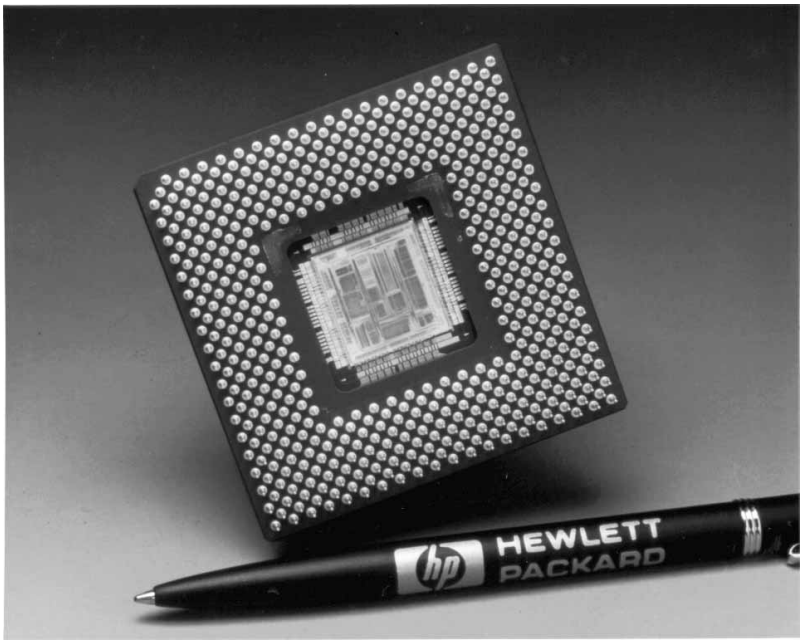
Since the design of the PA 7200 was based upon the PA 7100 CPU, we chose to use the same modeling language and proprietary simulator to model and verify its design. During the implementation stage a detailed simulation model was built to verify the correctness of the design. Early in the implementation stage, software behavioral models were used to represent portions of the design and these were incrementally replaced by detailed models. A switch-level model was also used late in the implementation stage to ensure equivalence between the actual design implementation and the simulation model. This switch-level model was extracted from the physical design's FET artwork netlists and was used in the final regression testing of the design.

Test cases were written to provide thorough functional coverage of the simulation model. The test case strategy for the PA 7200 was to:

- Run all existing cases derived for previous generations of PA-RISC processors
- Run all architectural verification programs (AVPs)

- Write and run test suites and cases directed at specific functional areas of the implementation, including the newly designed multiprocessor bus interface (Runway bus) and its control unit, the assist cache, the dual-issue control unit, and other unique functionality
- Generate and run focused random test cases that thoroughly stress and vary processor state, cache state, multiprocessor activities, and timing conditions in the various functional units of the processor.

Fig. 1. *The PA 7200 CPU in its pin-grid array package, with the lid removed to reveal the chip.*



Existing legacy test cases and AVPs targeted for other generations of PA-RISC processors often had to be converted or redirected in a sensible way to yield interesting cases on the PA 7200. Additional test cases were generated to create complex interactions between the CPU functional units, external bus events, and the system state. An internally developed automated test case generation program allowed verification engineers to generate thousands of interesting cases that focused upon and stressed particular CPU units or functions over a variety of normal, unusual, and boundary conditions. In addition, many specific cases were generated by hand to achieve exact timing and logical conditions. Macros were written and a macro preprocessor was used to facilitate high productivity in generating test case conditions.

All test code was run on the PA 7200 CPU model and on a PA-RISC architectural simulator and the results were compared on an instruction-by instruction basis. The test case generation and simulation process is shown in Fig. 2. A PA 7200-specific version of the PA-RISC architectural simulator was developed to provide high coverage in the areas of multiprocessor-specific conditions, ordering rules, cache move-in, move-out rules, and cache coherence. Some portions of the internal CPU control model were also compared with the architectural simulator to allow proper tracking and checking of implementation-specific actions. Since the PA 7200 was designed to support several processor-to-system-bus frequency ratios, the simulation environment was built to facilitate running tests at various ratios.

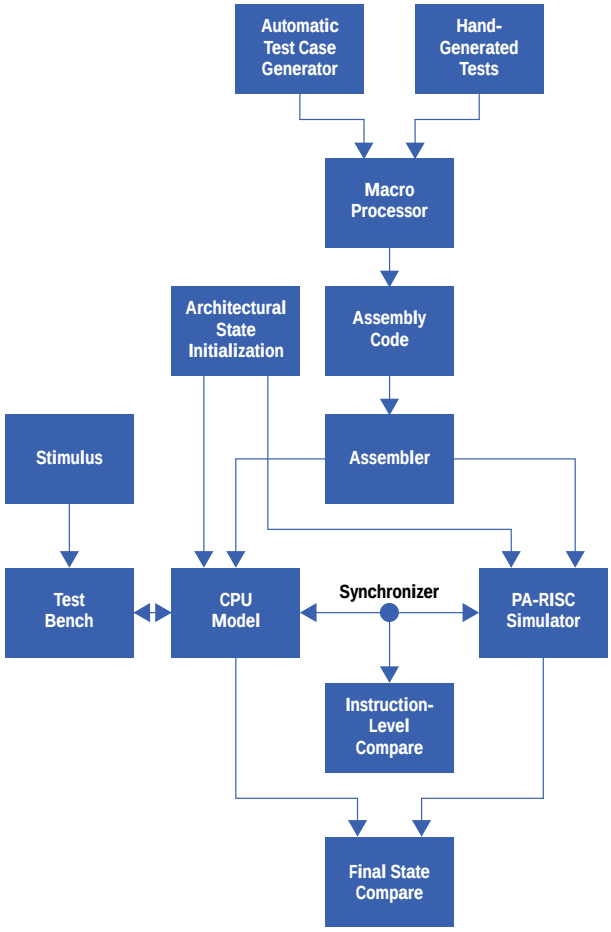
The architected state of the CPU and simulator, including architected registers, caches, and TLBs, was initialized at model startup time. Traces of instruction execution and relevant architected state from the CPU model and from the PA-RISC simulator were compared. These traces included disassembled code, affected register values, and relevant load/store or address information, providing an effective guide for debugging problems.

A test bench approach was used to model other system bus components and to verify proper system and multiprocessor behavior, including adherence to the bus protocol. The test bench accepted test case stimulus to stimulate and check proper CPU operation. Multiprocessor effects on the caches and the pipeline of the CPU being tested were checked in detail both by instruction execution comparison and by final state comparison of architected registers, caches, and TLBs.

The bulk of the testing during the implementation stage entailed running assembly language test vectors on the simulation model. The principal limitation of this stage was the limited execution speed of the simulation model.

As components of the simulation model became defined and individually tested, they were combined into increasingly larger components until a combined simulation model was built for the entire computer system including processors, memory, and I/O.

Fig. 2. *PA 7200 test case generation and simulation process.*



An effort was also made to evaluate the test case coverage of processor control logic to ensure that we had high coverage of the functional units with normal and corner-case conditions. During our regressions of functional simulation, the simulation model was instrumented to provide coverage data, which was postprocessed to yield coverage metrics on the control logic.

This verification effort consumed many engineering months in test bench development, test case generation, and test checking. Billions of simulation cycles were run on scores of high-performance technical workstations during nights and weekends in several different geographical locations. The result of this effort is a high-quality CPU that booted its operating system and enabled postsilicon functional and electrical verification efforts soon after the first silicon parts were received.

This simulation approach also facilitated a productive debugging and regression testing environment for testing fixes. Specifically, when making a correction to the CPU, the simulation environment allowed the verification team to run regression suites that stressed the faulty area, providing more complete simulation coverage of the problem.

Postsilicon Functional Verification

Despite the massive presilicon testing, there are always bugs that are found once the first hardware becomes available. Bugs that affect all chips regardless of temperature, voltage, or frequency are termed functional bugs. Bugs that are made worse by environmental conditions or do not occur in all chips are termed electrical problems, and the test strategy for finding those problems is detailed in the section on electrical verification.

Testing machines as complex as the HP 9000 J-class and K-class systems, the first systems to use the PA 7200, was a large effort involving dozens of people testing specific areas. This section will describe how the processor design laboratory created processor-focused tests to find processor bugs. Many other people contributed to testing other portions of the systems with intentional overlap of testing coverage to ensure high quality.

Because of the complexity of modern processor chips, not all bugs are found in presilicon testing. The processor is so complicated that adequate testing would take years running at operational speed to hit all the interesting test cases. Presilicon testing is orders of magnitude too slow to hit that many cases in a reasonable amount of time. Thus, when presilicon testing stops finding bugs, the chip is manufactured and postsilicon testing commences. However, finding bugs is not as simple as just turning the power on and waiting for the bugs to appear. One problem is deciding what tests to run to look for failures. Poorly written tests will not find bugs that customers might find. Another problem is debugging failures to

their root causes in a timely manner. Knowing a problem exists is a great start, but sometimes discovering exactly what has gone wrong in such complex systems can be very difficult. Postsilicon testing loses much of the observability of processor state that was easily obtained in the simulation environment.

To provide high coverage of design features, three testing tools were prepared to stress the hardware. These tools were software programs used to create tests to run on the prototype machines. Each tool had its own focus and intended overlap with other tools to improve coverage. All tools had a proven track record from running on previous systems successfully. To ensure adequate testing, two tools were heavily modified to stress new features in the PA 7200.

All of the tools had some features in common. They all ran standalone independently on prototype machines under a small operating system. Because they did not run under the HP-UX* operating system, much better machine control could be achieved. In addition, not needing the HP-UX operating system decoupled hardware debugging from the software schedule and let the hardware laboratory find bugs in a timely manner. (Later in the verification process, HP-UX-based system testing is performed to ensure thorough coverage. However, the team did not rely on this to find hardware problems.) All hardware test tools also had the ability to generate their own code sequences and were all self-checking. Often these code sequences were randomly generated, but some tools supported hand-coded tests to stress a particular hardware feature.

Uniprocessor Testing

Even though PA 7200 systems support up to four processors, it is desirable to debug any uniprocessor problems before testing for the much more complex multiprocessor bugs. The first tool was leveraged from the PA 7100LC effort to provide known good coverage of uniprocessor functionality.

This tool operated by generating sequences of pseudorandom instructions on a known good machine, like an HP 9000 Model 735 workstation. On this reference machine, a simulator would calculate the correct expected values and then create a test to be run on the prototype hardware. This test would set up hundreds of various initial states and run the prepared sequence. Each time it ran the sequence, the tool would determine if it got the correct result and display any differences. Since much of the work was done on another machine to prepare the correct answer, this tool was very robust and was a good initial bring-up vehicle. It also could run its sequences very quickly and give good coverage in a short amount of time. However, uniprocessor bugs ramped down very quickly, and so this tool was used much less after initial bring-up.

Multiprocessor Testing

The verification team was especially concerned with multiprocessor bugs, since experience indicated that they are much more difficult to find and debug than uniprocessor cases. These complex bugs were often found later in the project. For this reason, the two other tools used were heavily modified to enhance PA 7200 testing for multiprocessor corner cases.

The first multiprocessor-focused tool attempted to do exhaustive testing of the effects of various bus transactions interacting with a test sequence. The interference transactions were fixed but were chosen to hit all the cases that were considered interesting. The test sequence could be randomly generated or written manually to stress a particular area of the processor.

To determine if a test operated properly, the tool would run the test sequence once without any interference from other processors. It would capture the machine state after this run (register, cache, memory) and save it as the reference results. The tool did not need to know what the test was doing at all—it simply logged whatever result it got at the end. To test multiprocessor interference transactions, the tool would then arrange to have other processors try all combinations of interesting transactions selected to interact with the sequence. This was accomplished by running the test in a loop with the interference transactions being moved through every possible cycle in a predetermined timing window. This exhaustive testing of interference transactions against a code sequence provided known good coverage of certain troublesome areas. When there were failures, many useful debugging clues were available regarding which cases passed and which cases failed to help in tracking down the bug.

The main deficiency of this tool was that it relied on high uniprocessor functionality. If there were bugs that could affect the reference run, the tool would not be able to detect them. Thus, this tool could not run until uniprocessor functionality was considered stable. As it turned out, the initial PA 7200 silicon had very high uniprocessor functionality and so testing began on initial silicon. One advantage of this tool over the uniprocessor tool was that it could run for an unlimited amount of time on the prototype hardware and generate test cases on its own. This ability made running this tool much simpler than the uniprocessor tool.

The final tool was the backbone of the functional verification effort. In many ways, this tool merged the good ideas from other tools into one program to provide high coverage with ease of use. This tool generated sequences of pseudorandom instructions on each processor, ran the sequences across all the processors simultaneously, and then checked itself. It calculated the correct results as it created the instruction sequences, so it could run unattended for an unlimited time. The sequences and interactions it could generate were much more stressful than the other tools. Much of this ability came from the effort put into expanding this tool, but some of it came from basic design decisions.

This tool relied completely on pseudorandomly generated code sequences to find its bugs. The tool took probabilities from an input file which directed the tool to stress certain areas. These focused tests enhanced coverage of certain processor

functionality such as the new data prefetching ability of the PA 7200. Almost any parameter that could be changed was changed constantly by this tool to hit cases beyond what the verification team could think of. Having almost no fixed parameters allowed this tool to hit bugs that no other tool or test has ever hit.

This final tool received additional modifications to test DMA (direct memory access) between peripheral cards and memory. The new Runway bus added new bus protocols involving I/O transactions, which the processor needed to obey to ensure system correctness. DMA was used to activate these bus protocols to verify that the PA 7200 operated properly. To make sure these extra cases were well-covered, DMA was performed using various peripheral devices while the processor testing was done. This extra testing was worth the investment since several bugs were found that might not have been caught otherwise.

The postsilicon verification effort was considered successful because the team found almost every bug before other groups and could communicate workarounds for hardware problems to keep them from affecting software schedules. The operating system testing actually found very few processor bugs, and all serious bugs were found by the postsilicon hardware verification team. Some of the later hardware bugs found may never be encountered by the current operating system because the compilers and the operating system are limited in the code sequences they emit. However, the hardware has been verified to the point that if a future compiler or operating system uses a feature not used before, it can in all likelihood do so without encountering a bug.

Electrical Verification

Electrical verification of a VLSI device is performed to guarantee that when the product is shipped to a customer, the device will function properly over the entire operating region specified for the product. The operating region variables include ambient temperature, power supply voltages, and the clock frequency of the VLSI device. In addition, electrical verification must account for integrated circuit fabrication process variation over the life of the device. Testing for sensitivities to these variables and improving the design to account for them improves fabrication yield and increases the margin of the product. This section describes the various electrical verification activities performed for the PA 7200 CPU chip.

Electrical Characterization

Electrical characterization refers to the task of creating different test environments and test code with the goal of identifying electrical failures on the chip. Once an electrical failure is detected, characterization also includes determining the characteristics of the failure like dependencies on voltage, temperature, and frequency.

Electrical failures may manifest themselves on one, several, or every chip at some operating point (temperature, voltage, or frequency) of the CPU. Electrical failures cause the chip to malfunction and typically have a root cause in some electrical phenomenon such as the familiar hold time or setup time violation. As chip operating frequencies increase, other electrical phenomena such as coupling between signals, charge sharing, and unforeseen interchip circuit interactions increasingly become issues.

To ensure a high level of quality, various types of testing and test environments are used to check that all electrical failures are detected and corrected before shipment to customers. Dwell testing and shmoo testing are two types of testing techniques used to characterize chips.

For the PA 7200, dwell testing involved running pseudorandom code on the system for extended periods of time at a given voltage-temperature-frequency point. Since the test code patterns are extremely important for electrical verification, dwell testing was used to guarantee that the pseudorandom code would generate sufficient patterns to test the CPU adequately.

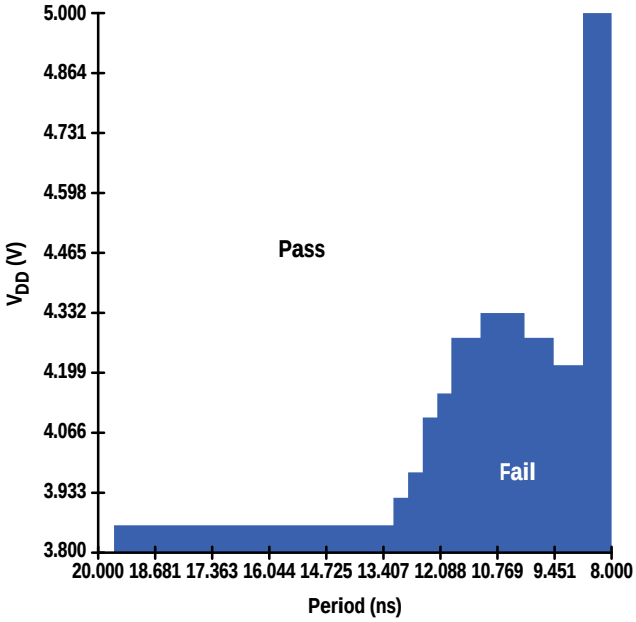
Shmoo testing involves creating voltage-frequency plots (shmoo plots) by running test code at many voltage-frequency-temperature combinations. Fig. 3 shows a typical style of shmoo plot. This plot is for a failing chip that has some speed problems. By examining the shape of the shmoo plot, much can be learned about the design of the chip. Voltage–frequency– temperature points well beyond the legal operating range should be included in the shmoo plot. It is not sufficient to rely only on the minimum allowed margin (in terms of voltage-frequency-temperature) to determine if the design is robust. The test code run for creating shmoo plots is extremely important. Simple code can create a false sense of quality.

Testing Environments

There were four main testing environments for the PA 7200: system characterization, chip tester characterization, production characterization, and functional characterization.

System Characterization. This testing is focused on running the CPU in the actual system and altering the operating variables to determine the characteristics of the design. The variables that are involved here are test code, ambient temperature, voltages (internal chip voltage, I/O pad voltage, and cache SRAM voltage), frequency of the chip, types of CPU chips (variations in manufacturing process), types of cache SRAMs (slow versus fast), and system bus speed. Various types of test code are run on the system, including pseudorandom PA-RISC code, HP-UX application code, and directed PA-RISC assembly code.

Fig. 3. *Voltage-frequency shmoo plot.*



Chip Tester Characterization. This testing consists of running a set of chips processed with different manufacturing process variables on a VLSI chip tester over ranges of temperature, voltage, and frequency, using a set of specific tests written for the PA 7200. The chip tester can run any piece of code at operating frequency by providing stimulus and performing checks at the I/O pins of the chip. Testing is accomplished through a mixture of parallel and scan methods using a VLSI test system. The majority of testing is done with at-speed parallel pin tests. Tests written in PA-RISC assembly code for the PA 7200 that cover logical functionality and speed paths are converted through a simulation extraction process into tester vectors. Scan-based tests are used for circuits such as standard-cell control blocks and PLA structures, which are inherently difficult to test fully using parallel pin tests. These parallel tests are run on the tester well beyond the operating speed of the chip.

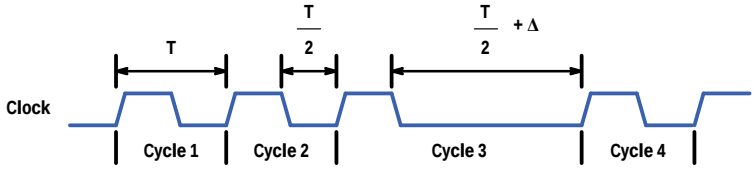
Production Characterization. All PA 7200 chips go through a set of tests on the chip tester. Since a large number of chips are manufactured for prototyping purposes, the results of the normal manufacturing tests are very valuable for characterization. This testing provides characterization data for all the chips that are manufactured with a set of specific tests written for the PA 7200 over ranges of temperature, voltage, and frequency. Parallel and scan tests written for the PA 7200 are run within the operating range possible on customer systems as well as in well-defined regions of margin outside this operating range. This type of testing over all the chips shows electrical failures that could happen if there are variations in the manufacturing process over time.

Functional Characterization. This testing involves running pseudorandomly generated tests on the system at the nominal operating point for very long periods of time (months). Even though this testing uses code environments targeted for functional verification, it can be very effective in detecting electrical issues. This type of testing can often find any test cases (circuit paths) that have not been covered in the prior three types of testing and will reduce the chance that the customer will ever have any electrical problems.

Debugging

When a problem is seen within the operating region of the chip, the problem must be debugged and fixed. Tests are run well beyond the operating region to look for anomalies. Failures outside the operating region are also understood to make sure that the failure will not move into the operating region (with a different environment, test, or manufacturing process shift). The root causes of these electrical problems need to be characterized and understood. In the characterization of the problem many chips are tried in various environments to understand the severity of the problem. To understand the cause of the failure, the test code is analyzed and converted to a small directed test with only the pertinent failing sequence. This is necessary to limit the scope of the investigation. Then the problem is further analyzed on the chip tester. The chip tester can run any piece of code at speed but it can run only reasonable sizes of code because of the amount of tester memory. The tester can perform types of experiments that the system cannot provide, such as varying the clock cycle for a certain period of time. This process is called *phase stretching* (see Fig. 4). Often the failing path can be determined at this point based on phase stretching experiments. Various other techniques can also be used on the tester to isolate the failing path. Once the failing path is isolated, the electrical failing mechanism needs to be understood. Various tools are used to determine the failing mechanism.

Fig. 4. *Timing diagram of a phase stretched clock. The normal period (T) of the clock is shown in cycles 1, 2, and 4. The normal phase time is T/2. In the second phase of cycle 3, the phase is stretched by time Δ for a total phase time of T/2 + Δ.*



One method to help identify the failing mechanism is to use an electron-beam (E-beam) scoping tool on the chip tester. In this process, the failing test is run in a loop on the tester and internal signals are probed to look at waveforms and the relationships between signals. It is very similar to using an oscilloscope to look at a signal on a printed circuit board except that it is done at the chip level.

As final confirmation of the failing mechanism, the failing circuit is modeled by the designer. The electrical components of the circuit path are extracted and simulated with a circuit simulator (SPICE). The modeling needs to be accurate to reproduce the failure on the simulator. Once the failing mechanism is confirmed in SPICE, a fix is developed and verified.

Since a chip turn to determine whether the fix will work and that the fix has no side effects takes a long time, the fix can often be verified with a focused-ion-beam (FIB) process. *FIBing* is a process by which the chip's internal connections can be modified, thereby changing its behavior or functionality. In the FIB process, metal wires can be cut, or joined by metal deposition. FIB is an extremely valuable tool to verify fixes before implementing them in the actual design.

After the electrical failing mechanism is understood, additional work is done to create the worst-case test for this failure. The insight gained from understanding the root cause allows the test to be tailored to excite the failing mechanism more readily. This can cause the test to fail more often, at a lower or higher frequency, a lower or higher voltage, or a lower or higher temperature. Developing a worst-case test is an extremely important step. The extent of the original problem cannot be understood until the worst-case test is developed. Including the worst-case test in the production screen ensures that parts shipped to customers will never exhibit the failure even under varying operating conditions and the most stressful hardware and software environments.

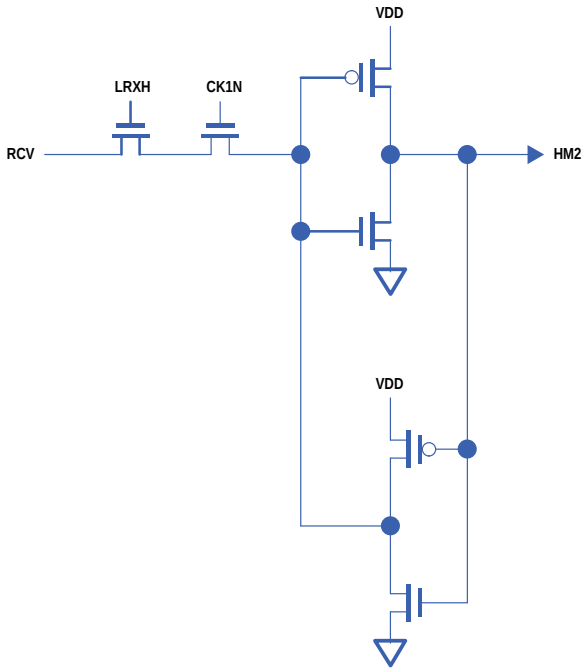
These points can be illustrated with a case study. The nominal operating point of the PA 7200 is 120 MHz at a V_{DD} of 4.4 volts. In this particular example, a failure occurred while running a pseudorandom test at 5.1 volts and 120 MHz at high temperature (55°C ambient). Even though the PA 7200 is not required to operate at this voltage the verification team did not expect this failure. Thus, this problem needed to be characterized and its root cause understood.

In this example, this chip was the only one that failed at 5.1 volts. However, a few other chips failed at even higher voltages. This problem was worse at higher frequencies and higher temperatures. The test code that was failing was converted from pseudorandom system code to tester code. Next the test code was run on the tester and analyzed. Since this problem did not occur at lower frequencies, each phase of the clock in the test was stretched to determine which clock phase made the chip pass or fail. The internal state of the chip was also dumped out on the tester using serial scan. The failing and passing internal scanned states were compared to see which states were improperly set. This helped to isolate the failing path. Once this was done, the failing path for this failure was analyzed to understand the electrical failing mechanism. For this problem, E-beam was used to understand the failing mechanism.

Fig. 5 shows the circuit that was failing in this debugging example. The circuit is a latch with the signal LRXH controlling the transfer of data into the latch. When LRXH and CK1N (clock) are true (logic 1), the latch is open and the inverted level of the input RCV gets transferred to the output HM2. When LRXH is false (logic 0), the latch is closed and the output HM2 holds its state.

Fig. 6 shows the waveforms of the internal signals that were captured through E-beam. The last two signals, CK1N and CK2N, are the two-phase clock signals on the chip. The passing and failing waveforms for LRXH and HM2 are shown at the top of the figure. The passing waveforms for LRXH and HM2 are called LRXH/4.7V@Ird+0ns and HM2/4.7@Ird+0ns, respectively. The failing waveforms for LRXH and HM2 are called LRXH/4.7V@Ird − 1.0ns and HM2/4.7@Ird − 1.0ns, respectively. The input signal RCV (not shown in the figure) is 1 during the first two pulses of LRXH shown, 0 during the third pulse, and 1 thereafter. The output HM2 is expected to transition from 0 to 1 during the third LRXH pulse and stay 1 until the fourth pulse. However, the slow falling edge on LRXH causes a problem. In the failing case, on the third LRXH pulse, HM2 transitions from 0 to 1 but the slow falling edge on LRXH also lets the next input value of RCV (1) propagate to the output HM2. HM2 therefore transitions back to 0. In the passing case, the falling edge of LRXH arrives a little earlier and the output HM2 maintains what was captured in the latch (1). Once the failing mechanism was understood, the worst-case test was developed.

Fig. 5. *A latch circuit that was failing at 5.1 volts.*



In this case study, the worst-case test caused many parts to fail at nominal conditions. The failing mechanism was modeled in a circuit model by the designer. Once this was done, a fix was developed. FIB was used to verify the fix. This failure mechanism was fixed by speeding up LRXH by adding a buffer to the long route of LRXH. Fig. 7 shows how this was done. The figure is a photograph of a die that was FIBed to buffer the LRXH signal. To do this, the long vertical metal 3 wire on the right side of the figure was cut with the FIB process and a buffer was inserted. A buffer was available on the left side of the figure; however, metal 3 covered this buffer. The FIB process was used to etch the metal 3 area surrounding the buffer to expose the metal 2 connections of the buffer. The FIB process was then used to deposit metal to connect the metal 2 of the buffer to the vertical metal 3 wires. The FIBed chip was then tested to make sure that the failing mechanism was fixed.

Testability

We leveraged the test circuits and strategy for the PA 7200 from the PA 7100 chip. The scan controller was required to change from our proprietary diagnostic instruction port to the industry-standard JTAG. This was a minimal change, since both protocols do the same function. The new test controller was leveraged from the PA 7100LC chip to keep the design effort to a minimum. Before tape release we verified that the basic test circuits would work.

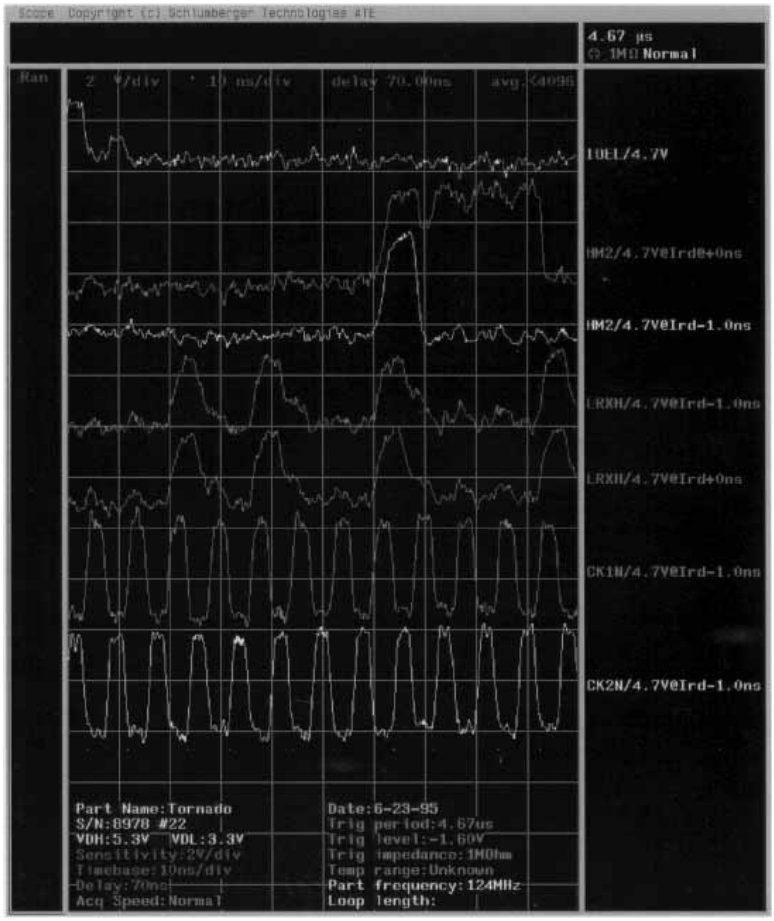
Since the test circuits were leveraged from the PA 7100, the obvious choice was to leverage the test strategy from the PA 7100 chip as well. A fast parallel pin tester was chosen early on as the tester for the PA 7200. This tester would provide both parallel pin testing and scan testing. We decided that data path circuits would be tested by parallel pin testing, and scan testing would be limited to a few control blocks. All speed testing was to be done with parallel pin testing.

Benchtop Tester

Since the parallel pin tester was located elsewhere, we knew we could not use it for local debugging of the chip. Many problems needed only simple debugging capability and could be greatly accelerated by the presence of a local debugging station. For that purpose, we chose an inexpensive benchtop tester developed internally. This tester applied all vectors serially to the chip. Vectors developed for serial use could be used as is. The parallel pin vectors could be translated into what we called *pin vectors*, which is a boundary scan, looking-into-the-chip approach. No speed testing capability was planned, although some support for speed testing was present in the PA 7200.

The PA 7200 chip has on-chip clock control. This was essential to our success because the benchtop tester was not practically able to provide a separate clock control signal. The tester can (and did) issue clock control commands in the serial data. Having these commands interpreted on-chip saved us from having to build that circuitry off-chip. This made the chip test fixture very simple.

Fig. 6. *Waveforms of the internal signals of the failing latch circuit, captured by electron-beam probing.*



The benchtop tester was the only means of standalone chip testing we had collocated with the design team, and therefore was very important to the debugging efforts. The tester used a workstation as a controller and interface, and was capable of storing very long vectors (limited only by the workstation's virtual memory). We had the ability to load the entire parallel pin vector suite (590 million shifts) into the benchtop tester at one time, although this took so long as to be practically prohibitive. The benchtop tester had both scan and some limited parallel pin capabilities for driving reset pins.

Benchtop Tester Environment

The benchtop tester was based on an HP-UX workstation and could be operated from a script. This allowed us to put our own script wrappers around the software, which provided essential control for power supplies and the pulse generator. These script wrappers also provided transparent workarounds to some of the limitations of the tester.

We had two testers that we controlled access to via HP TaskBroker. By using HP TaskBroker, we could easily share the test fixtures between the various uses, such as test development, chip debugging, and automatic test verification. For chip debugging, an engineer could obtain an interactive lock on the tester (a window would pop up when an engineer got the tester), and did not have to worry about interference from an unattended job trying to run. Also, a test could be initiated from an engineer's desk, and when a tester was free, the test would run and return the results to the engineer. HP TaskBroker handled all the queuing and priority issues.

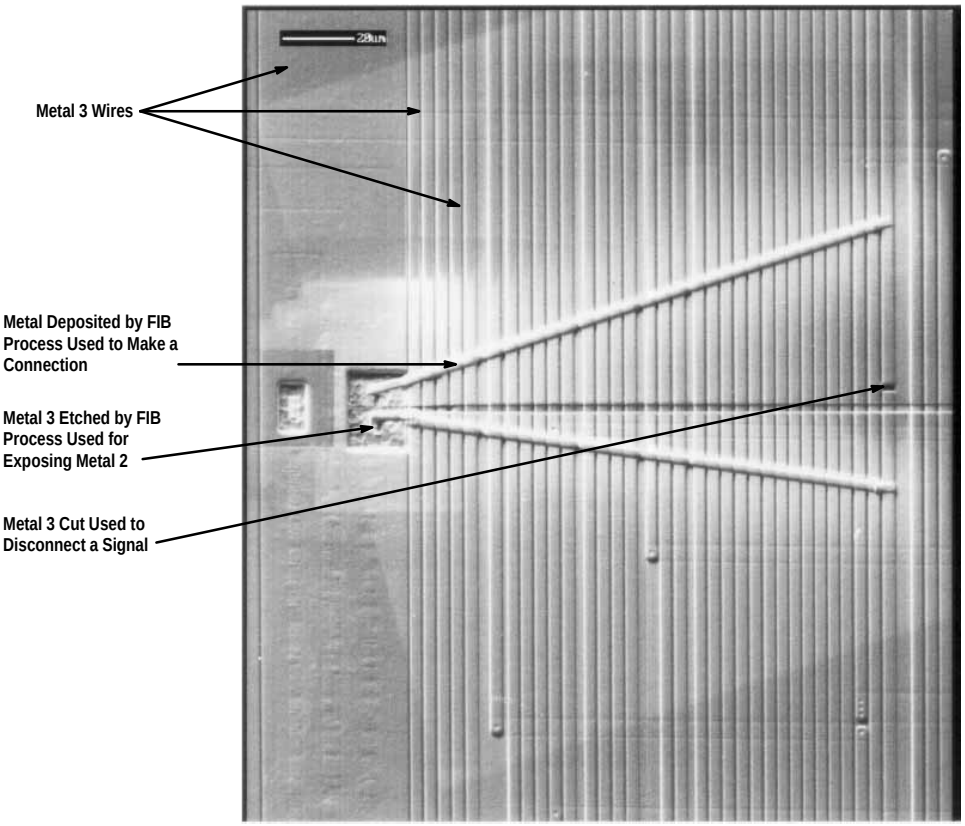
As our experience increased and our needs became clear, we wrote more simple scripts around those we already had. This allowed us to write complex functions as composites of simple blocks.

Double Step

As chip bring-up progressed, we found that we could benefit from some simple local speed test capabilities. As a result, we chose to implement basic speed testing on the benchtop tester stations we had in place.

We employed programmable pulse generators and had the software to control the frequency. All that was needed was to convert the tests to double-step pin vectors and make sure they worked. A double-step pin vector is the same as a single-step pin vector, except that two chip cycles are run at speed. This requires that the I/O cells be able to store two values, not just one as would be required for single stepping. This feature was already in the I/O cell design.

Fig. 7. *Once the latch problem was found and a fix developed, the fix was verified by modifying one die using a focused-ion-beam (FIB) process. The long vertical metal 3 wire on the right was cut with the FIB process and a buffer was inserted. A buffer was available on the left side of the figure; however, metal 3 covered this buffer. The FIB process was used to etch the metal 3 area surrounding the buffer to expose the metal 2 connections of the buffer. The FIB process was then used to deposit metal to connect the metal 2 of the buffer to the vertical metal 3 wires. The FIBed chip was then tested to make sure that the failing mechanism was fixed. Photo courtesy of FAST (FIB Applied Semiconductor Technology), San Jose, California.*



By converting the tests to double-step pin vectors and making some minor changes to the design, we got double-step pin vectors working. This capability to do at-speed local testing was very valuable in debugging the chip.

Additional Tools

A simple tool was put together to produce shmoo plots of about 50 points for a single test. We spent considerable effort optimizing this script. The engineers doing debugging found this very valuable.

When doing speed path debugging, the engineer wants to know which cycles are slow. One way to find out is to take a failing test and make some cycles slower, and if the chip passes, that means that the chip was failing on one of those cycles. Just observing the pins is not enough, however, since a failure may stay inside the chip for a while before propagating to the pins. We implemented this kind of test by changing our pin vector strategy from a double step to a combination of half steps and single steps during selected cycles of the test. Since the clock commands take a long time to shift in, this effectively slows down some cycles of the test. We call this style of testing phase stretching (see Fig. 4).

Another very valuable tool was an automated phase stretching tool. It would take a chip and find the slow cycles with a given set of tests. This would take a few hours, but need not be supervised, so overnight tests worked well. While this would not tell what the problem was directly, it provided valuable clues.

We also had the ability to run part of a test, then stop it and dump the state of the internal scan chains. A chip expert could look at these dumps and see what went wrong. Use of this tool was extremely useful during our debugging efforts.

The benchtop testers were considered very valuable to the debugging of the PA 7200. The software written for the testers contributed greatly to their success. The benchtop testers became known for their reliability, ease of use, and locality.

Design-Process Interactions

To achieve the highest quality in any VLSI product, it is very important to ensure that there is good harmony in the relationship between the chip design and the chip fabrication process. This relationship on the PA 7200 went through some rocky roads and had its own interesting set of problems. In the end, however, the desired harmony was achieved and is reflected in the high quality and yields of the final product. This section will describe the situation that existed on the PA 7200 and some of the steps taken to anticipate and smooth out problems in this area.

The characteristics of the IC process have a big influence on decisions made at every step of the development cycle of a VLSI product, starting from the early stages of the design. The influence can be seen in many areas like the goals of the design, the feature set to be included, and the details of the implementation at the transistor level. For example, the process dictates the intrinsic speed of the transistor, which is a key factor in setting the frequency goals of the chip. Similarly, the minimum feature sizes (line width, spacing, etc.) of the process largely dictate the size of the basic storage or memory cell. This in turn is a factor that determines, for example, the size of the TLB on the die, which is a key component in determining the performance of a microprocessor. An example of this influence at the implementation level would be an input pad receiver designed to trip at a particular voltage level on the external (input) signal. The implementation has to ensure that the trip level is fairly tightly controlled at all corners of the process, which is not easy to do. Another trivial example is the size of a power or ground trace. The size of the trace required to carry a certain amount of current is largely dictated by the resistance and electromigration limits of the metal.

There were two target HP IC processes in mind when the design of the PA 7200 began: CMOS26 and CMOS14. CMOS26 was the process of the previous generation CPUs, the PA 7100 and the PA 7100LC. Its benefits were that it was a very mature and stable process. Also, some circuits of the PA 7100 are used in the PA 7200 with little or no modification, and the behavior of these circuits was well-understood in this process. CMOS14 was the next-generation process being developed. Its benefits were, obviously, smaller feature size and better FET speed. However, only a few simple chips had been fabricated in this process before the PA 7200, and many startup problems were likely to be encountered. That we had a choice influenced the design methodology. Taking advantage of the scalability of CMOS designs, the initial design was done in CMOS26. An artwork shrink process was developed to convert the design to CMOS14. The shrink process is a topic that merits special attention and is described in *Article 3*.

As the design went along, it became clear that to meet the performance and size goals of the product, CMOS14 was the better choice. To demonstrate feasibility and to iron out problems with the shrink process, the existing PA 7100 CPU was taken through the shrink process and fabricated in CMOS14. Several issues were uncovered, leading to early detection of potential problems.

Related to the IC fabrication process, the goal of electrical verification and characterization is to ensure that the VLSI chip operates correctly for parts fabricated within the bounds of the normal process variations expected during manufacturing. An incomplete job done here or variations of the process outside the normal range can cause subtle problems that often get detected much later on. There are two yield calculations that are often used to quantify the manufacturability of a VLSI product. The *functional yield* denotes the fraction of the total die manufactured that are fully operational (or functional) at some electrical operating point, that is, some combination of frequency, voltage, and temperature. The *survival yield* denotes the fraction of the functional die that are operational over the entire electrical operating range of the product, that is, the product specifications for frequency, voltage, and temperature. (In reality, to guarantee this, there is some guardbanding that occurs beyond the operating range of the product)

To achieve the highest quality and manufacturability of the final product, the following are some of the objectives set for electrical characterization:

- Ensure that the design has solid operating margin (in voltage, frequency, and temperature) for parts fabricated at all the different corners of the process.
- Ensure consistently high survival yield for a statistically large number of wafers and lots fabricated.
- To ferret out problems that may be otherwise hard to find, fabricate some parts at points beyond the normal variations of the process. Debug problems in these parts to ensure the robustness of the design.

The PA 7200 chip was the first complex VLSI chip to be fabricated in CMOS14. That the process was not fully mature at that point had important implications on the electrical characterization and debugging effort. Special care had to be taken to distinguish between the different types of problems: design problems, process problems, and design-process interaction problems.

A fundamental design problem is one that shows up on every lot (a batch of wafers processed together in the fabrication shop), whatever the process parameters for that lot might be. For example, a really slow path on the chip may have some frequency variation from lot to lot, but will show up on every lot.

Process problems show up on some lots. The most common symptom is poor functional yield. Sometimes, however, the symptom can be a subtle electrical failure that is hard to debug. For example, one problem manifested itself as a race between the falling edge of a latching signal and new data to the latch. SPICE simulations showed that the failure could occur only under abnormally unbalanced loads and clock skews, which were unrealistic.

A design-process interaction problem shows up to varying degrees on different lots. It points to a design that is not very robust and is treated very much like a design problem. However, typically there tends to be some set of process conditions that aggravate the problem. Tighter process controls or retargetting the process slightly can reduce the impact of such problems temporarily, but the long-term solution is always to fix the design to make it more robust. For example, some coupling issues on the PA 7200 occurred only at one corner of the process. By retargetting the process to eliminate that corner, the survival yield was significantly increased.

The shrink process mentioned earlier had given us tremendous benefits in terms of flexibility and the ability to leverage existing circuits. However, effort was also spent in identifying circuits that did not shrink very well. These circuits were given special care and modified when the decision to use CMOS14 was made. Overall, the shrink effort was very successful largely because of the scalability of CMOS designs. However, the characterization and debugging phase exposed some interesting new limitations on the scalability of CMOS designs. When a shrink-related problem circuit was found, the chip was scanned for other circuits that could have a similar problem. These circuits were then fixed to prevent future problems.

Throughout the project, the team always tracked down problems to their root causes. This approach guaranteed complete fixes for problems and kept them from ever showing up again. The result is high-quality bug-free parts and high yields in manufacturing.

In addition to finding and fixing problems in the design, there was also a related activity that happened in parallel. Process parametric data was analyzed in detail for every lot to look for an optimum region in the process. Detailed correlation data was produced between process parameters and chip characteristics like speed, failing voltages, types of failures, and so on. Many different experiments with process parameters and masks were also tried, including polysilicon biases, metal thicknesses, and others. This enabled us to fine-tune the process to increase the margins, yields, and quality of the product.

Conclusion

With the increasing complexity of VLSI chips, specifically CPUs, design verification has become a critical and challenging task. This paper has described the methodology and techniques used to verify the PA 7200 CPU. The approaches used yielded very good results and led to the efficient detection and isolation of problems on the chip. This has enabled Hewlett-Packard to achieve high-quality, volume shipments in a timely manner.

Acknowledgments

The authors would like to thank all the different teams who contributed to the successful verification and characterization of the PA 7200 chip. Special thanks to the many individuals from the Computer Technology Laboratory and the Cupertino Systems Laboratory in Cupertino, California who were involved. Many thanks also to our key partners: the Engineering Systems Laboratory in Fort Collins, Colorado, the Integrated Circuits Business Division in Fort Collins and Corvallis, Oregon, the General Systems Laboratory in Roseville, California, the Chelmsford Systems Laboratory in Chelmsford, Massachusetts, the Colorado Computer Manufacturing Operation in Fort Collins, and the Cupertino Open Systems Laboratory in Cupertino.

▶ Go to *Article 5*
▶ Go to Table of Contents
▶ Go to HP Journal Home Page