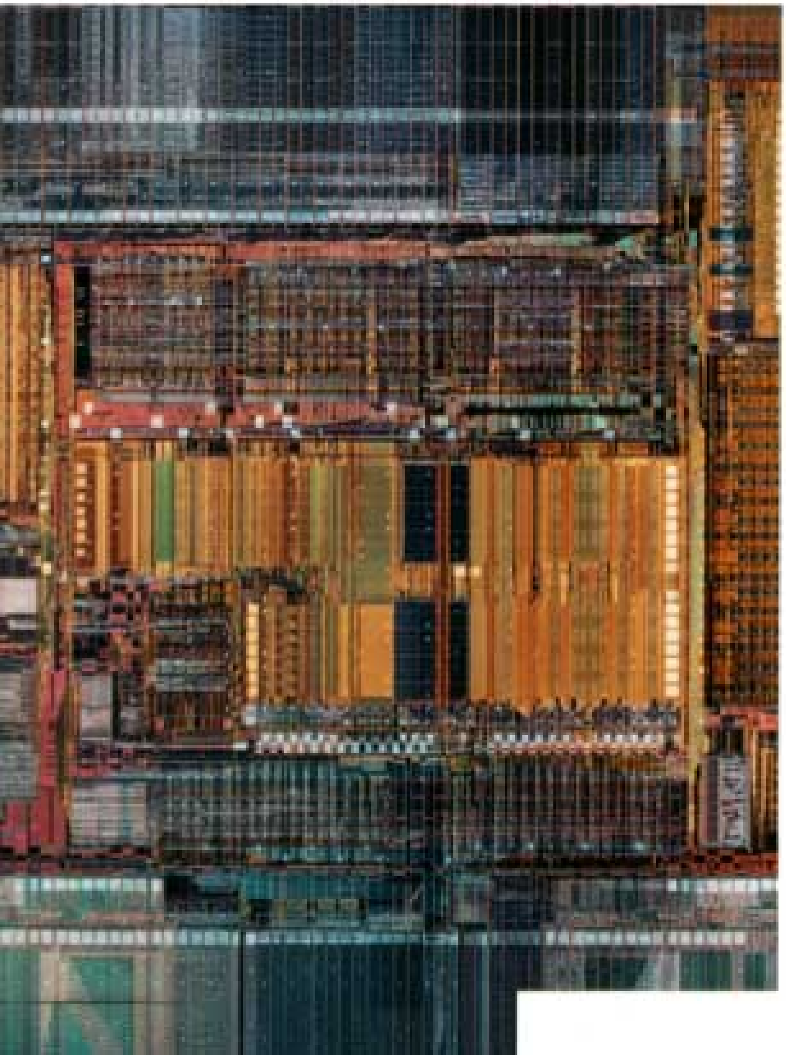
HEWLETT-PACKARD

JOURNAL

August 1997



HEWLETT PACKARD

table of contents

August 1997,
Volume 48, Issue 4

Articles

Four-Way Superscalar PA-RISC Processors

The HP PA 8000 and PA 8200 PA-RISC CPUs feature an aggressive four-way superscalar implementation, speculative execution, and on-the-fly instruction reordering.

by Anne P. Scott, Kevin P. Burkhart, Ashok Kumar, Richard M. Blumberg, and Gregory L. Ranson

The HP PA 8000 and PA 8200 PA-RISC CPUs are the first implementations of a new generation of microprocessors from Hewlett-Packard. The PA 8000[1-3] is among the world's most powerful and advanced microprocessors, and at the time of introduction in January 1996, the undisputed performance leader. The PA 8200,[4] introduced in June 1997, continues this performance leadership with higher frequency, larger caches, and several other enhancements. Both processors feature an aggressive four-way superscalar implementation, combining speculative execution with on-the-fly instruction reordering. This paper discusses the objectives for the design of these processors, some of the key architectural features, implementation details, and system performance. The operation of the *instruction reorder buffer* (IRB),[5] which provides out-of-order execution capability, will also be described.

PA 8000 Design Objectives

The primary design objective for the PA 8000 was to obtain industry-leading performance on a broad range of real-world applications. To sustain high performance on large applications, not just on benchmarks, we designed large, external primary caches with the ability to hide memory latency in hardware. We also chose to implement dynamic instruction reordering in hardware to maximize the instruction-level parallelism available to the execution units. Another goal was to provide full support for 64-bit applications. The processor implements the new PA-RISC 2.0 architecture, which is a binary compatible extension of the previous PA-RISC architecture. All previous code will execute without recompilation or translation. The processor also provides glueless support for up to four-way multiprocessing via a high-bandwidth Runway system bus.[6] The Runway bus is a 768-Mbyte/s split-transaction bus that allows each processor to have several outstanding memory requests.

PA-RISC 2.0 Enhancements

The new PA-RISC 2.0 architecture incorporates a number of advanced microarchitectural enhancements. Most of the extensions involve support for 64-bit computing. Integer registers and functional units, including the shift/merge units, have been widened to 64 bits. Flat virtual addressing up to 64 bits is supported, as are physical addresses greater than 32 bits (40 bits were implemented on the PA 8000). A new mode bit has been implemented that governs address formation, creating increased flexibility. In 32-bit addressing mode, it is still possible to take advantage of 64-bit compute instructions for faster throughput. In 64-bit addressing mode, 32-bit instructions and conditions are still available for backwards compatibility.

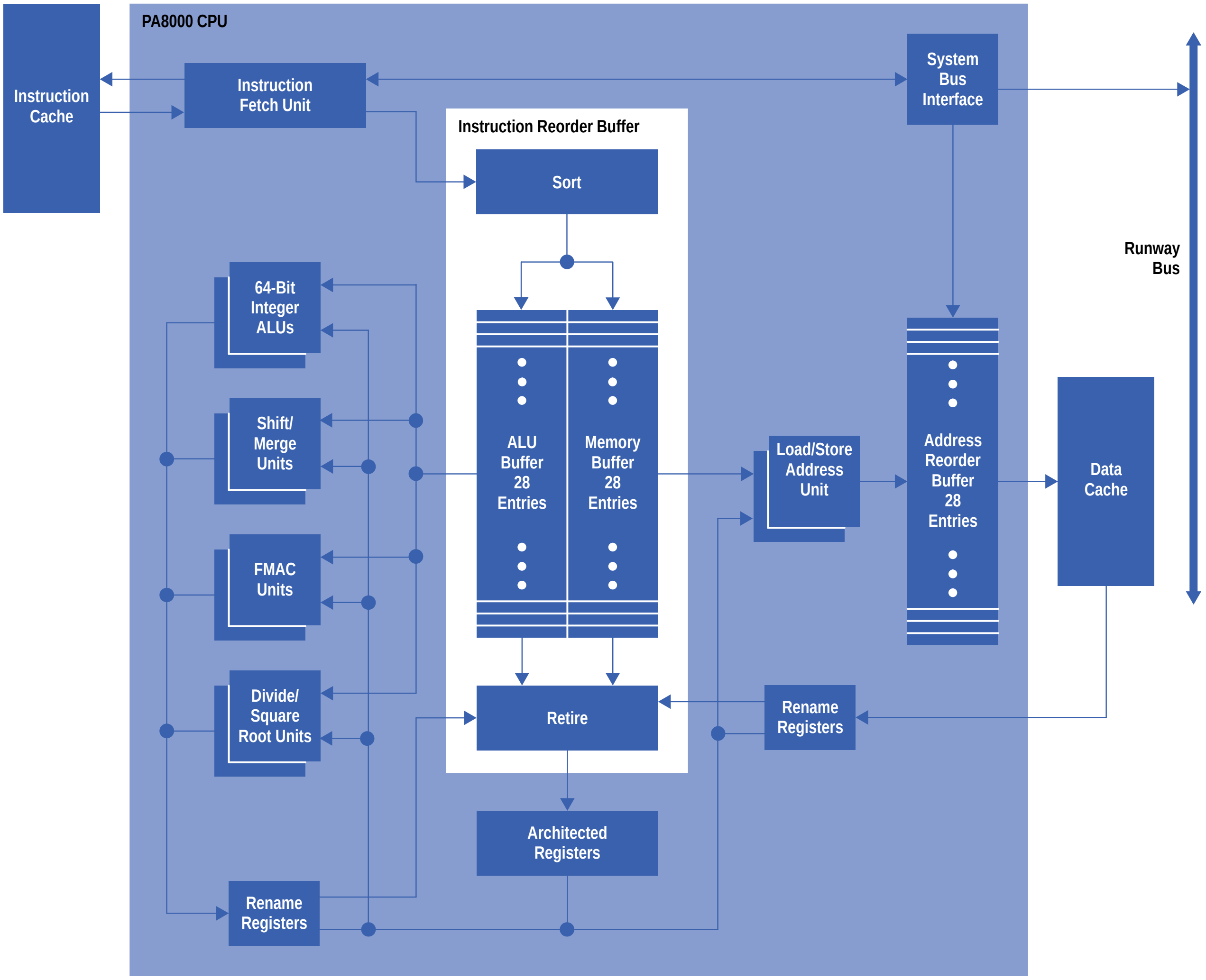
Other extensions help optimize performance in the areas of virtual memory and cache management, branching, and floating-point operations. These include fast TLB (translation lookaside buffer) insert instructions, load and store instructions with 16-bit displacement, memory prefetch instructions, support for variable-sized pages, half-word instructions for multimedia support, branches with 22-bit displacements and short pointers, branch prediction hinting, floating-point multiply-accumulate instructions, floating-point multiple compare result bits, and other carefully selected features.

Hardware Design

The PA 8000 features a completely redesigned core that does not leverage any circuitry from previous-generation HP processors. This break from previous CPUs allowed us to include new microarchitectural features we deemed necessary for higher performance. Fig. 1 is a functional block diagram of the processor showing the basic control and data paths.

The most notable feature of the chip, illustrated in the center of the diagram, is the industry's largest instruction reorder buffer of 56 entries, which serves as the central control unit. This block supports full register renaming for all instructions in the buffer, and tracks interdependencies between instructions to allow data flow execution through the entire window.

The PA 8000 features a peak execution rate of four instructions per cycle, made possible by a large complement of computational units, located on the left side of the diagram. For integer operation, two 64-bit integer ALUs and two 64-bit shift/merge units are included. All integer functional units have a single-cycle latency. For floating-point applications, dual floating-point multiply and accumulate (FMAC) units and dual divide/square root units are included. The FMAC units are optimized for performing the very common operation A times B plus C. By fusing an add to a multiply, each FMAC can execute two floating-point operations in just three cycles. In addition to providing low latency for floating-point operations,



FMAC = Floating-Point Multiply and Accumulate
ALU = Arithmetic Logic Unit

Fig. 1. *Functional block diagram of the HP PA 8000 processor.*

the FMAC units are fully pipelined so that the peak floating-point throughput of the PA 8000 is four floating-point operations per cycle. The two divide/square root units are not pipelined, but other floating-point operations can be executed on the FMAC units while the divide/square root units are busy. A single-precision divide or square root operation requires 17 cycles, while double precision requires 31 cycles.

Having such a large array of computation units would be pointless if those units could not be supplied with enough data upon which to operate. To this end, the PA 8000 incorporates two complete load/store pipes, including two address adders, a 96-entry dual-ported TLB, and a dual-ported cache. The right side of Fig. 1 shows the dual load/store units and the memory system interface. The symmetry of dual functional units throughout the processor allows a number of simplifications in the data paths, the control logic, and signal routing. In effect, this duality provides for separate *even* and *odd* machines.

As pipelines get deeper and the parallelism of a processor increases, instruction fetch bandwidth and branch prediction become increasingly important. To increase fetch bandwidth and mitigate the effect of pipeline stalls for branches predicted to be taken, the PA 8000 incorporates a 32-entry *branch target address cache*, or BTAC. This unit is a fully associative structure that associates the address of a branch instruction with the address of its target. Whenever a branch predicted to be taken is encountered in the instruction stream, an entry is created in the BTAC for that branch. The next time the fetch unit fetches from the address of the branch, the BTAC signals a hit and supplies the address of the branch target. The fetch unit can then immediately fetch the target of the branch without incurring any penalty, resulting in a zero-state taken branch penalty for branches that hit in the BTAC. In an effort to improve the hit rate, only branches predicted to be taken are kept in the BTAC. If a branch hits in the BTAC but is predicted not to be taken, the entry is deleted.

To reduce the number of mispredicted branches, the PA 8000 implements two modes of branch prediction: *dynamic mode* and *static mode*. Each TLB entry has a bit to indicate which prediction mode to use. Thus, the mode is selectable on a page-by-page basis. In dynamic prediction mode, a 256-entry *branch history table*, or BHT, is consulted. The BHT stores the results of the last three iterations of each branch (either taken or not taken), and the instruction fetch unit predicts that the outcome of a given branch will be the same as the majority of the last three outcomes. In static prediction mode, the PA 8000 predicts most conditional forward branches to be untaken, and most conditional backward branches to be taken. For the common compare-and-branch instruction, the PA-RISC 2.0 architecture defines a branch prediction bit that indicates whether this normal prediction convention should be followed or whether the opposite convention should be used. Compilers using either heuristic methods or profile-based optimization can use static prediction mode to communicate branch probabilities effectively to the hardware.

Cache Design

The PA 8000 features large, single-level, off-chip, direct-mapped instruction and data caches. Both caches support configurations of up to four megabytes using industry-standard synchronous SRAMs. Two complete copies of the data cache tags are provided so that two independent accesses can be accommodated and need not be to the same cache line.

Why did we design the processor without on-chip caches? The main reason is performance. Competing designs incorporate small on-chip caches to enable higher clock frequencies. Small on-chip caches support benchmark performance but fade on large applications, so we felt we could make better use of the die area. The sophisticated IRB allows us to hide the effects of a pipelined two-state cache latency. In fact, our simulations demonstrated only a 5% performance improvement if the cache were on-chip and had a single-cycle latency. The flat cache hierarchy also eliminates the design complexity associated with a two-level cache design.

Chip Statistics

The PA 8000 is fabricated in HP's 0.5-micrometer, 3.3-volt CMOS process. Although the drawn geometries are not very aggressive, we still obtain a respectable 0.28-μm effective channel length (L_{eff}). In addition, extensive investment was made in the design process to ensure that both layout and circuits would scale easily into more advanced technologies with smaller geometries. There are five metal layers: two for tight pitch routing and local interconnect, two for low-RC global routing, and a final layer for clock and power supply routing.

The processor is designed with a three-level clock network, organized as a modified H-tree (see ***Article 2***). The clock sync signals serve as primary inputs. They are received by a central buffer and driven to twelve secondary clock buffers located in strategic spots around the chip. These buffers then drive the clock to the major circuit areas, where it is received by *clock gaters* featuring high gain and a very short input-to-output delay. There are approximately 7,000 of these gaters, which have the ability to generate many flavors of the clock: two-phase overlapping or nonoverlapping, inverting or noninverting, qualified or nonqualified. The qualification of clocks is useful for synchronous register sets and dumps, as well as for powering down sections of logic when not in use. Extensive simulation and tuning of the clock network were done to minimize clock skew and improve edge rates. The final clock skew for this design was simulated to be no greater than 170 ps between any two points on the die.

Under nominal operating conditions of room temperature and 3.3-volt power supplies, the chip is capable of running at frequencies up to 250 MHz. Although we cannot guarantee processor performance based on results obtained under ideal conditions, there appears to be an opportunity for greater frequency enhancement. The die measures 17.68 mm by 19.1 mm and contains 3.8 million transistors. Approximately 75% of the chip is either full-custom or semicustom. A photograph of the die with all major areas labeled is shown in Fig. 2. Again, the IRB is in the center of the chip, providing convenient access to all the functional units. The integer data path is on the left side of the chip, while the right side contains the floating-point data path.

By using flip-chip packaging technology, we were able to support a very large number of I/O signals—704 in all. In addition to the I/O signals, 1,200 power and ground solder bumps are connected to the 1,085-pin package via a land grid array. There are fewer pins than the total of the I/Os and bumps because each power and ground pin can be connected to multiple bumps. A picture of the packaged part is shown in Fig. 3. The chip is flipped onto the ceramic carrier using solder bump interconnect, and the carrier is mounted on a conventional printed circuit board. This packaging has several advantages. The wide off-chip caches are made possible by the high-pin-count capability. The ability to place I/O signals anywhere on the die improves area utilization and reduces on-chip RC delays. Finally, the low inductance of the signal and power supply paths reduces noise and propagation delays.

Performance

At 180 MHz with one megabyte of instruction cache and one megabyte of data cache, the HP PA 8000 delivers over 11.8 SpecInt95 and greater than 20.2 SpecFP95, making it the world's fastest processor at the time of introduction. A four-way multiprocessor system has also produced 14,739.03 TpmC ($132.25/TpmC), where TpmC is an industry-standard benchmark for online transaction processing. That system configuration was made available in June 1996.

Enabling the PA 8000 to achieve this level of performance are several distinguishing features. First, there are a large number of functional units—ten, as described previously. However, multiple units alone are not enough. To sustain superscalar operation beyond two-way demands advanced instruction scheduling methods to supply a steady stream of independent tasks to the functional units. To achieve this goal, an aggressive out-of-order execution capability was incorporated. The instruction reorder buffer provides a large window of available instructions combined with a robust dependency tracking system.

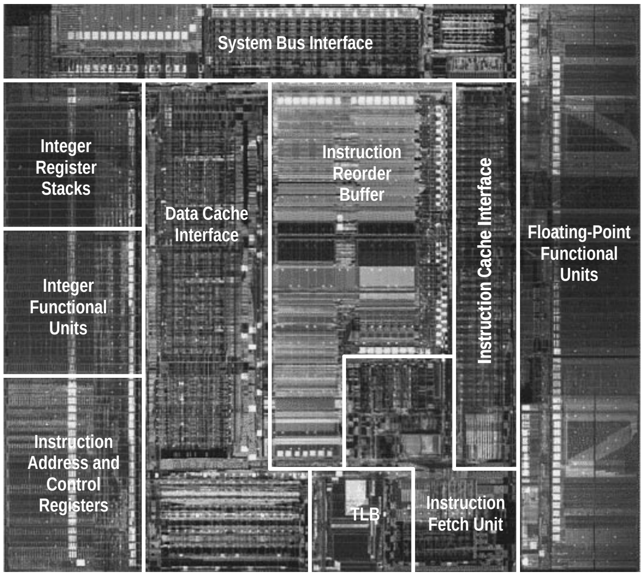

System Bus Interface
Integer Register Stacks
Data Cache Interface
Instruction Reorder Buffer
Instruction Cache Interface
Floating-Point Functional Units
Integer Functional Units
Instruction Address and Control Registers
TLB
Instruction Fetch Unit

Fig. 2. *PA 8000 CPU with major areas labeled.*

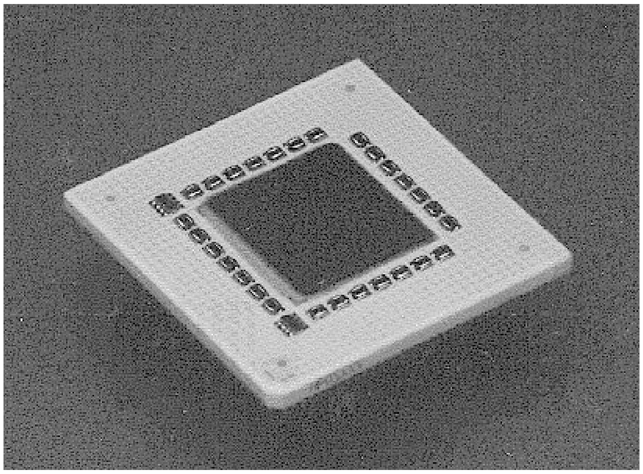


Fig. 3. *Packaged PA 8000 CPU.*

Second, having explicit compiler options to generate hints to the processor helps a great deal. These special instructions can be used to prefetch data and to communicate statically predicted branch behavior to the branch history table, as described previously.

Finally, the system bus interface is capable of tracking up to ten pending data cache misses, an instruction cache miss, and an instruction cache prefetch. Since multiple misses can be serviced in parallel, the average performance penalty caused by each is reduced.

Instruction Reorder Buffer

Because of restrictions on compiler scheduling, a key decision was made to have the PA 8000 perform its own instruction scheduling. To accomplish this task, the PA 8000 is equipped with an instruction reorder buffer, or IRB, which can hold up to 56 instructions. This buffer is composed of two pieces: the ALU buffer, which can store up to 28 computation instructions, and the MEM (memory) buffer, which can hold up to 28 load and store instructions. These buffers track over a dozen different types of interdependencies between the instructions they contain, and allow instructions anywhere in the window to execute as soon as they are ready.

As a special feature, the IRB tracks branch prediction outcomes, and when a misprediction is identified, all instructions that were incorrectly fetched are flash-invalidated. Fetching then resumes down the correct path without any further wasted cycles.

The IRB serves as the central control logic for the entire chip, yet consists of only 850,000 transistors and consumes less than 20% of the die area. A high-performance IRB is of paramount importance, since today's compilers simply lack run-time information, which is useful for optimal scheduling. The reorder buffer on the PA 8000 is 40% larger than that of the nearest competitor.

Instruction reordering also leads to the solution for another bottleneck: memory latency. Although the dual load/store pipes keep the computation units busy as long as the data is cache-resident, a data cache miss can still cause a disruption. Execution can continue for many cycles on instructions that do not depend on the data cache miss. The PA 8000 can execute instructions well past the load or store that was missed, since the IRB can hold so many instructions. When useful work can be accomplished during a data cache miss latency, the net impact on performance is significantly reduced.

The large window of available instructions also allows overlap of multiple data cache misses. If a second data cache miss is detected while an earlier miss is still being serviced by main memory, the second miss will be issued to the system bus as well.

Life of an Instruction

A block diagram of the PA 8000's instruction reorder buffer is shown in Fig. 4. Instructions enter through the sort block and are routed to the appropriate portion of the IRB based on instruction type, where they are held until they retire. The functional units are connected to the appropriate section of the IRB based on what types of instructions they execute. After execution, instructions are removed from the system through the retire block.

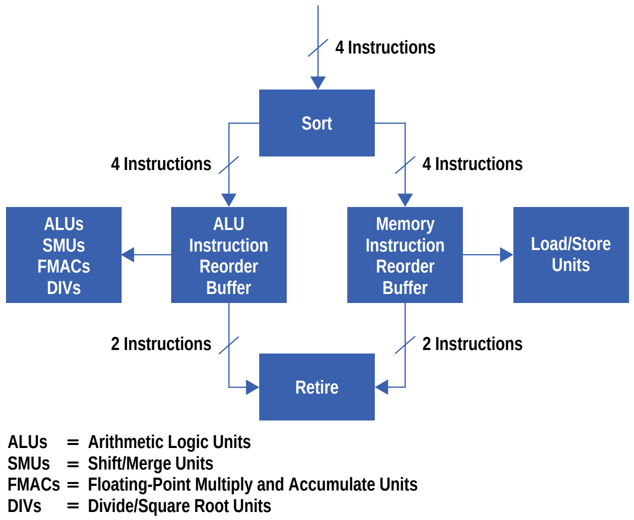

4 Instructions
Sort
4 Instructions
4 Instructions
ALUs
SMUs
FMACs
DIVs
ALU Instruction Reorder Buffer
Memory Instruction Reorder Buffer
Load/Store Units
2 Instructions
2 Instructions
Retire
ALUs = Arithmetic Logic Units
SMUs = Shift/Merge Units
FMACs = Floating-Point Multiply and Accumulate Units
DIVs = Divide/Square Root Units

Fig. 4. *PA 8000 instruction reorder buffer block diagram.*

Instruction Insertion. The IRB must be kept as full as possible to maximize the chances that four instructions are ready to execute on a given cycle. A high-performance fetch unit was designed to maximize IRB occupancy. This unit fetches, in program order, up to four instructions per cycle from the single-level off-chip instruction cache.

Limited predecode is then performed, and the instructions are inserted in a round-robin fashion into the appropriate IRB. Each IRB segment must be able to handle four incoming instructions per cycle, since there are no restrictions on the mix of instructions being inserted.

There are several special cases. Branches, although executed from the ALU IRB, are also stored in the MEM IRB as a placeholder to indicate which entries to invalidate after a mispredicted branch. Instructions that have both a computation and a memory component and two targets, such as the load word and modify (LDWM) instruction, are split into two pieces and occupy an entry in both portions of the IRB.

Instruction Launch. Instructions are allowed to execute out of order. During every cycle, both segments of the IRB allow the oldest even and the oldest odd instruction for which all operands are available to execute on the functional units. Thus, up to four instructions can be executed at once: two computation instructions and two memory reference instructions. Once an instruction has been executed, its result is held in a temporary rename register and made available for use by subsequent instructions.

Instruction Retire. Instructions are removed or retired from the IRB in program order once they have executed and any exceptions have been detected. Enforcing strict retirement order provides software with a precise exception model. As instructions are retired, the contents of the rename registers are transferred to the general registers, stores are placed in a queue to be written to cache, and instruction results are committed to the architected state. The retire unit can handle up to two ALU or floating-point instructions and up to two memory instructions each cycle.

The HP PA 8200 Processor

After the successful introduction of PA 8000 processor-based products, the PA 8000 design team initiated a follow-up program. Performance analysis on key applications identified several opportunities for future products. The PA 8200 CPU team formulated a plan for improvement based on the following goals set by HP customers and management:

- Improved performance
- Compatibility with existing applications
- Leverage of the PA 8000 design foundation
- Rapid time to market.

Improved Performance. Application trace studies identified branch prediction, TLB miss rates, and increased cache sizes as significant opportunities. The availability of next-generation 4M-bit SRAMs with improved access times allowed the design team to increase processor clock speed and double cache size to 2M bytes for both the instruction cache and the data cache. The faster access time of 4M-bit SRAMs allowed higher processor clock rates without changes to the cache access protocol. The combination of increased clock frequency, larger caches, improvement of branch prediction accuracy, and reduction of TLB miss rates enables performance improvements of 15% to 30% on key applications.

Compatibility with Existing Applications. Follow-on products using the PA 8200 had to preserve our customers' investment in PA 7200-based and PA 8000-based software and hardware. It was considered essential to maintain binary compatibility with existing PA-RISC applications and provide an upgrade path for improved performance.

Leverage of PA 8000. The PA 8200 design team leveraged the extensive functional and electrical verification results accumulated during the prototyping phase of the PA 8000 development. A wealth of design data is collected in the process of turning a design into a product. This information identified the paths limiting CPU operating speed and the performance limiters in the branch and TLB units. Characterization of the PA 8000 cache design provided the basis for a new design using high-speed 4M-bit SRAMs.

Rapid Time to Market. The competitive situation dictated that speed upgrades to the PA 8000 were needed to maintain HP's performance leadership in the high-performance workstation and midrange server markets. Therefore, design changes and characterization of the expanded cache subsystem had to be completed within a very aggressive schedule.

In the following sections, PA 8200 design changes to the PA 8000 processor will be detailed.

PA 8000 Performance Analysis

Given the goals of increased performance with low risk and a short time to market, it was necessary to understand fully where the PA 8000 excelled and where significant improvements could be made. Key customer applications were examined to determine how real-world code streams were being executed on the PA 8000.

For the PA 8000, the expectation was set that no code recompilation would be necessary to see a 2× speedup over the PA 7200. We did not want to change this expectation for the PA 8200, so all code experiments were performed using nonrecompiled, nontuned code. It was shown that the PA 8200's performance could be enhanced significantly over that of the PA 8000 by reducing the amount of time the PA 8200 spent waiting for instructions or data. The branch history table (BHT) and translation lookaside buffer (TLB) are architectural features that are intended to reduce wasted cycles resulting from penalties, particularly in pipelined machines. For mispredicted branches, TLB misses, and cache misses, the number of penalty cycles increased from the PA 7200 to the PA 8000. It was expected that a corresponding reduction in mispredictions and misses and the ability to hide penalty cycles using out-of-order execution would result in an overall decrease of wasted cycles. The analysis of the application suite showed otherwise, as the number of wasted cycles increased from the PA 7200 to the PA 8000, accounting for 36 to 86 percent of the total number of cycles spent on each instruction (CPI). If the number of mispredictions and misses could be decreased, a significant performance boost would be realized. As a result, increases in the size of the BHT, TLB, and caches were examined as potential high-benefit, low-risk improvements to the PA 8000.

BHT Improvement

The biggest performance weakness observed was the mispredicted branch penalty. By its nature, out-of-order execution increases the average penalty for mispredicted branches. Therefore, significant design resources were allocated for the PA 8000's branch prediction scheme to lower the misprediction rate, thereby offsetting the higher penalty. The results of the performance analysis revealed that cycles wasted because of branch penalties were still significantly impacting performance. Relative to the PA 7200, the misprediction rate is generally about 50% lower across the sample workload of technical applications. However, the cycle penalty for a mispredicted branch rose by 200%, more than offsetting the reduction in miss rate. There are clearly two possible solutions: decreasing the miss rate or decreasing the miss penalty. Because of the short time schedule of the program, redefining how mispredicted branches are handled to reduce the penalty was not a viable alternative. The more practical solution was to improve branch prediction accuracy.

Improvements to the BHT focused on two areas. The first was the table size and the second was the branch prediction algorithm. The PA 8000 uses a three-bit majority vote algorithm and a 256-entry BHT. Since the PA 8000 also allows up to two branches to retire simultaneously, the table ideally would be able to update two entries per cycle. Parallel BHT update was not implemented on the PA 8000, resulting in the outcome of one of the branches not having its information entered into the BHT. Analysis of this limitation revealed a minor penalty that could easily be eliminated in the PA 8200.

Initial investigation for BHT improvements focused on the size of the table since it is easier to increase the size of an existing structure than to start from scratch and redefine the algorithm. To have the minimum impact on control logic, it was desirable to increase the table size by a multiple of two. Visual inspection of the area around the BHT revealed that the number of entries could be increased to 512 with little impact.

Next, possible changes in the prediction algorithm were explored. Using a more common algorithm became the key to allowing the BHT to grow to 1024 entries. The new algorithm requires only two bits of data compared to the three-bit algorithm implemented on the PA 8000. Analysis of the two algorithms showed that they result in almost the same predictions with only a few exceptions. The reduction in the number of bits per entry from three to two allowed the BHT to grow from 512 to 1024 entries. The increase from the algorithm change was shown through simulation to provide more of an incremental improvement than was lost by the switch to the two-bit algorithm.

One additional improvement was made to the BHT concerning the handling of multiple branches retiring at the same time. Allowing two entries to be updated simultaneously required the data entries to have two write ports. This functionality was not included in the PA 8000, so implementing a two-port solution on the PA 8200 would be very expensive in die area.

Therefore, a control-based solution was devised. When two branches retire on the same cycle, the information necessary to update the cache for one of the branches is held in a one-entry queue. On the next cycle, the data in the queue is used to update the table. If another branch also retires on the next cycle, the queue data is written into the BHT and the newly retiring branch's data is stored in the queue. Only if two branches retire while the queue contains data is the data for one branch lost. This condition is considered to be quite rare, since it requires that multiple pairs of branches retire consecutively. The rarity of this situation makes the performance impact of losing the fourth consecutive branch's data negligible.

The risk involved with making the described changes to the BHT was relatively low. The data storage elements are well-understood structures and could be expanded with little risk. The control for the new BHT could mostly be leveraged from the PA 8000 implementation with the exception of the new branch store queue. Significant functional verification was done to ensure correctness of the new BHT. Since control and data paths remained almost the same as the old BHT, there was high confidence that the changes would not introduce new frequency limiters.

TLB Improvement

The second major area of improvement involved the TLB. Relative to the PA 7200, the PA 8000 uses significantly more cycles handling TLB misses on most of the applications used to analyze performance. The reason for this increase is twofold. First, the penalty for a TLB miss increased from 26 cycles on the PA 7200 to 67 cycles on the PA 8000. The increase in TLB miss penalty was mainly caused by an increase in control complexity resulting from the out-of-order capability of the PA 8000. Second, the TLB miss rate for most of the applications examined also increased. The total number of entries decreased by 20% from 120 to 96 between the PA 7200 and the PA 8000. However, the PA 8000 has a combined instruction and data TLB while the PA 7200 has separate instruction and data TLBs. At the time, a decrease in size seemed an acceptable trade-off since instruction and data TLB entries could now use the entire TLB.

Since the penalty for a TLB miss could not be reduced without significant redefinition of how a TLB miss is handled, the number of entries was the area of focus. Simulation revealed that increasing the number of entries provided a nearly linear improvement in the TLB miss rate, leveling off at about 128 entries. In looking at the area the TLB occupied and the surrounding routing channels, it became clear that 128 entries would involve an unacceptable design risk. Since the implementation is most efficient in multiples of 8, we next examined 120 entries. Initial examination of the artwork showed that this target would be aggressive, yet reasonable. Simulations were done assuming 128 entries to provide some additional timing margin and to allow for increasing to 128 entries if it became possible. Most of the circuit timing paths were found to have nearly the same performance with 120 entries as 96 entries since the critical variable for timing is generally the width of an entry and not the number of entries. Some minor changes to transistor sizing provided the additional margin necessary on critical paths that traversed the TLB array. The goal of these changes was to increase the number of TLB entries over the PA 8000 without impacting speed.

The biggest risk that the TLB changes posed was to the project schedule. The area affected by the changes was much larger than that of any other change, and there were hard boundaries to other functional units that constrained design area. To increase the size of the TLB, two complex signal channels were rerouted. Although necessary to provide the additional room, the changes were time-consuming and presented significant schedule risk. Routing changes also increased the chance of a change in the electrical performance of the affected signals. To minimize this risk, a tool was written to verify that signal integrity was not compromised. Overall, the rerouting of the channels was the critical path to tape release and also the highest risk.

Frequency Improvement

In addition to improving the BHT and TLB performance, the target frequency for the PA 8200 was increased over that of the PA 8000. We took a two-pronged approach to timing analysis. The first approach consisted of analyzing a software model of PA 8000 timing and the second approach consisted of examining data from prototype systems in which we increased the frequency to the failing point.

The PA 8000 timing was modeled using Epic's Pathmill and Timemill suite and Verilog's Veritime. These tools provided an ordered set of paths ranked according to predicted operation frequency. We grouped the data into paths that were internal to the chip (*core paths*) and paths that received or drove information to the cache pins (*cache paths*). It became readily apparent that there was a very small set of core paths and a much larger set of cache paths that could potentially limit chip frequency. The core paths tended to be independent of all other core paths and could be improved on an individual basis within the CPU. The cache path limiters tended to funnel into a couple of key juncture points and could be globally improved by addressing those points. As an additional degree of freedom, cache paths could be addressed through a combination of CPU, board, and cache SRAM improvements.

Once it was determined which core paths might limit chip frequency, we had to devise a method to correlate the simulated frequency with actual chip performance. Targeted tests were written to exercise potential core limiters. Paths were chosen based on their independence from known limiters and for their ability to be completely controlled by the test. The targeted tests ran consistently faster on silicon than the model predicted, giving us confidence that core paths would not be frequency limiters.

We then looked at correlating cache paths between the model and the system. Cache paths tend to be multistate paths dependent on the timing of the cache SRAMs. Because of these attributes, it was not feasible to craft a chip-level test to exercise specific cache paths. Therefore, we decided to rely upon system data for determining worst-case cache paths and then use the model data to show the frequency of other cache paths relative to the worst case. System work revealed two cache path frequency limiters. Both paths were predicted by and correlated with the timing model.

Based on the cache paths exposed through system work, an additional timing investigation was launched. Both paths funnelled into a similar set of circuits to send addresses to the SRAMs. All other inputs into those circuits were examined and individually simulated using SPICE to determine if they had the potential to become frequency limiters. From this effort, one additional set of inputs was identified as having a high risk of becoming a frequency limiter once the known limiters were improved. The proposed improvements to the known limiters improved the newly identified path as well, keeping it from becoming a critical path.

The final step taken to understand the frequency limitations of the PA 8000 was to devise a way to look beyond the known limiting paths in a system. The lowest frequency speed limiter was a cache path related to an architectural feature to improve performance. On the PA 8000, this feature can be disabled. However, the second speed limiter was not programmable and was therefore capable of masking other paths. We turned to focused ion beam (FIB) technology to help us solve this problem.

The second speed limiter was a single-phase path that started with the rising edge of a clock and ended with the falling edge of a derived clock. By delaying the falling edge of the derived clock, we could increase the frequency at which the path could run, creating a new region in which we could search for failing paths. We used the FIB to cut away and rebuild the circuitry for the derived clock. In the process of stripping away the metal on the chip and then redepositing it to rebuild the circuit, resistance is added, slowing down the circuit. We were able to add 220 ps to the path, increasing the failing frequency for this limiter by approximately 22 MHz. The FIB-modified chip was placed in a system for extensive testing. No additional failing paths were found in the newly opened frequency region.

In improving the critical paths for the PA 8200, a conservative design approach was adopted. Most of the improvements involved moving clock edges, allowing latches to update earlier than before. Such changes can expose races or setup violations. The paths were carefully simulated to eliminate the risk of introducing a race. In cases where it was difficult to precisely determine the setup time needed for a signal, conservative changes were made.

Cache Improvement

Yet another area for improvement on the PA 8200 was the cache subsystem. The cache size plays an integral role in determining how well the system performs on both applications and benchmarks. In addition, the off-chip cache access path can limit the operating frequency of the system because of the tight coupling between the CPU and the SRAMs.

The PA 8000 offered a maximum cache size of 1M bytes for both the instruction and data caches. A total of 20 1M-bit industry-standard late-write synchronous SRAMs were employed for this configuration. The printed circuit board design was cumbersome because of the large number of SRAM sites. The design resulted in relatively long round-trip delays. As the PA 8200 was being defined, the next generation of SRAMs became available. These 4M-bit parts were fully backwards compatible with those used with the PA 8000. The emergence of these higher-density components made possible a 2M-byte instruction cache and a 2M-byte data cache while reducing the number of SRAMs to 12. The resulting board layout was more optimal, contributing to shorter routes and better signal integrity.

In addition to cache size, the frequency limitation of the off-chip cache was carefully addressed. For much of the post-silicon verification of the PA 8000, the two-state cache access presented a frequency barrier that limited the amount of investigation beyond 180 MHz. Two main contributors allowed the frequency of the PA 8200 to be increased well beyond 200 MHz. The first was the new SRAM placement and routing for the cache subsystem. The 12-SRAM configuration yielded a new worst-case round-trip delay that was 500 ps shorter than the 20-SRAM configuration previously used. The second enabler was linked to the next-generation SRAMs. Not only did these parts provide four times the density, they also reduced their access times from 6.7 ns to 5.0 ns. The combined benefit of these two enablers resulted in raising the maximum cache-limited frequency from 180 MHz to 230 MHz. The value of this improvement was really twofold. First, it enabled system-level electrical characterization and CPU core speed path identification in a space previously unexplored. Second, it resulted in a manufacturable product that could meet the performance needs of our workstations.

PA 8200 Performance

Under nominal operating conditions of room temperature and 3.3-volt power supplies, the PA 8200 is capable of running up to 300 MHz, 70 MHz faster than its predecessor. Table I summarizes its performance.

Table I
HP PA 8200 CPU Performance

Benchmark	Estimated Performance	Frequency
SPECint95	16.1	230 MHz
SPECfp95	25.5	230 MHz

Conclusion

The HP PA 8000 RISC CPU achieved industry-leading performance across a wide variety of applications by using an aggressive out-of-order design and carefully balancing hardware utilization throughout the system. The PA 8200 leverages that design, improving key areas identified by customer needs and applications. The number of TLB and BHT entries was increased, chip operating frequency was increased, and the cache configuration was updated to include the latest available SRAM technology. Together these changes improved system performance across customer applications up to 23%, once again delivering industry-leading performance.

Acknowledgments

The authors would like to acknowledge the contributions of the technical and management staffs at HP's Engineering Systems, Enterprise Systems, Workstation Systems, Operating Systems, Compiler, and System Performance Laboratories located in Fort Collins, Colorado, Roseville, California, and Cupertino, California. Without the tremendous effort put forth by so many individuals on these projects, the PA 8000 and PA 8200 would not have been possible.

References

1. D. Hunt, "Advanced Performance Features of the 64-bit PA 8000," *Compcon Digest of Papers*, March 1995.
2. A. Kumar, "The Hewlett-Packard PA 8000 RISC CPU: A High Performance Out-of-Order Processor," *IEEE Micro*, April 1997.
3. J. Lotz, G. Lesartre, S. Naffziger, and D. Kipp, "A Quad-Issue Out-of-Order RISC CPU," *ISSCC Digest of Technical Papers*, February 1996.
4. P. Perez, "The PA 8200: A High-Performance Follow-On to the PA 8000," *Microprocessor Forum*, October 1996.
5. N. Gaddis, J. Butler, A. Kumar, and W. Queen, "A 56-Entry Instruction Reorder Buffer," *ISSCC Digest of Technical Papers*, February 1996.
6. W.R. Bryg, K.K. Chan, and N.S. Fidducia, "A High-Performance, Low-Cost Microprocessor Bus for Workstations and Midrange Servers," *Hewlett-Packard Journal*, Vol. 47, no. 1, February 1996, pp. 18-24.

Design Methodologies and Circuit Design Trade-Offs for the HP PA 8000 Processor

This paper discusses the various design methods used in the PA 8000, specific design techniques for the new packaging technology, the clock distribution scheme, cross-chip signal integrity issues, and some of the new tools and techniques.

by Paul J. Dorweiler, Floyd E. Moore, D. Douglas Josephson, and Glenn T. Colon-Bonet

The increasing demands for greater processor performance to remain competitive in today's computer market necessitate careful attention to the methods used in designing processors to achieve these performance goals. Processor designs are increasing in complexity to meet performance goals, with such features as out-of-order execution and superscalar operation. Design cycles are decreasing in length, so design quality must increase as well. All of these factors call for new design techniques to ensure continued success.

This paper will present some of the design methodologies and choices used in the design of the HP PA 8000 CPU, the first HP processor to implement the PA-RISC 2.0 architecture and the first capable of 64-bit operation. The various design methods used in the PA 8000, specific design techniques for the new packaging technology used, the clock distribution scheme, and cross-chip signal integrity issues will be discussed. We will also present some of the new tools and techniques employed by HP to ensure a high level of quality on first silicon, based in large part on our experiences with previous PA-RISC microprocessor designs.

Design Trade-Offs and Methodologies

Processor design is a continuous series of trade-offs between die area, complexity, performance, speed, power use, and design time. Given the complexity of a four-way out-of-order processor such as the PA 8000, it is not appropriate to employ the same circuit design techniques for all blocks on the chip. For the PA 8000, three major circuit design techniques were used.

The first is the traditional *static design* approach, in which all output signals are held true as long as the inputs to the static cell remain constant. Storage of values, or *state*, is in latches, and logic functions are implemented using a variety of different logic blocks, allowing minimization of area or path evaluation time. Since static logic is fairly immune to noise effects (at least on a local basis), this is the safest design approach. Frequently this is also the design approach that needs the fewest engineering resources. The synthesis and layout steps can be accomplished by automated tools, with oversight by the designer to ensure that the block satisfies requirements, timing paths are met, electrical rules (such as metal electromigration) aren't violated, and so on.

Static design techniques are not ideally suited for large fan-in and fanout functions. Because of their pullup/pulldown design, static gates are not the fastest evaluation method for certain high fan-in/fanout applications. *Single-rail dynamic logic* or *domino logic* is better suited to these applications, particularly OR functions. A good example of such a function is the operand dump lines from register files. For an out-of-order processor with operand data coming from both rename and architected state registers, the number of drivers on one bus is quite large. In the case of the PA 8000 there are 56 rename registers and 32 architected state registers on both the integer and floating-point sides. Trying to drive a single bus with 88 static drivers is a much more difficult task than using single-rail dynamic logic. The lower capacitance of simply using an n-channel FET driver and a bus precharger for the nondump state helps tremendously in this instance. Static logic will also consume more area to implement these types of functions because it requires extra p-channel FET pullup trees in each block. However, dynamic logic is more susceptible to noise, requires more careful design attention than static logic, will in general use more power, and since it is a clocked mechanism, also increases the clock load. This type of logic is employed in the data path portions of the PA 8000.

Single-rail dynamic logic does fail in some instances, particularly when trying to use the inversion of a value in the middle of a logic chain, or using an AND function. In this instance and where static logic is not fast enough, a *dual-rail dynamic logic* scheme can be employed. In this type of logic, both the positive sense and the negative sense of a signal are derived, both in a low-go-high fashion.* Inversions are accomplished simply by switching the low-sense and high-sense signals between gates. This logic can be quite fast since the design of the gates optimizes one transition edge and dynamic techniques are employed in the pulldown trees of the logic gates. In addition, since timing information is included with the transition of one or the other output sense, it is a self-timed mechanism. By employing latches that sense just the first transition of an output

* Low-go-high means that the signal starts at the ground voltage and transitions only once during an evaluate state to the supply voltage V_{DD}.

pair, this type of logic can be pipelined and used in multiple stages. Dual-rail dynamic logic does consume a large amount of area and power, and therefore was employed only in the most time-critical portions of the PA 8000, most notably the floating-point execution units.

Alpha Particle Sensitivity

The decision to use lead solder bump technology to enable flip-chip die attach for the PA 8000 presented a new design challenge for the team. Previous designs were all wire-bonded dice in ceramic pin-grid array packages (CPGA). To prevent alpha particles (which are identical to helium nuclei) emanating from the package or wire bonds from upsetting sensitive storage nodes within the processor, a silicon compound is used on the die surface. The flip-chip attach method, however, places arrays of mostly lead (Pb) hemispherical bumps over a significant portion of the die surface. The bump material contains some heavy elements that are radioactive and the decay of these elements produces alpha particles and beta and gamma rays that can cause a *single-event upset* of a sensitive storage node.

The single-event upset is a high concern in integrated circuits because a change of state of a storage node can have serious consequences for executing programs. Any alpha particle that leaves the solder bump has sufficient mass and energy to cause an ionized trail of hole-electron pairs that create mobile charges that can flood a positively charged storage node and cause an unintended state change of a memory element. To minimize this undesired event, certain design changes were adopted for PA 8000 memory circuits.

A SPICE current pulse model that simulated the behavior of an alpha particle was derived from both empirical measurements on existing products and simulation using IC process modeling software. A design rule for the minimum storage charge ($Q_{critical}$) was set and all storage nodes were designed to meet the new guideline, then verified by SPICE simulations using the alpha particle current pulse model.

Clock Distribution Scheme

In a high-frequency design such as the PA 8000, minimizing cross-chip clock skew is critical to ensure the maximum amount of time for logic and data path operations to complete. Lack of attention to clock distribution for the entire chip will result in a lower frequency of operation and more design resources being spent on reducing delays in budgets that contain cross-chip paths. Excessive clock skew also increases the likelihood of introducing races into the design that will need to be identified and fixed. For these reasons a considerable amount of effort was spent in the investigation and design of the clock distribution scheme for the PA 8000.

Also affecting clock skew across the chip is the amount of load on the global clock signal. With single-rail and dual-rail dynamic circuitry in the data path sections, the overall clock load is greater than it would have been had only static circuitry been used. This places an additional burden on the clock distribution network because skew increases with load for a given clock network definition.

The clock distribution method employed on the PA 8000 is an H-tree metal structure (see Fig. 1) to deliver the clock signal from the C4 solder bumps to a first-level on-chip clock receiver. The output of this receiver is then routed using matched wire lengths to a second level of clock buffers, with each buffer carefully positioned on the chip and the output load of each buffer matched as closely as possible. Given the large size of the die for the PA 8000 (19.2 by 17.8 mm), process variation will inevitably make the FETs used in these second-level clock buffers unequal in strength. The design of these buffers attempted to minimize this speed variation. A graph of the overall skew using the final clock distribution scheme is shown in Fig. 2. Using this design, the overall clock skew across the die was held to 170 picoseconds.

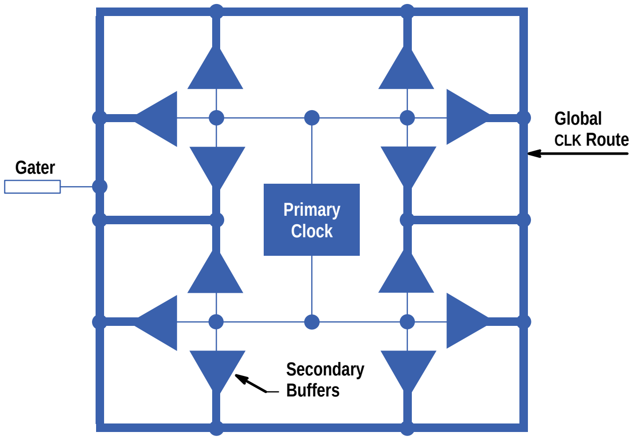


Fig. 1. *H-tree distribution network.*

From the second-level clock buffers, careful attention was paid to the routes of the buffered clock outputs to the next level of circuitry. To minimize the power dissipation of the chip and provide nonoverlapping clocks to control blocks, controlled

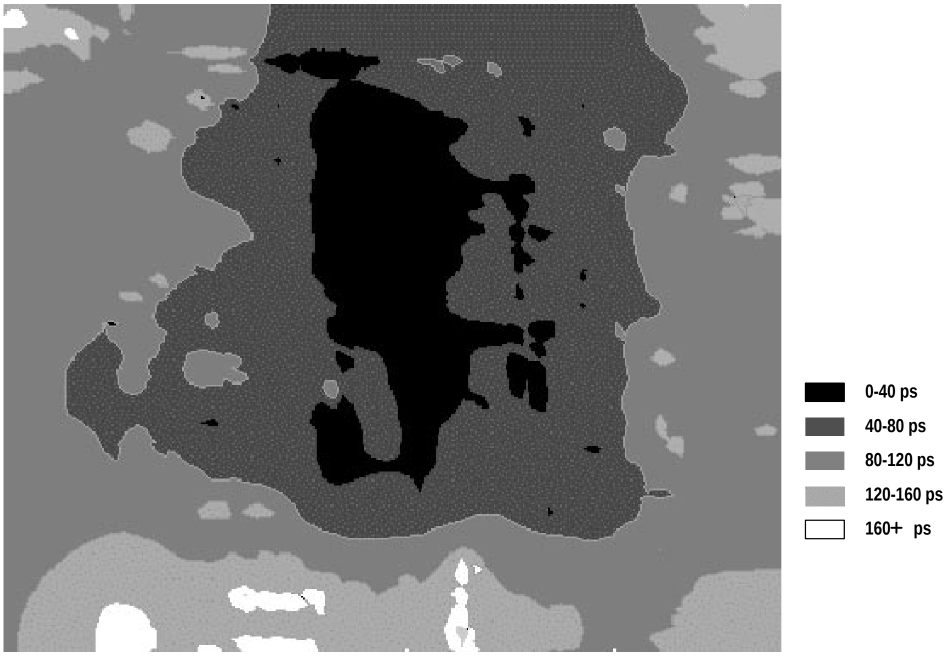


Fig. 2. *Clock skew topography map.*

buffer blocks called *clock gaters* are employed. Different types of clock gaters can generate overlapping and nonoverlapping clocks, and each size of gater is rated for a specific amount of output load. Checks were performed to ensure that the proper loading was maintained on all gater outputs, since the clock outputs for these gater blocks were guaranteed to a certain specification only under a rated load range. Whenever possible, the clock gaters were qualified with control signals to strobe their clock outputs only when necessary. This allows the clocks for various functional units to be clocked only when actual work needs to be done, reducing overall chip power dissipation.

Timing

To ensure high-frequency operation and a short post-tape-release period, vigorous timing checks were employed by PA 8000 block and top-level designers. The timing effort on the PA 8000 was far greater than on previous HP processors, and was a significant factor in producing first silicon that ran at the targeted design frequency from the first boot of the operating system.

The size of the die complicated top-level timing analysis because the sheer distance some signals had to travel added significant delay to cross-chip budgets. Over-the-block routing was necessary given the large number of top-level signals present on the chip. Noise and capacitance to metal layers inside of the blocks being routed over had to be factored into the top-level timing analysis.

Repeaters were employed on the PA 8000 for long-route, timing-critical signals to reduce the delay and allow for faster signal edges. In some cases this was accomplished with one noninverting buffer, and in other cases split inverters along the route were used. Where possible, single inverters were used in cross-chip paths if this level of inversion could be absorbed by the receiving or driving logic, thus speeding up these paths.

Block designers ran timing simulators, both path-driven and stimulus-driven, to check the internal timing of their blocks and to verify that their published drive and receive times for global signals were valid. Close to tape release, a large effort was put into driving down the number of slow cross-chip paths, which threatened the frequency goal of the PA 8000.

In addition to the timing checks performed on the PA 8000, other quality checks were performed to detect potential problems discovered on previous processors. The checks will be described in the remainder of this article. Most of these problems are related to noise events on signals and supplies that trip sensitive circuitry, causing failures.

Latch Margin Checks

Latches are an important part of any processor design. A large amount of state information about a currently running program needs to be stored. Control logic and data paths both employ latches to a large degree. Latch designs trade off setup, hold, and in-to-out delay times by optimizing the size of various FETs in the latch structure, particularly the feedback inverter, which holds the state of the latch and must be overcome to change the state. The PA 8000 design employs transparent latches in which the input signal passes through a series n-channel FET and thus suffers a gate threshold voltage drop as well.

Since changing the state of a latch inadvertently is potentially disastrous, avoiding poor latch designs was a critical design goal. For this reason, a specific tool was developed to analyze the electrical margins of a latch and was run on all the latches on the PA 8000. The complexity of this tool grew from a desire to be able to check both full and half latches. A full latch consists of two cross-coupled inverters while a half latch has a single FET connected to the inverter output (see Fig. 3).

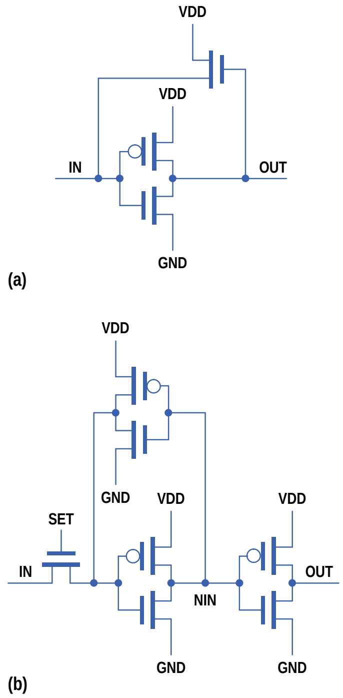


Fig. 3. *Two types of latches. (a) Half latch. (b) Full latch.*

The latch check program evaluated the set drive path to determine if it was strong enough to overcome the feedback FETs. Since the input drive signal must be known to accomplish this evaluation and extracting this drive signal from all of the places where latches are used is a rather complex task, the program had to make some assumptions about the driving block when run only on the latch cell. For critical paths or latches with particularly small margins, the actual driving path was placed into a small schematic and the program was run on this schematic to ensure that the latch was acceptable.

Signal Noise Checks

In implementing the PA 8000, additional levels of interconnect were required with finer geometries than had been used on past designs to connect the blocks on the chip together. This posed a number of problems in guaranteeing that the design would be electrically robust at the high frequencies at which the PA 8000 operates. Experience during electrical characterization of previous designs indicated that internal signal integrity would be a serious issue for the PA 8000.

Signal Integrity Issues in Advanced Processes

Three major problems arise with interconnect as processes continue their inexorable march toward smaller dimensions and higher frequencies:

- Signal cross talk is very significant at the 0.5-μm process generation and beyond.
- Signal rise and fall times decrease as transistor speed increases.
- Signal coupling increases because smaller dimensions are used for interconnect. The smaller dimensions especially increase coupling between metal lines on the same interconnect layer.

Signal cross talk (noise effects) includes both capacitive and inductive components. In the equations $i = Cdv/dt$ and $v = Ldi/dt$, all of the factors—C, L, dv/dt, and di/dt—are increasing with decreasing interconnect dimensions and faster transistors. This leads to voltage and current disturbances in lines that couple to adjacent metal lines through mutual capacitive and inductive effects. An example of an interconnect and circuit topology that can cause these problems is shown in Fig. 4.

Very fast edge rates require high transient currents (tens of amperes) from the off-chip and on-chip power networks. High currents are also present in the main clock network on the chip. Power supply networks require careful design to minimize inductive and capacitive effects on voltage levels. Clock nets also need to maintain good voltage levels as well as minimize clock skew delays between various blocks.

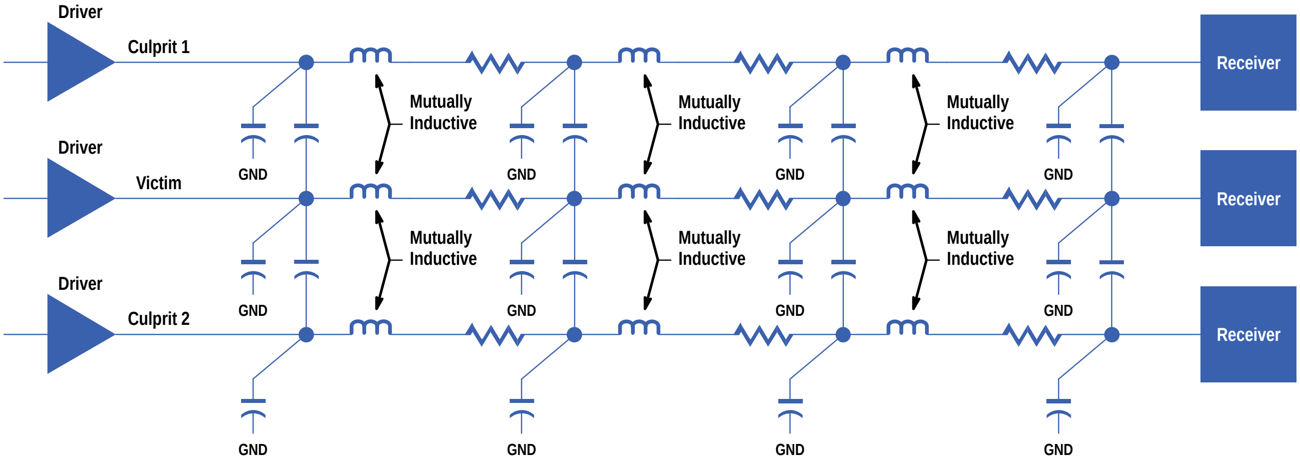


Fig. 4. *Interconnect topology causing crosstalk problems.*

Solving Signal Integrity Problems

Different approaches can be used to solve signal integrity problems. In general, combinations of the following techniques were used on the PA 8000:

- Adjust spacing of signals relative to each other
- Include shields above and below signals
- Include restoring logic (repeaters) in the route
- Design signal receivers that reject noise events.

A key component of the effort to correct signal integrity problems is a toolset that can be used to identify them in the first place. This toolset needs the ability to do RC extraction and the ability to identify circuit topologies that may be susceptible to noise problems. RC extraction allows determination of the extent of possible coupling problems. By combining it with identification of susceptible circuits, solutions to problems can be implemented.

To identify circuits with noise susceptibility, an existing internal tool was heavily modified and extended to allow easy traversal of the current schematic or artwork netlist hierarchy. This tool could display all connections of a given signal down to the transistor level, including information on FET sizes and estimates of capacitive loading (from schematics) or extracted capacitive load (from artwork). Information on port directionality and other text properties added by the block designer could also be displayed, as well as what terminals of a FET are connected to the signals. One additional important feature of the tool was that it could track any changes in real time, as soon as they were made by designers. This tool was used for many purposes by designers in addition to its use in noise checks.

The latching methodology used on the PA 8000 has a potential failure mode: excursions of a signal beyond a supply rail (e.g., below local ground for a given latch) could cause the latch to lose its value. An example of this is illustrated in Fig. 5. The latch shown is holding a high value—node IN1 is at V_{DD}, held by the weak feedback inverter. If the victim line is at 0V and

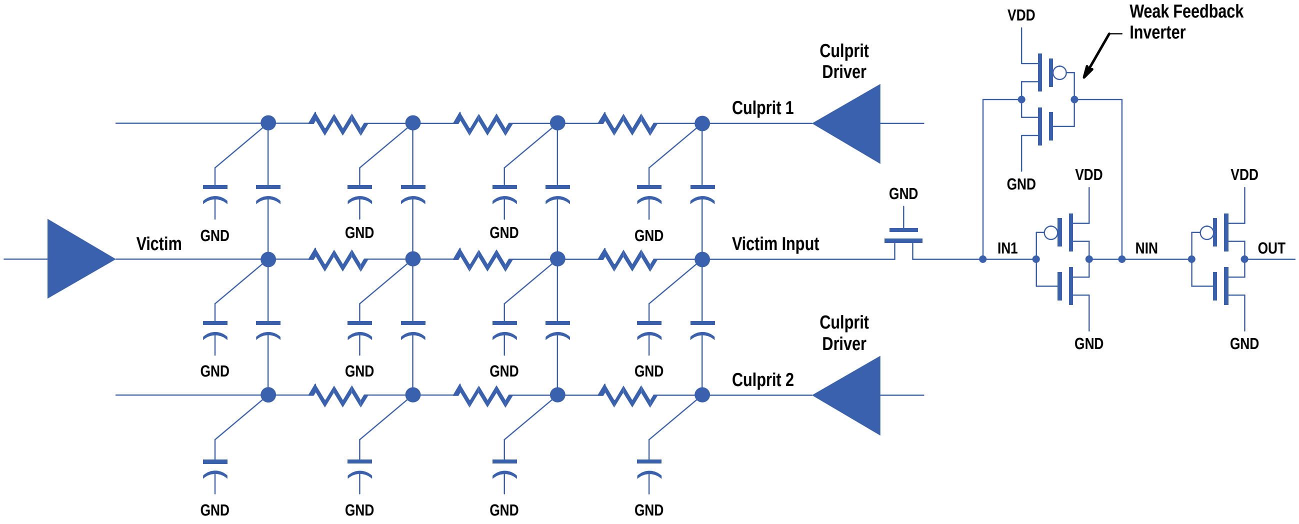


Fig. 5. *Latch failure caused by cross-chip noise.*

the culprit lines are at V_{DD} and transition to 0V quickly, an excursion of the input signal below local ground is possible, induced by capacitive coupling from the culprit lines to the victim line as the culprit lines transition from 1 to 0. This input signal excursion can cause the n-channel FET pass gate that serves as the input to the latch to turn on even though its gate is held at 0V (V_{GS} for the transistor is greater than V_{TN}). This is because the victim input is temporarily below local ground. With this n-channel FET pass gate on, the latch can spuriously dump the value it was holding by discharging the IN1 node if the transient is enough to overcome the feedback inverter and trip the forward inverter. This type of failure may change the state of the chip, and is a serious problem that must be avoided.

Other possible problem circuits were also identified by this tool, including heavily ratioed* combinations of p-channel FETs and n-channel FETs and long routes connected to gate inputs of pass FET latches. However, diffusion-connected inputs were the most common problems. To identify diffusion-connected inputs, the netlist traversal tool was run on every top-level signal in the design. The tool identified top-level signals connected to the source or drain of a pass FET in a latch. This gave a textual report of all connections down to the FET level for every top-level signal, in addition to the FET terminal connections and whether the signal was an input or output of that particular leaf cell.

Once the report was generated, a parser analyzed the connectivity to determine if any signal connected to a FET diffusion was also an input (outputs of leaf cells were ignored). Other checks were performed for additional suspect circuit topologies. When potential problem signals were identified, the information was integrated with RC extraction results to determine priorities for fixing signals, and the results were distributed to designers to give them feedback on which signals in their blocks needed to be fixed. Extensive simulations showed that only routes longer than a specified threshold length would need to be fixed. This threshold gave designers a limit at which they would have to do something to reduce susceptibility to noise on a signal being received by their block.

In most cases, designers used one of the techniques described above to alleviate these noise problems. The most popular solution inserted a restoring inverter in front of the pass gate and modified the latch slightly to make it logically equivalent to the latch that needed to be replaced, as shown in Fig. 6. The restoring inverter in front of the pass FET makes the latch far more immune to noise events on the input. At other times, repeaters (inverters and buffers) were inserted in routes to cut down the distance of the route, thus reducing the susceptibility of a given line to transitions by its neighbors.

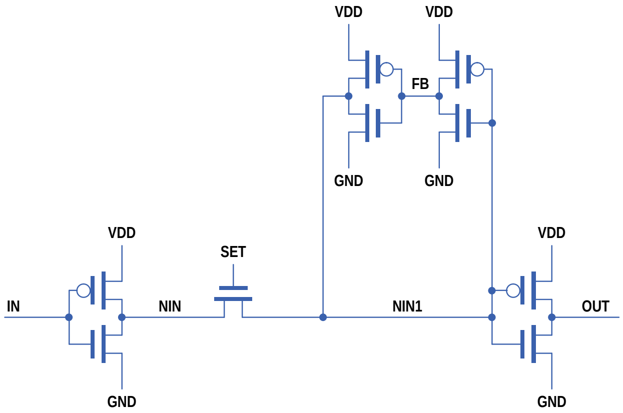


Fig. 6. *Input noise resistant full latch.*

Signal Integrity Results

Overall, the techniques described above were effective in eliminating noise-induced electrical failures in the PA 8000 design, and probably saved several months of characterization to investigate noise failures that would have existed had this tool not been developed. Over 7000 potential problems were flagged with the first run of the tool. All of these problems were investigated and either fixed or waivered before tape release. The PA 8000 was a very electrically robust design given its complexity level when silicon was received.

One drawback of this tool was that it was only run on top-level signals. Since some of the blocks on the PA 8000 were very large, long routes and therefore noise problems could also be embedded inside blocks. One such problem was found during characterization of the chip at the block level. We are currently extending the noise analysis tools to operate at deeper levels throughout the chip hierarchy to thoroughly check all signals on the chip. RC extraction is being extended to allow deeper levels of extraction without long run times, and inclusion of inductive effects is also being investigated.

A limitation of this type of tool is that it can generate a lot of noise, that is, report problems that really aren't problems. This affects designer productivity because the problems reported by the tool must be investigated. However, the penalty and cost for finding a noise problem in a design can be very high, especially late in the characterization process, so effort spent early

* Heavily ratioed combinations are combinations of inverters and other FETs in which the effective p-channel FET drive strength is significantly different from the effective n-channel FET drive strength.

to eliminate possible problems is very worthwhile. We are currently developing more advanced tools to eliminate some of this noise and make sure that only problems serious enough to warrant fixing are included.

Block Quality Checks

Block design, especially for complex blocks, is a time-consuming process in which—despite the best intentions of the designer—problems can sneak through without being noticed. For this reason several additional tools were developed to allow designers to check for potentially troublesome circuits in their blocks.

One tool checks for so-called “ugly” polysilicon structures. Given the resistance of the polysilicon layer in the HP process used to fabricate the PA 8000, long polysilicon routes are undesirable and can cause numerous problems, chief among these being slow speed. Standard cell routed blocks suffered less from this problem because the routers employed used only metal layers for signal interconnect. Long polysilicon problems occurred primarily in semicustom and full-custom designs. This tool flagged polysilicon routes between 25 and 50 micrometers long as warnings and over 50 micrometers as errors.

With the significant use of clock gaters to create many different flavors of clocks, both overlapping and nonoverlapping, races were expected to be more prevalent in the PA 8000 design. Pass-gate blocks in particular cause these types of problems. Clock-qualified signals (signals derived from clock edges) driving other clock-qualified nodes were checked to cover signal races not detectable by the previous race checking methodology used in PA-RISC processor designs.

Summary

All of the techniques described above helped to make the PA 8000 processor a successful project, achieving its frequency, performance, and aggressive post-tape-release schedule. This was a great achievement given the sheer complexity of the design, the fact that it was a new processor architecture, and the number of new technologies employed in the design. This success is due in large part to the design methodologies used for this processor, particularly the new methodologies developed for the PA 8000 design.

Acknowledgments

The authors wish to thank the entire design and engineering team for the PA 8000, particularly John Spencer for contributing the alpha particle discussion in this article. We also wish to thank Shaun McEvoy and Jim Wilmore for their help with the latch checking program.

Functional Verification of the HP PA 8000 Processor

The advanced microarchitecture of the HP PA 8000 CPU has many features that presented significant new verification challenges. These include out-of-order instruction execution, register renaming, speculative execution, four-way superscalar operation, decoupled instruction fetch, concurrent system bus interface, and PA-RISC 2.0 architecture enhancements. Enhanced functional verification tools and processes were required to address this microarchitectural complexity.

by Steven T. Mangelsdorf, Raymond P. Gratias, Richard M. Blumberg, and Rohit Bhatia

Computer system performance has been improving recently at a rate of 40 to 60 percent per year. This growth rate has been fueled by several factors. Advancements in integrated circuit technology have made higher microprocessor clock rates and larger caches possible. There have been contributions from system software as well, such as compilers that emit more efficient machine code to realize a given function. The PA-RISC instruction set architecture has evolved to keep pace with changes in technology and customer workloads.

These factors alone, however, would not have been sufficient to satisfy customer demand for increased performance in a very competitive industry. The balance has been made up by innovations in microarchitecture that increase the amount of useful work that a microprocessor performs in a clock cycle. This has increased the complexity of the design and thus the effort required for successful functional verification.

Many of our previous microprocessor projects have reused existing cores (although generally with significant modifications and enhancements). In contrast, the HP PA 8000 CPU has a new microarchitecture that borrows little from previous projects. Some of the features in its microarchitecture presented significant new verification challenges:

- Out-of-order execution. A 56-entry queue of pending instructions is maintained by an *instruction reorder buffer* (IRB). The queue hardware selects instructions for execution that have their operands available irrespective of program order.
- Register Renaming. Write-after-write and write-after-read ordering dependencies are eliminated by remapping references from an architectured register to a temporary register.
- Speculative Execution. The PA 8000 predicts whether a branch is taken and can tentatively execute instructions down the predicted path. The side effects of all such instructions must be canceled if the prediction turns out to be incorrect.
- Four-Way Superscalar Operation. The PA 8000 has ten functional units and can sustain an execution rate of four instructions per cycle.
- Decoupled Instruction Fetch. Instructions are fetched and inserted into the queue by an autonomous *instruction fetch unit* (IFU). The IFU performs branch prediction and caches the target addresses of recently taken branches in a *branch target address cache* (BTAC).
- Concurrent System Bus Interface. Memory requests can be issued out of order, and data returns can be accommodated out of order. Up to 16 requests can be outstanding at a time.
- PA-RISC 2.0 Architecture Enhancements. These provided important new capabilities, such as 64-bit addressing and computation, but they necessitated tool rework and limited reuse of existing test cases.

This paper describes the enhanced functional verification tools and processes that were required to address the daunting microarchitectural complexity of the PA 8000.

Verification Overview

The purpose of functional verification is to identify defects in the design of a microprocessor that cause its behavior to deviate from what is permitted by the specification. The specification is the PA-RISC instruction set architecture and the bus protocols established by industry standards or negotiated with the designers of other system components. Performance specifications, such as instruction scheduling guidelines committed to the compiler developers, may also be considered.

Although it is not possible to prove the correctness of a microprocessor design absolutely through exhaustive simulation or existing formal verification techniques, a functional verification effort must achieve two things to be considered successful. First and foremost, it must provide high confidence that our products will meet the quality expectations of our customers. At the same time, it must identify defects early enough in the design cycle to avoid impacting the product's time to market.

A typical defect caught early in the design cycle might cost only one engineering day to debug and correct in the RTL (Register Transfer Language). Close to tape release, it might take five to ten days to modify transistor-level schematics and layout, modify interblock routing, and repeat timing analysis. Therefore, tape release can be delayed if the defect rate is not driven down quickly.

After tape release, lost calendar time is the primary cost of defects because the time required to fabricate a new revision of the design is at best a few weeks and at worst a few months. Defects that are so severe that they block a software partner's development, tuning, or testing efforts can put them on the critical schedule path of the product. The worst-case scenario is a masking defect that blocks further testing efforts for a certain functional area of the design, and this delays the discovery of additional defects by the time required to fabricate a new revision. One or more masking defects in series can quickly devastate the product schedule.

The PA 8000 verification effort consisted of a presilicon phase and a postsilicon phase. The purpose of the presilicon phase was to find defects concurrently with the design, when the cost of correcting them was small, and to drive up the quality level at first tape release so that the first prototypes would be useful to our software partners. This was done using three tactics: RTL simulation, accelerated simulation, and switch-level simulation. The postsilicon effort consisted of aggressive characterization of hardware prototypes to complete verification before systems were shipped to customers. Also, performance verification was done at various stages in the project.

RTL Simulation

Most previous PA-RISC microprocessor projects have built their functional verification efforts around an internally developed RTL simulator that compiles RTL descriptions of blocks into C code which are then compiled with HP's C compiler. Block execution is scheduled dynamically using an event-driven algorithm. This simulation technology achieves modest performance (about 0.5 Hz running on a typical workstation), but it does provide capabilities for rapid prototyping such as the ability to simulate very high-level RTL and quick model builds. Therefore, our RTL simulator became the cornerstone of our verification effort early in the design.

Fig. 1 shows the verification environment used for RTL simulation. There are four basic components in the environment:

- The RTL model for the PA 8000.
- Bus emulators, which can apply interesting stimulus to the input buses of the PA 8000 including responses to its transactions. We included emulators for all components sharing the system bus including the memory system, I/O adapter, and third-party processors.
- Checking software, which monitors the behavior of the PA 8000 and verifies that it complies with the specifications. This also helps speed debugging by flagging behavioral violations as soon as they occur.
- A variety of test case sources and tools that can compile the test cases into an initial state for the PA 8000 model and configure the bus emulators.

Checking Software

The most important check is a thorough comparison between instructions retiring in the PA 8000 model and instructions retiring in the PA-RISC architectural simulator. *Retiring* means exiting the instruction reorder buffer, or IRB (see ***Article 1***). A tool called the *depiper* captures information about each instruction retiring in the PA 8000 model, including what resources (such as destination registers) are being modified and the new values. The synchronizer compares this with similar information obtained from the PA-RISC architectural simulator which is also running the same test case. This provides very high confidence that the PA 8000 complies with the basic PA-RISC instruction set architecture. A final-state comparison of all processor and memory state information is also done at the end of each test case.

The depiper also provides the synchronizer with information about architecturally transparent events such as cache misses. Using this information, the synchronizer can perform strong checks in the areas of cache coherency, memory access ordering consistency, and memory-to-cache transfers. In addition, a number of checkers were developed for other areas:

- A checker for the instruction queues, including whether the order in which instructions are sent to functional units complies with data dependencies
- A checker for protocol violations on the system bus
- A checker for the bus interface block, discussed in more detail below
- A checker that detects unknown (X) values on internal nodes.

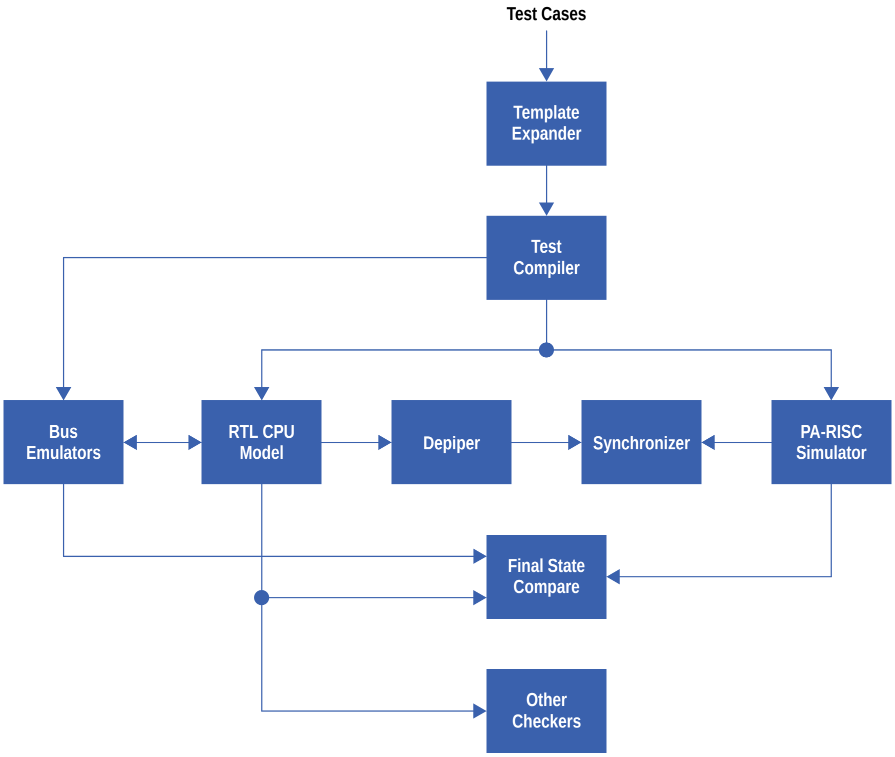


Fig. 1. *Block diagram of the presilicon RTL (Register Transfer Language) verification environment.*

Test Case Sources

A test case is essentially a test program to be run through the RTL model of the processor to stress a particular area of functionality. These are generally written in a format similar to PA-RISC assembly language, with annotations to help specify initial cache and TLB contents. In addition, a control file can be attached to a test case to specify the behavior of the bus emulators. The emulators have useful default behavior, but if desired the control files can precisely control transaction timing.

A test case is compiled using a collection of tools that includes the PA-RISC assembler. The result of the compilation is a set of state initializations for the RTL model. These include the processor registers, caches, TLB, and memory. In addition, the bus emulators are initialized with the commands they will use during execution of the test case.

Previous PA-RISC microprocessor projects had built up a library of test cases and architectural verification programs (AVPs). Although we did run these, it was clear from the beginning that a large source of new cases would be required. The existing cases were very short, so their ability to provide even accidental coverage for a machine with a 56-entry IRB was questionable. Moreover, we needed cases that targeted the unique microarchitectural features of the PA 8000.

We developed a test case template expander to improve our productivity in generating the large number of cases required. An engineer could write a test template specifying a fundamental interaction, and the tool would expand this into a family of test cases. Some of the features of this tool included:

- The ability to sweep a parameter value. This was often used to vary the distance between two interacting instructions.
- The ability to fill in an unspecified parameter with a random value.
- An if construct, so that a choice between two alternatives could be conditional on parameters already chosen.
- Instruction groups, so that an instruction could be specified that had certain characteristics without specifying the exact instruction.

We also used the pseudorandom code generator and test coverage measurement techniques discussed below in the RTL simulation environment. To improve our coverage of multiprocessor functionality, we configured our bus emulators to generate random (but interacting) bus traffic.

Structural Verification

A block can be described by a single large RTL procedure or by a schematic that shows the interconnection of several smaller blocks, each of which is described by RTL. At the beginning of the project, RTL tends to be written at a high level because it can simulate faster and is easier to write, debug, and maintain when the design is evolving rapidly. Block designers, however, have a need to create schematics for their blocks, so there is a risk that these will diverge from the RTL reference.

We considered three strategies to verify that the two representations of the block were equivalent. The first, formal verification, was not pursued because the required tools were not yet available from external vendors. The second strategy was to rely on the switch-level verification effort. This was unattractive because defects would be found too late in the design cycle, and the planned number of vectors to be run might not have provided enough coverage. The strategy selected was to retire the higher-level RTL description and replace it in the RTL model with the lower-level representation. The more timely and thorough verification that this provided compensated for some disadvantages, including slower simulation and more difficulty in making changes. We also used this strategy selectively, relying on switch-level simulation to cover regular blocks such as data paths with little risk.

Divide and Conquer

In any large design effort, one faces a choice of whether to verify components individually, together, or both. Verifying a component separately has several potential advantages. Simulation time is greatly reduced. Input buses can be directly controlled, so effort need not be expended manipulating the larger model to provide interesting stimulus. Finally, dependencies between subprojects are eliminated.

For separate verification to succeed, the interfaces to other components must be very well-specified and clearly documented. Investments must be made in a test jig to provide stimulus to the component and in checking software to verify its outputs. In addition, some portion of the verification must be repeated with all components integrated to guard against errors in the specifications or different interpretations of them.

The PA 8000's bus interface block was particularly well-suited to separate verification. The block had clean external interfaces but contained a lot of complexity, including the hardware to manage multiple pending memory accesses. A software checking tool was written to monitor the block's interfaces and verify its operation. Checking that a request on one bus ultimately results in a transaction on the other bus is a simple example of numerous checks performed by this tool. A very low defect rate demonstrated the success of the divide-and-conquer strategy for this block.

Most of our remaining verification effort was focused on the complete PA 8000. As a final check, a system-level RTL model was built that included several processors, the memory controller, the I/O adapter and other components. Although throughput was very low, basic interactions between the components were verified using this model.

Accelerated Simulation

The speed of the RTL simulator was adequate to provide quick feedback on changes and for basic regression testing, but we lacked confidence that on a design as complex as the PA 8000 it would be sufficient to deliver an adequate quality level. We saw a strong need for a simulation capability that was several orders of magnitude faster so that we could run enough test cases to ferret out more subtle defects. We considered two technologies to provide this: cycle-based simulation and in-circuit emulation.

Cycle-based simulation provides a much faster software simulation of the design. With an event-driven simulator such as our RTL simulator, a signal transition causes all blocks that the signal drives to be reexecuted, and any transitions on the outputs of these blocks are similarly propagated until all signals are stable. The overhead to process every signal transition, or event, is fairly high. Cycle-based simulators greatly improve performance by eliminating this overhead. The design is compiled into a long sequence of Boolean operations on signal values (AND, OR, etc.), and execution of this sequence simulates the operation of the logic in a clock cycle. The name cycle-based simulator comes from the fact that the signal state is only computed at the ends of clock cycles, with no attempt to simulate intermediate timing information. Our investigation revealed that speedups of 500 times were possible, so a simulation farm of 100 machines could have a throughput on the order of 25,000 Hz. The biggest drawback of this strategy was that cycle-based simulators were not yet available from external vendors.

With in-circuit emulation, the gates in a Boolean representation of the design are mapped onto a reconfigurable array of field-programmable gate arrays (FPGAs). The design is essentially built using FPGAs, and the emulated processor is connected to the processor socket in an actual system. The clock rate of the emulation system is on the order of 300,000 Hz, so very high test throughput is possible. It is even possible to boot the operating system. Unfortunately, there were many issues involved in using in-circuit emulation successfully:

- Custom printed circuit boards would have to be designed for the caches, large register files, and any other regular structures that consume too much emulation capacity. Changes in the design would be difficult to accommodate.
- A system was needed to exercise the emulated processor, including a memory controller and I/O devices. Firmware and hardware tinkering would have been needed to make this system functional at the slow clock rates required by the emulation system.

- Productivity was reduced by long compile times and limited observability of internal signals. Only one engineer at a time could use the system for debugging.
- The strategy was difficult to extend to multiprocessor testing. It was prohibitively expensive to emulate multiple processors. We planned to use a software emulator to create third-party bus traffic and verify the processor's responses, but there was a risk that the software's performance would throttle the emulation system's clock rate.
- The emulation system was a very large capital investment.

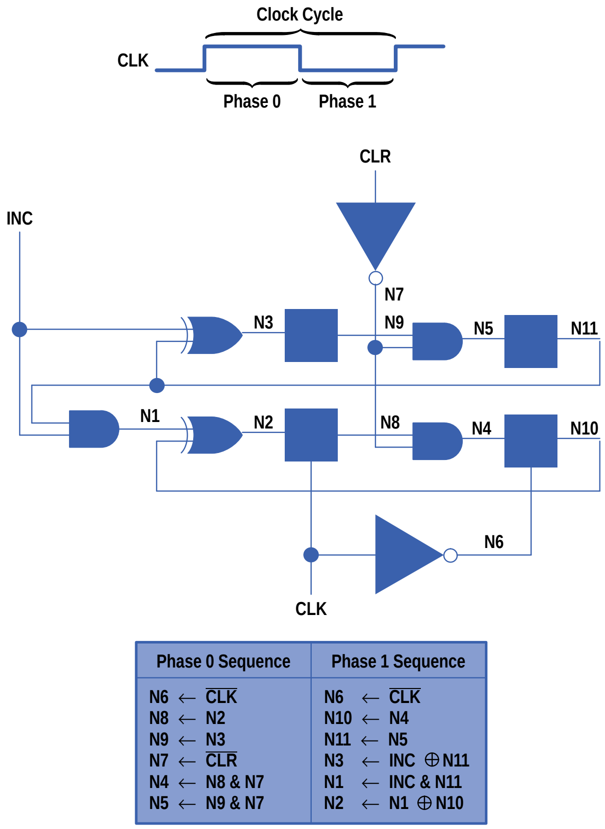
We were quite wary of in-circuit emulation since its use on a previous project had failed to make a significant contribution to functional verification. We were also willing to give up the performance advantage of in-circuit emulation to avoid tackling the ease-of-use issues. The decision to use cycle-based simulation would have been simple except that it meant that we would have to develop the simulator ourselves. R&D organizations in HP are challenged to focus on areas of core competency and look to external vendors to fulfill needs such as design tools that are common in the industry. We did select cycle-based simulation because we were confident that its lower risk and higher productivity would translate into a competitive advantage.

We were careful to reuse components wherever possible and to limit the scope of the project to providing the tool functionality required to verify the PA 8000. We did not attempt to create a simulation product useful to other groups within HP. This turned out to be a good decision because comparable tools have recently started to become available from external vendors.

Cycle-Based Simulation Compiler

The cycle-based simulation compiler operates only on simple gate-level primitives such as logic gates and latches, so higher-level RTL must first be synthesized into a gate-level equivalent. We had to develop our own translator for this because the RTL language used by our RTL simulator was defined before the industry standardization of such languages. Another simplification is that signal values are limited to 0 and 1, with no attempt to model an unknown (X) state.

Fig. 2 shows a simple example circuit, a two-bit counter, that we will use to illustrate the compilation process. The user must describe to the compiler information about the circuit's clocks. The clock cycle is broken down into two or more phases, with the state of the clocks fixed during each phase. This circuit has a clock cycle of two phases, and the clock (CLK) is low during the first phase and high during the second phase.



Phase 0 Sequence	Phase 1 Sequence
N6 ← $\overline{\text{CLK}}$	N6 ← $\overline{\text{CLK}}$
N8 ← N2	N10 ← N4
N9 ← N3	N11 ← N5
N7 ← $\overline{\text{CLR}}$	N3 ← INC ⊕ N11
N4 ← N8 & N7	N1 ← INC & N11
N5 ← N9 & N7	N2 ← N1 ⊕ N10

Fig. 2. *Cycle-based simulation compilation example.*

The compiler uses this information to determine which gates need to be evaluated during each phase. This is done in two steps. First, for each phase, the compiler propagates the clock values into the circuit. This uses simple rules of Boolean logic, such as the fact that the output of an AND gate with a zero input must be zero. The goal is to identify latches with a zero control, which are therefore provably opaque during that phase. Next, again for each phase, the compiler finds all gates that can be reached from a clock or other input through a path that does not contain an opaque latch.

Next, a sequence of Boolean operations is emitted corresponding to the gates in each phase. Because we used PA-RISC machines for simulation, the sequences were actually output in PA-RISC assembly language. The sequences totaled more than two million instructions for the PA 8000 design. The gates are ordered in sequence so that a gate is not emitted until its inputs have been computed. Cycles, or loops, in the circuit are handled by looping through the gates in the cycle until all circuit nodes are stable.

Numerous optimizations are done on the output assembly language sequences:

- Clock signals have known values during each phase, which can be propagated into the circuit. These constant values can simplify or eliminate some of the Boolean operations.
- The 32 PA-RISC registers are used to minimize loads and stores to memory. Boolean operation scheduling and victim register selection are employed to minimize the number of loads and stores.
- The compiler can determine which circuit nodes carry information from one phase to the next. The remaining nodes are temporaries whose values need not be flushed to memory after their final use within a phase.
- To eliminate NOT operations corresponding to inverting gates, the compiler can represent nodes in inverted form and perform Demorgan transformations of Boolean operations (e.g., NOT-AND is equivalent to OR-NOT).
- Aliasing of circuit nodes is done to eliminate code for simple buffers and inverters.

Any one of the Boolean operations in the output assembly language sequence operates on all 32 bits of the PA-RISC data path, as shown in Fig. 3. We make use of this parallelism to run 32 independent test cases in parallel. This is possible because the simulator always executes exactly the same sequence of assembly language instructions regardless of the test case (assuming the circuit being simulated is the same). This does not reduce the time to solution for a given test case, but it does increase the effective throughput of the simulator by 32 times. This was still very useful because our verification test suites are divided into a vast number of fairly short test cases.

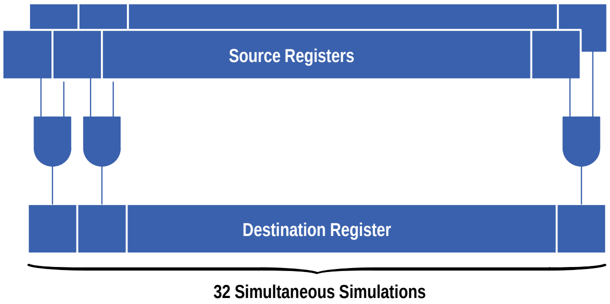


Fig. 3. *Multislot cycle-based simulation.*

The compiler allows the user to write C++ behavioral descriptions of blocks such as memories and register files that are not efficient to represent using gate-level primitives. The compiler automatically schedules the calls to this C++ code, and an API (application programming interface) gives the code access to the block's ports.

Pseudorandom Testing

We had learned from previous projects that the type of defects likely to escape the RTL simulation effort would involve subtle interactions among pending instructions and external bus events. With up to 56 instructions pending inside the processor and a highly concurrent system bus with multiprocessing support, it is not possible to count—much less fully test—all of the interactions that might occur. We believed that the value of handwritten test cases and test cases randomly expanded from templates was reaching diminishing returns, even with the low simulation throughput achievable with the RTL simulator.

We had also learned that pseudorandom code generators were a very effective means of finding these kinds of defects. Such a program generates a pseudorandom sequence of instructions that use pseudorandom memory addresses and pseudorandom data patterns. However, it is important that the program make pseudorandom selections in a manner that considers the microarchitecture of the processor and the kinds of interaction defects that are likely to occur.

Selecting memory addresses is a good example. Memory addresses are 64 bits wide. If they were selected truly randomly, reusing the same address within a test case would be an impossibly rare event. This would fail to stress important aspects of the machine, such as the logic that detects that a load is dependent on a preceding store with the same address. There are hundreds of selections that a generator makes in which the microarchitecture must be carefully considered.

We chose to target the cycle-based simulation environment for a new pseudorandom code generator. Our pseudorandom code generator was carefully tuned for the microarchitecture of the PA 8000 and included support for the new PA-RISC 2.0 instruction set. Hundreds of event probabilities could be specified by a control file to provide engineering control over the types of cases being generated.

We also chose not to port the rich set of checking software from the RTL simulation environment to the cycle-based simulation environment because of the effort involved and risk that performance would be reduced. Generators such as our pseudorandom code generator predict the final register and memory state of the processor, and defects will generally manifest themselves as mismatches between the simulated and predicted final state. It is possible that an error in state will be overwritten before the end of a test case, but a defect won't be missed unless this happens in every test case that hits it, which is extremely unlikely statistically. Our experience with hardware prototype testing, in which internal signals are unavailable and all checking must be done through final state, also made us confident in this strategy.

Cycle-Based Simulation Environment

Fig. 4 shows the cycle-based simulation environment, which will be described by following the life cycle of a typical test case. The job controller controls the 32 independent simulations that are running in the data path positions, or slots, of the cycle-based simulation model. It starts and ends test cases in the 32 slots independently. It is controlled by a UNIX® shell, which is driven either by a script or interactively for debug activities.

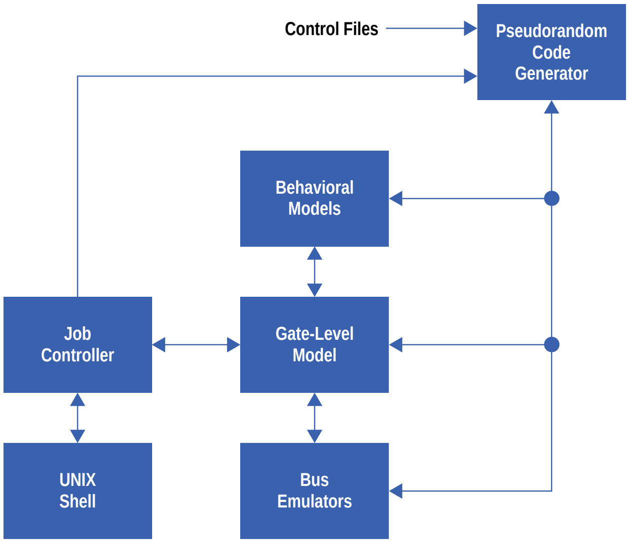


Fig. 4. *Block diagram of the cycle-based simulation environment.*

When a slot becomes available, the controller commands the pseudorandom code generator to generate a new test case, occasionally first reading a new control file. The test case is specified by the initial state of memory and the processor's registers, and the pseudorandom code generator specifies the initial state of the caches as well to prevent an initial flurry of misses.

The pseudorandom code generator downloads the initial state of the simulation into various components of the simulated model. These include the gate-level model, behavioral models representing caches, register files, and other regular structures, and emulators representing bus devices such as the memory system, I/O adapter, and third-party processors. The model is then stepped for numerous clock cycles until a breakpoint trigger fires to indicate the end of the test case. The pseudorandom code generator is then commanded to extract the relevant final state from the simulated model and compare it with the final state that it predicted to determine whether the test case passed.

We used a simulation farm of up to 100 desktop workstations and servers for cycle-based simulation. Jobs were dispatched to these machines under the control of HP Task Broker.[1] Each job ran several thousand test cases that were generated using a specific pseudorandom code generator control file.

Multiprocessor Testing

Multiprocessor testing was a key focus area. We wrote emulators for additional processors and the I/O adapter, which share the memory bus. It was only necessary to emulate the functionality required to initiate and respond to bus transactions, but the emulators were accurate enough that defects in the processor related to cache coherency would manifest themselves as mismatches in the final memory state.

We established linkages with our pseudorandom code generator so that the emulators would be more effective. When a test case started, the pseudorandom code generator downloaded control file information so that parameters such as transaction density and reply times could be easily varied. The pseudorandom code generator also downloaded the memory addresses used by the test case so that the emulators could initiate transactions that were likely to cause interactions.

Coverage Improvement

Improving the test coverage of our pseudorandom code generator was an ongoing activity. The pseudorandom code generator has hundreds of adjustable values, or *knobs*, in its control file, which can be varied to focus the generated test cases. We found that the defect rate quickly fell off when all knobs were left at their default settings.

We used two tactics to create more effective control files. First, we handcrafted files to stress particular functional areas. Second, we generated files using pseudorandom techniques from templates, each template specifying a particular random distribution for each knob. We found with both strategies that it was important to monitor the quality of the files generated.

We did this in two ways. First, our pseudorandom code generator itself reported statistics on the test cases generated with a given control file. A good example is the frequency of traps. Traps cause a large-scale reset inside the processor, including flushing the instruction queues, so having too many traps in a case effectively shortens it and reduces its value. We made use of instrumentation like this to steer the generation of control files.

Feedback is often needed based on events occurring within the processor, which our pseudorandom code generator cannot generally predict. For example, an engineer might need to know how often the maximum number of cache misses are pending to be confident that a certain area of logic has been well-tested. Test case coverage analysis was accomplished by an add-on tool in the simulation environment. This tool included a basic language that allowed engineers to describe events of interest using Boolean equations and timing delays. The list of events could include those that were expected to occur regularly or even those that a designer never expected to occur. Both ends of this spectrum could provide useful information.

Once the events were defined, the add-on tool provided monitoring capabilities during the simulation. As test cases were run, the tool would generate output every time it detected a defined event. This output was then postprocessed and assembled into an event database. The event database could contain results of thousands of test case runs. Event activity reports were then generated from this event database. These reports included statistics such as frequency of events, duration of events, the average, maximum, and minimum distance between two occurrences of a event, and so on.

The event activity reports were then analyzed by engineers to identify weak spots in coverage and provide feedback to the generation of control files. This methodology provided one other benefit as well. For many functional defects, especially ones that were hard to hit, the conditions required to manifest the defect were coded and defined as an event. Then this add-on tool was used with a model that contained a fix for the defect to prove that the conditions required for the defect were being generated.

Switch-Level Simulation

In typical ASIC design methodologies, an RTL description is the source code for the design, and tools are used to synthesize transistor-level schematics and IC layout mechanically from the RTL. Verifying the equivalence of the synthesized design and the RTL is largely a formality to guard against occasional tool defects. In the full-custom methodology used on the PA 8000, however, designers handcraft transistor-level schematics to optimize clock rate, die area, and power dissipation. Therefore, we needed a methodology to prove equivalence of the handcrafted schematics and the RTL.

At the time the project was undertaken, formal verification tools to prove this equivalence were not available. Instead, we turned to an internally developed switch-level simulator. Although much slower than the RTL simulator, the switch-level simulator included essential features such as the ability to model bidirectional transistors, variable drive strength, and variable charge ratios. Thanks to this careful effort in switch-level verification on the PA 8000, not a single defect was found on silicon that was related to a difference between the transistor-level schematics and the RTL.

Verification was performed by proving that a block behaved the same when running a test case in the RTL simulator and in the switch-level simulator. First, a full-chip RTL simulation of a test case was done with the ports of a block monitored. These vectors were then turned into stimulus and assertions for a switch-level simulation of the block. Initializing the state of the block identically in the two environments was a challenge, especially since the hierarchies and signal names of the RTL and schematic representations can differ.

Initially, this strategy was used to turn on the switch-level simulator models of individual blocks on the chip. This helped to distribute the debug effort and quickly bring all blocks up to a reasonable quality level. Afterward, the focus shifted to full-chip switch-level simulator verification. In addition to collecting vectors at the ports of the chip, thousands of internal signals were monitored in the RTL simulation and transformed into assertions for the switch-level simulation. These were valuable for debugging and raising our confidence that there were no subtle behavioral differences between the two models.

The RTL simulation effort was a plentiful source of test cases, but they were targeted at functional defects rather than implementation errors, and the slower speed of the switch-level simulator allowed only a portion of them to be run. To improve coverage, the process shown in Fig. 5 was used at the block level. The RTL description for the block was converted into an equivalent gate-level model using tools developed for cycle-based simulation. Automated test generation tools, normally used later in the project for manufacturing, were then used to create test vectors for the gate-level model. If the switch-level simulation using these vectors failed, then the two representations were known to differ. While the automated test generation tools do not generate perfect test vectors, this process still proved to be a valuable source of additional coverage.

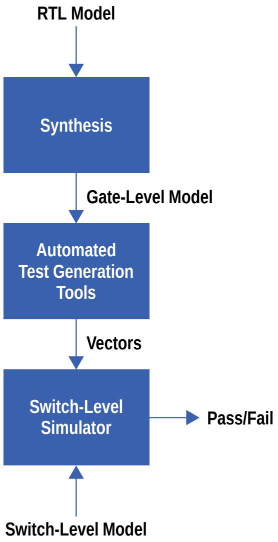


Fig. 5. *Process used at the block level to improve test coverage.*

The switch-level simulator also supports several different kinds of quality checks. These include dynamic decay checking to detect undriven nodes, drive fight checking to detect when multiple gates are driving the same node, and a race checking methodology. This was implemented by altering how the clock generator circuits were modeled to create overlap between the different clocks on the chip. Failures that arose from overlapped clocks pointed to paths requiring detailed SPICE simulations to prove that the race could not occur in the real circuits. Reset simulations were done from random initial states to ensure that the chip would power up properly. Finally, a switch-level simulator model was built from artwork netlists to prove that there were no mismatches between the artwork and the schematics that were missed by other tools.

Postsilicon Verification

Presilicon verification techniques are adequate to find nearly all of the defects in a design and bring the level of quality up to the point where the first prototypes are useful to software partners. However, defects can be found in postsilicon verification that eluded presilicon verification for many reasons.

First, test code can be run at hardware speeds, several orders of magnitude faster than even the fastest simulators. Errors in the simulation models or limitations in the simulators themselves can cause the behavior of the silicon to differ from that predicted by the simulators. Finally, most simulation is targeted at system components, such as the PA 8000 itself, rather than the entire system. Errors in the specifications for interfaces between components, or different interpretations of these specifications, can become apparent when all components are integrated and the system exercised.

Overlapped Test Coverage

Previous projects had established the value of running test code from as many sources as possible. Each test effort had its own focus and unique value, but each also had its own blind spots. This is even true for pseudorandom code generators. In the design of these very complex programs, many decisions are made that affect the character and style of the generated code. There can be code defects as well that cause coverage holes. The large overlap in coverage between efforts proved to be an invaluable safety net against the limitations and blind spots of individual tools.

The PA 8000 verification team focused its effort on pseudorandom code testing. Experience showed that this would be the primary source of subtle defects and would allow us to find most defects before our software partners. We ran several tools including our pseudorandom code generator and generators used in the development of the PA 7200 and PA 7300LC processors and the HP 9000 Model 725 workstation. Several tools were capable of generating true multiprocessing test cases that included data sharing between random sequences running on different processors. Data sharing with DMA processes was implemented as well.

We developed a common test environment for most of these random code generators. The HP-UX* operating system is not a suitable environment because its protection checks do not permit many of the processor resources to be easily manipulated. Our test environment allowed random testing of privileged operations and also included many features to improve repeatability and facilitate debugging. For example, it performed careful initialization before each test case so that, aided by logic analyzer traces, we could move a failing test case to the RTL simulator for easy debugging (in hardware, there is no access to internal signals). We established a plug-and-play API so that the investment in the environment could be leveraged across several generators.

In parallel with our pseudorandom testing, our software partners pursued their own testing efforts. While primarily targeted at their own software, this provided stress for the processor as well. The test efforts included the HP-UX and MPE/XL operating system kernels, I/O and network drivers, commands, libraries, and compilers. Performance testing also provided coverage of benchmarks and key applications. Finally, although it did not find any defects in the PA 8000, HP's Early Access Program made available preproduction units to customers and external application developers.

Ongoing Improvement

When defects were found, we used the process shown in Fig. 6 to learn as much as possible about why the defect was missed previously and how coverage could be improved to find additional related defects. After the root cause of the defect was determined, actions were taken in the areas of design, presilicon verification, and postsilicon verification.

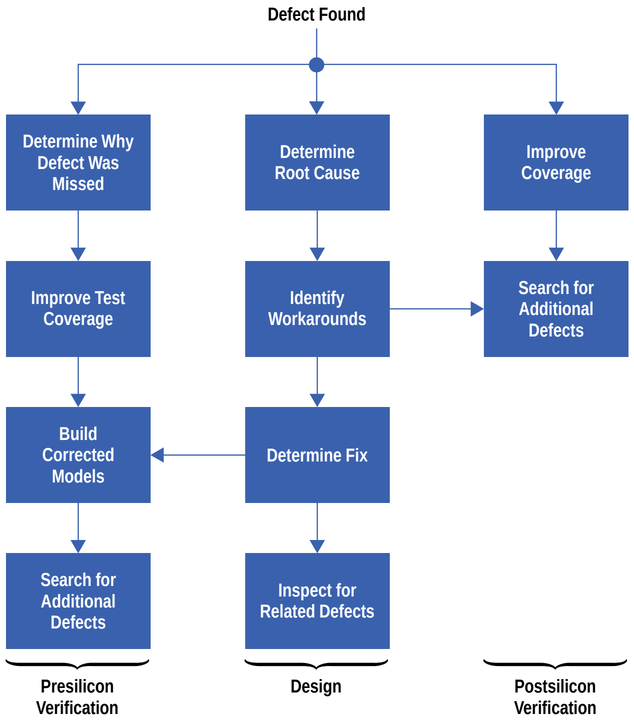


Fig. 6. *Postsilicon quality improvement process.*

The designers would identify workarounds for the defect and communicate these to our software partners, at the same time seeking their input to evaluate the urgency for a tape release to fix the defect. The design fix was also determined, and inspections were done to validate the fix and brainstorm for similar defects.

In the area of presilicon verification, reasons why the defect was missed would be assessed. This usually turned out to be a test case coverage problem or a blind spot in the checking software. Models would then be built with the design fix and other corrections. Test coverage would be enhanced in the area of the defect, and simulations were done to prove the fix and search for any related defects. Cycle-based simulation played a big role here by finding several introduced defects and incomplete fixes.

The postsilicon verification activities were similar. Coverage for the tool that found the defect would be enhanced, either by focusing it with control files or by tool improvements. Spurred by a healthy rivalry, engineers who owned other tools would frequently improve them to show that they could hit the defect as well. All of this contributed to finding related defects.

Performance Verification

At the time the PA 8000 was introduced in products, it was the world's fastest available microprocessor. Careful microarchitectural optimization and verification of the design against performance specifications were factors in achieving this leadership performance.

In a microarchitectural design as complex as the PA 8000, seemingly obscure definition decisions and deviations of the design from the specification can cause a significant loss of performance when system-level effects and a variety of workloads are considered. A good example is a design defect that was found and corrected in the PA 8000. When a cache miss occurred under certain circumstances, a dirty cache line being evicted from the cache would be written out on the system bus before the read request for the missing line was issued. Since the addresses of the two lines have similar low-order bits, both mapped to the same bank of main memory. The memory controller would begin processing the write as soon as it was visible on the bus, busying the memory bank and delaying the processing of the more critical read.

A detailed microarchitectural performance simulator was written early in the project to help guard against such issues. It was used to project performance and generate a statistical profile for a variety of benchmarks and applications. Workloads with surprising results or anomalous statistics were targeted for more detailed analysis, and through this process opportunities were identified to improve the microarchitecture. Particularly valuable feedback came from the compiler development team, who used the simulator to evaluate the performance of compiler prototypes. The concurrent development of tuned compilers with close cooperation between hardware and software teams was a key contributor to the PA 8000's performance leadership.

The microarchitectural performance simulator was written at a somewhat abstract level, so it could not provide feedback on whether the detailed design met the performance specifications. Comparing the performance of the RTL simulator against the microarchitectural simulator was the obvious way to address this, but the RTL simulator was far too slow. As a compromise, we performed this comparison on key performance kernels that were tractable enough for the RTL simulator. We also developed a path by which a workload could be run up to a critical point using the microarchitectural simulator, at which point the state of the memory, caches, and processor registers could be transferred into the RTL simulator for detailed simulation.

Performance verification continued in the postsilicon phase of the project. The PA 8000 incorporated several performance counters that could be configured to count numerous events. These were used to help identify workloads or segments of workloads needing closer analysis. The PA 8000's external pins and debug port provided sufficient information to determine when instructions were fetched, issued for execution, and retired. Isolation of specific performance issues was aided by a software tool called the depiper which presented a visual picture of instruction execution. Through these efforts, several performance-related hardware defects were identified and corrected before production.

Results

Achieving a defect-free design at first tape release is not a realistic expectation for a design as complex as the PA 8000. Nevertheless, we were extremely satisfied with the quality we achieved at first tape release. The first prototypes were capable of booting the operating system and running virtually any application correctly. In fact, only one defect was ever hit by an application, although a few defects were encountered in stress testing of system software.

Fig. 7 shows the sources of defects found and corrected after first tape release. Surprisingly, about a third of the total defects were found by continued use of the presilicon verification tools (mostly the cycle-based simulation environment) for a few months following tape release. This indicates that despite the outstanding performance of cycle-based simulation, the project would have benefited from even more throughput, or perhaps use of the tool earlier. A third of the defects were also found by one of the pseudorandom code generators running on hardware prototypes. Inspections were a significant source of defects. The remaining defects were split between turn-on work, performance analysis work, and partner software testing. Since very few defects were discovered by partners, we could generally communicate workarounds ahead of time and take other steps to minimize the impact.

Fig. 8 shows the impact of the defects found after first tape release on our software partners. A large majority were never seen outside the environment in which they were found and had no significant impact. About half of these involved functional areas, such as debugging hardware, that are not even visible to applications or system software. Most of the remaining defects had only a moderate impact. Examples are defects that were found by a partner at the expense of their testing resources, defects that required a workaround in system software, and defects that required certain performance-related features in the processor to be disabled. Only a handful of defects were severe enough to temporarily block or significantly disrupt a partner's development and testing efforts. All but one of these were early multiprocessing defects that slightly delayed bringing up the multiprocessing operating system.

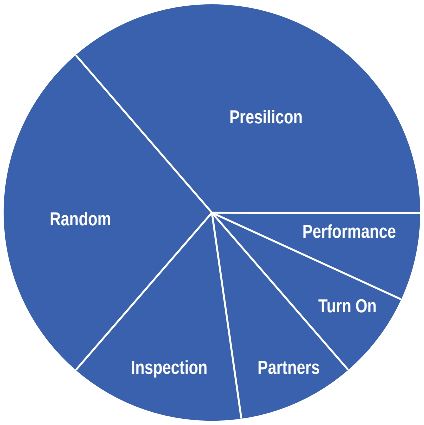


Fig. 7. *Sources of defects found and corrected after first tape release.*

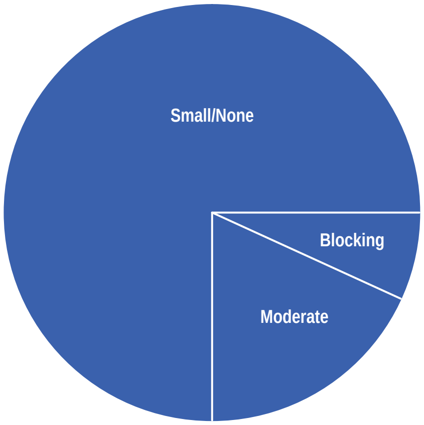


Fig. 8. *Partner impact.*

Cycle-Based Simulation Results

The cycle-based simulation effort made an essential contribution to the verification of the PA 8000. Fig. 9 shows the sources of defects that eluded our RTL simulation effort (which incorporated existing best practices). If we had not made the investment in cycle-based simulation, the number of defects that would have had to be found by postsilicon techniques would have been three times higher. It was much less expensive to fix the defects caught by cycle-based simulation as the design progressed than it would have been to fix them in later revisions.

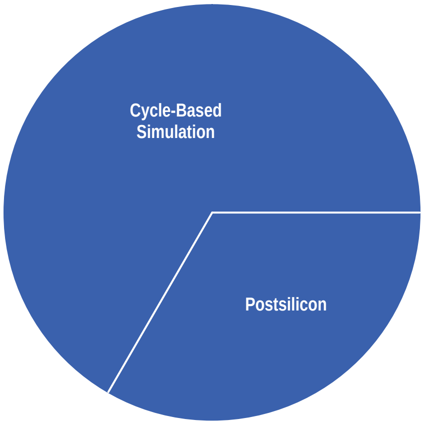


Fig. 9. *Defects escaping traditional presilicon verification.*

Also, because cycle-based simulation tended to find the most severe defects early, no masking defects were present, and the number of serious blocking defects that we had to manage after the first tape release was reduced by three to six times. If our software partners had been exposed to this level of severe defects, it is probable that the product's time to market would have been impacted.

Finally, cycle-based simulation provided a high-confidence regression test before each tape release. Several incomplete bug fixes and new defects that had been introduced in the design were found in time to be corrected before a tape release.

Conclusions

Continuous innovation in functional verification tools and processes is required to keep pace with the increasing microarchitectural complexity of today's CPUs. This paper has described the methodologies used to verify the PA 8000. These met our most important goal of improving the quality of the PA 8000 to the high level demanded by our customers. By finding defects early, they also helped us conserve our engineering resources and quickly deliver the industry-leading performance of the PA 8000.

Acknowledgments

The authors would like to thank all of the different teams who contributed to the successful functional verification of the PA 8000. Special thanks to all of the individuals in the Engineering Systems Laboratory in Ft. Collins, Colorado and the General Systems Laboratory in Roseville, California who were involved in the design and verification effort. Many thanks also to our partners: the Cupertino Open Systems Laboratory for testing efforts and flexibility in working around defects, the Systems Performance Laboratory and Cupertino Language Laboratory for performance verification work, and the Integrated Circuit Business Division for supporting our simulation tools.

Reference

1. T.P. Graf, et al, "HP Task Broker: A Tool for Distributing Computational Tasks," *Hewlett-Packard Journal*, Vol. 44, no. 4, August 1993, pp. 15-22.

Electrical Verification of the HP PA 8000 Processor

Electrical verification applies techniques from both functional verification and reliability and environmental testing to improve the quality of the CPU. Electrical verification checks that the CPU functions correctly under stressful environmental conditions, well outside the normal operating environment.

by John W. Bockhaus, Rohit Bhatia, C. Michael Ramsey, Joseph R. Butler, and David J. Ljung

Early in a product's design life cycle, considerable attention is paid to its functional correctness. This functional verification is carried out on the earliest prototypes and, especially in the case of complex devices such as large VLSI circuits, even earlier through simulation.

As a product approaches customer shipments, testing it against HP's stringent reliability and environmental specifications is a critical task.

In between these two test methodologies, there is a third method that is becoming increasingly important. This is *postsilicon electrical verification*. Electrical verification applies techniques from both functional verification and reliability and environmental testing to improve the quality of a device or product.

The philosophy of electrical verification is different from the other two methods. While it is possible, although not necessarily common, to complete functional verification and reliability testing without finding reasons to change a design, electrical verification's goal is to find a design's weaknesses and fix them. Even a very good design should come out of electrical verification with a higher level of quality.

Like functional verification, electrical verification seeks to exercise as many of a design's logic states, signal paths, and state transitions as possible. However, entering a state, driving a signal, or triggering a transition once is not enough for electrical verification. Combining reliability testing with the coverage of functional verification, electrical verification repeats the functional tests under stressful conditions beyond what a product may ever see in a real application.

Electrical verification goes beyond the limits of reliability and environmental testing, which is typically done only at the system level. Reliability tests are usually done independently of each other. For example, line voltage variations are not applied concurrently with ambient temperature tests. Reliability tests stop at predefined limits. In contrast, electrical verification varies many test parameters at the same time. The ranges of those parameters are continually increased until failures are found.

Electrical verification is not simply a random scattering of tests executed in the hopeful search for some kind of statistical confidence. Instead, the goal is complete coverage of the product's operating space and beyond. This serves two purposes. First, the design is verified over all of the combinations of actual conditions it may encounter. Secondly, failure mechanisms or critical features that may lie outside normal operating limits can be found, identified, and possibly fixed. These items, such as critical timing paths, charge sharing conditions, or marginal driver strengths, can move inside normal limits as a product ages, manufacturing conditions change, or other unanticipated situations arise. Removing them early in the design life cycle ensures a reliable product with a longer life for the customer and avoids a costly in-production change for HP. Furthermore, fixing these electrical failures can increase the yield of the device at a given frequency and can enable higher-frequency and higher-performance upgrades.

An additional purpose of electrical verification is to help determine production test limits and guardbands. By including IC process parameters in the verification effort, test limits can be extrapolated to predict proper function through normal operating conditions.

Fig. 1 is a flowchart of the electrical verification process.

Shmoo Plots

The hunt for electrical bugs begins with the *shmoo plot*. The shmoo was a character in the Lil' Abner cartoon strip that had a changing, bloblike shape. The shmoo plot shows whether the device under test passed or failed as a function of various combinations of electrical parameters applied to it. The name has become part of the engineer's vernacular because the region where a particular device passes or fails, plotted against the parameters applied, may have some of the rounded

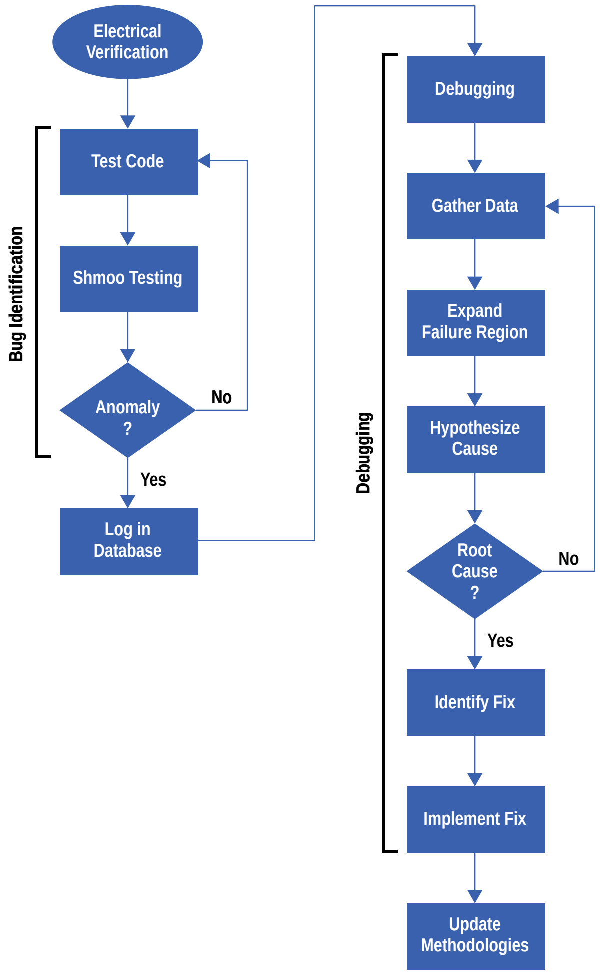


Fig. 1. *The electrical verification process.*

curves and shifting nature of the mythical shmoo. Often, the shape of the plot conveys information about the failures. Many common shapes have been given names (see ***Subarticle 4a***).

The shmoo's name has also become a verb. To shmoo is to run a test repeatedly while varying one or more parameters. These parameters are the axes of the shmoo plot and include both the obvious and the nonobvious.

Obvious shmoo plot parameters include power supply voltage and temperature. In complex systems such as those using the PA 8000, many different supply voltages exist. Supply voltages and temperature are clear choices for shmoo plotting because they so directly affect the operation of electronic devices.

The IC manufacturing process is a key shmoo plot parameter for projects like the PA 8000. In keeping with the goal of methodically covering the shmoo space, testing a large, random sample of parts isn't enough. Instead, prototype parts are manufactured with one or more process metrics intentionally modified. Typical IC shmoo plot parameters are transistor gate length and leakage currents.

Clock frequencies are also good shmoo plot parameters. Increasing the frequency of clocks is a good way to find slow signal paths. However, pushing frequencies higher only tells some of the device's story. Useful information can also be found by seeing how slowly it can go and by testing many frequencies in between the maximum and minimum. Logic races and transmission line reflections are just two potential problems that may lurk in the low frequencies, which the engineer often assumes are "easy."

A shmoo plot parameter that may be less obvious is the software executed on the device under test. Good shmoo plot code seeks to exercise as much of the device as possible. Executing a power-up self-test or booting an operating system may seem complicated, but they are not necessarily good shmoo tests. The PA 8000 shmoo process used a large number of tests that ranged from specially designed to randomly generated.

Test Cases

The success of any verification effort depends to a large extent on the nature and type of test cases that are run on the CPU. The testing code needs to be good enough so that when the systems reach customers the CPU is bug-free. To ensure this, the tests must provide adequate coverage of the design features incorporated into the CPU. Furthermore, because of the complexity of today's processors, it is impossible to imagine all of the interactions that must occur to cause a particular event in the processor. This complexity necessitates the use of random testing. In the postsilicon environment, random testing is aided by the fact that throughput is generally not a problem (compared with the presilicon simulations done on software models, which are millions of times slower than running on the actual hardware) and therefore a huge volume of random testing can be accomplished.

The electrical verification of the HP PA 8000 CPU relied upon the following sources for test cases:

- Directed handwritten tests
- Focused random tests targeted at specific CPU functions
- Random code generators
- Library of worst-case tests for previous bugs
- HP-UX* application code.

In most instances, these test cases were checking for failures in real time. In general, they would set up some initial conditions, run the test code, and perform some checking. Some cases checked for a specific outcome in memory or a general register. Others compared the full architectural state at the end of the test case to some expected final state.

For most of the sources mentioned above, the test cases were leveraged from the presilicon verification effort through the use of various scripts to perform modifications for the postsilicon operating environment. Several benefits can be realized by leveraging the work from presilicon verification. First, leveraging all the tools results in less development time and hence less total work for the postsilicon verification team. Second, by sharing the tests and tools, we get a common environment between presilicon and postsilicon verification. This allows easier modeling of postsilicon failures in presilicon simulation tools and makes the learning curve easier as well. It also provides a path to go back and forth between the two environments, which allows directed test development for postsilicon failures.

Since presilicon verification is targeted at functional correctness, the use of tests leveraged from that effort requires some caution. Electrical verification tests in general need to be much more data-pattern-sensitive than their functional counterparts. For example, a bus may need to be driven from all zeros to all ones or alternating ones and zeros to adequately test for electrical failures, whereas the data patterns in a functional test don't usually influence the logical correctness of the design. Therefore, the random code generators used in the electrical verification effort were modified to provide knobs that allowed the test case writers to control the data patterns used in these tests.

Our library of worst-case test code for all bugs known to date was valuable in ensuring that no known failure mechanisms had gone uncovered as we went through different revisions of the chip. This library of tests was always kept updated and used repeatedly on all CPU parts.

One final source of test cases was our HP-UX stress test suite. Not only is this most similar to what the majority of customers are going to run, but HP-UX testing can also offer the most coverage. Usually this is used as the last set of testing to make sure that all of our test coverage so far has been close enough to the stress that HP-UX code puts the PA 8000 through. The stress test suite consists of a number of applications including the SPEC benchmarks and scripts that make sure that everything is running correctly. The drawbacks are that the run time for HP-UX stress tests is a few hours, which is orders of magnitude longer than the other test types, and that these tests are much harder to debug if failures occur.

Automated Tools

To save time, we created a number of tools that help us automate many of the verification tasks. Each of our test systems was connected to a controller system (see Fig. 2). We then replaced the boot ROMs on the test system with ROM emulators. Our controller system controlled the voltages, frequencies, and temperature and the code that was run on the test system, and it also monitored all of the I/O to and from the test system through the RS-232 port.

We needed to have complete control over the code that was run on the test system. Instead of using the boot ROM to initialize the system so that we could run a full operating system, we used our own framework code called the *CharROM*, short for *characterization ROM* (see Fig. 3). The CharROM installed a subset of the normal initialization code and then ran test code for us. All of our tests were compiled into a separate ROM called a *testROM* which could be uploaded without changing the CharROM. Each testROM contained the test code as well as information on how to run each test, such as verbosity settings and number of iterations. It also contained information about the test that was printed out so that our tools could save information on what tests we were running. With our systems set up this way, we could turn on power and within a few seconds the test system would be printing out initialization information and then test codes.

The CharROM had the flexibility to run tests in batch mode, one at a time according to the testROM, or interactively for debugging. In interactive mode the CharROM let us run tests and modify and view the state of the chip.

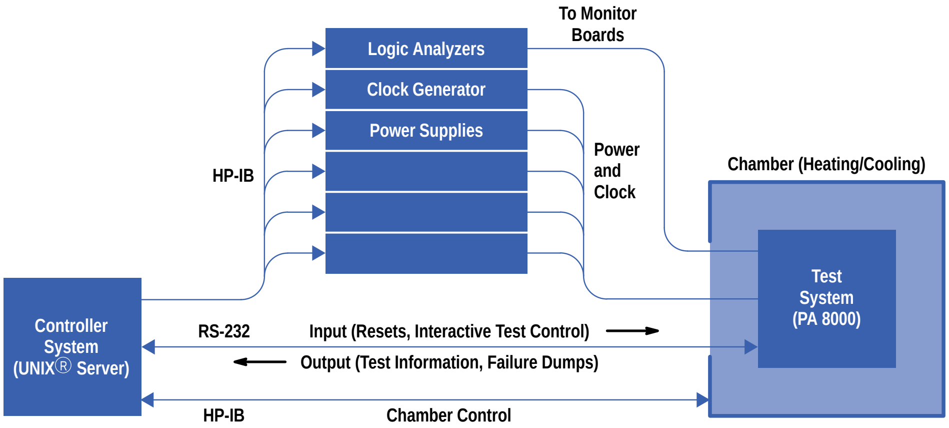

To Monitor Boards
Logic Analyzers
Clock Generator
Power Supplies
HP-IB
Power and Clock
Chamber (Heating/Cooling)
Test System (PA 8000)
Controller System (UNIX® Server)
RS-232
Input (Resets, Interactive Test Control)
Output (Test Information, Failure Dumps)
HP-IB
Chamber Control

Fig. 2. *Electrical verification test system setup.*

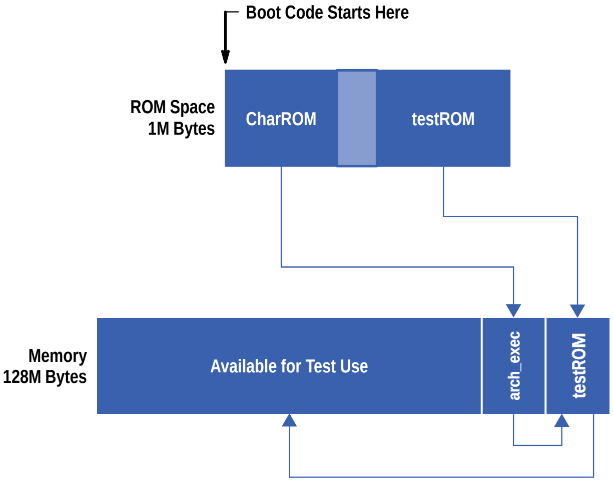


Fig. 3. *The CharROM makes it possible to boot quickly and change tests rapidly. Boot starts at the beginning of the CharROM.* arch_exec *and the testROM are copied to memory for speed reasons. Basic assembly tests are run either in ROM or in memory. Modified phase 1 format tests are unpacked into available memory by* arch_exec*, which runs and checks them.*

The big advantage of the CharROM was that it booted quickly and let us change tests rapidly. This saved a great deal of time during debugging when we needed to run many code experiments. It also eliminated a lot of the boot code that would be needed to run something like the HP-UX operating system. This meant that we were less likely to hit a failure while booting and if we did we could often make quick changes to the CharROM to avoid the failure. This was most obvious when we discovered an electrical bug in a branch instruction that was keeping us from booting. We were able to rewrite the CharROM framework in two days without using the failing types of branches, something that we could never have done with a full-fledged boot ROM.

The most important feature of the CharROM was the control it gave us over exactly what code was being run on the system. Running tests in an environment like HP-UX leaves the tester at the mercy of the operating system, which can switch processes in and out and limit access to privileged operations.

One of the important responsibilities of the CharROM was to initialize the chip state between tests. This helped control repeatability for a given test (so we weren't dependent on a random chip state) and also helped avoid dependencies between tests, that is, a previous test affecting the run of a future test.

Inside the CharROM we had a subframework called arch_exec that could run our modified presilicon test cases. arch_exec takes apart the initial state and sets up the chip accordingly. After the test is run, arch_exec compares the chip state to the expected final state information in the test, automatically showing us any failures. This let us deal with many tests in bulk.

To run our shmoo tests we had scripts that would boot the systems at each point in the shmoo test domain and read the output to decide if the tests had passed or failed. After running a shmoo test or even during a shmoo test we could analyze the output to ignore certain failures or focus on specific failures.

At the completion of a shmoo test, the shmoo script stored all the output in our shmoo database. We used the database to look for specific tests or specific parts so we could avoid duplicating work. This also turned out to be very useful when we needed to take another look at past bugs; we still had all of the shmoo information for the bug work.

Failure Identification

The process of electrical verification of a CPU begins with the task of identifying electrical failures. An electrical failure can be described as the malfunction of a chip under certain but not all operating environments. If the failure occurs regardless of the operating environment, then it is termed a functional failure and is not covered here. Typically, electrical failures can be traced to some electrical phenomenon that occurs only under certain operating environments. Some examples include latch setup or hold time violations, noise issues, charge sharing, leakage issues, and cross talk.

The task of identifying new failure mechanisms involves a number of steps. First, test code must be selected and run in a variety of operating environments. The data collected from this is displayed graphically in shmoo plots and anomalies are noted. Next, the anomalies are checked for repeatability. If the anomaly repeats reliably, the failure signature is analyzed in an attempt to classify the failure or narrow it down to a particular area of the CPU, if possible. The sensitivities to different operating environment variables are also determined to gain further understanding of the failure mode before it is moved into the debugging stage. Each of these steps will be discussed in more detail.

Searching for an Anomaly

Electrical verification of the PA 8000 CPU encompasses a huge test space that is impossible to cover in a reasonable amount of time. This is partly because of the increasing complexity of devices and partly because of the large number of operating environment variables. Operating variables include test code, ambient temperature, several supply voltages, frequency and bus ratios, types of CPU chips (i.e., variations in the fabrication process), different speed grades of cache SRAMs, and many others. A number of techniques were applied in the electrical verification of the PA 8000 CPU that, taken together, effectively covered this large test space.

Initially, the emphasis is placed on varying a large number of the operating variables and exercising the CPU with simple test code. A variety of CPU parts from different corners of the fabrication process are deliberately selected and run under various combinations of temperature and supply voltages to look for failures. For example, known fast PA 8000 CPUs were run in a cold chamber at high supply voltages to look for one class of failures. Similarly, a set of slow CPUs were run in a hot chamber at low supply voltages to look for another class of failures. Experience is always a good guide to the operating variable combinations that are likely to yield failures.

Stress testing is another technique that is applied to induce failures. Stress testing refers to running the CPU with test code on the fringes of the operating environments, under conditions to which an actual system in the field may never be subjected. However, a failure induced in this fashion can often be moved into an operating region that we care about, simply by further experimentation and analysis.

As the process of electrical verification proceeds, the emphasis shifts from running simple test code at a variety of operating points to more complex code sequences at fewer operating points. This can be compared to exploring the test space from a *breadth-first* search to a *depth-first* search. The more complex code sequences are derived from running several random code sequence generators, pseudorandom focused tests, directed tests, and HP-UX application code.

Using one or more of the techniques outlined above, test data is gathered and can be viewed in shmoo plots. These plots are examined and compared to what has been observed in the past for previous test runs on earlier silicon revisions. If the shmoo plots reveal regions of failure not observed before, then we have a *shmoo anomaly*, which needs to be pursued further.

Verifying Repeatability

Once a shmoo anomaly is identified, a number of steps need to be taken to validate and confirm it. It is important that the anomaly be reliably repeatable and be traceable to a CPU malfunction. The steps outlined below are used to satisfy the repeatability requirement.

1. The failing code sequence is rerun several times on the same CPU to confirm failure. This is done to rule out the possibility that an inadvertent change in the operating environment may have induced the failure.
2. In a system verification environment, several other components must be removed from suspicion before the anomalous behavior can be attributed to the CPU. The failing CPU can be placed in a completely different system and the failing code sequence rerun under similar operating conditions to repeat the failure. The failing CPU can also be used in several different processor boards to rule out any dependencies on the processor board characteristics.

3. The next step is to try to locate the failure mode on different but similarly fabricated CPU parts. These could be parts from the same wafer or with similar process characteristics. If the failure mode is not observed on any other CPU, then it is generally considered to be a test escape from the wafer and package screens, meaning there is a defect on this chip that the wafer and package screens did not find. In other words, we have not found an inherent problem with any circuits on the chip. In that case, we will investigate whether we have a coverage hole in our wafer or package screens, rather than move this failure into the debugging phase.

4. If the above three steps are satisfied, then the failure mode is checked for sensitivities to different operating environment variables. Most electrical failures are modulated by one or more of the operating variables, whether it be temperature, supply voltage, or delays on key system clocks. This can not only expand the failure region, but also provide some clues to the type of failure, which can be extremely useful information for the task of debugging.

5. Throughout the process of verifying the repeatability of the failure mode, it is also important to watch the failure signature and check it for consistency. That is, one must ensure that each rerun of the test code is producing the same failure mode.

Classifying the Failure Mode

After a shmoo anomaly is identified and has passed the repeatability requirement, it is time to classify and list the characteristics of the failure mode to see if it is unique or is one that has been observed before. Either classification is important. If it is new, then it needs to be debugged fully and its root cause determined. On the other hand, if it is a repeat failure mode, then this failing code sequence needs to be compared with the current known worst case for that failure mode and understood as well.

Failing code sequences can come from a variety of sources. Typically, each failing sequence will have a certain failure signature and much can be learned from it. Here, we will discuss three types of failures.

The first type of failure comes from self-checking code. A typical example might be code written to exercise and walk known patterns through the cache SRAMs. Such code will check its results and the failure messages will be self-explanatory.

The second type of failure is a final-state error, generally produced by random code generators. Random code generators produce tests that consist of initial CPU state, a sequence of assembly instructions, and an expected final state. When a test terminates, the final state is checked against the expected final state and discrepancies are noted. By analyzing the final-state error messages and looking at the test code sequence, one can infer quite a bit about the nature of the failure and come up with a set of experiments to further zero in on the failure.

The third type of failure is one in the framework code. Framework code is code written to allow tools such as random code generators to run on the hardware. The framework provides the environment for initializing memory, caches, and the architected state of the CPU. Sometimes, especially early in the project, failures will occur in running the framework code. In general, these are hard to debug since the failing code sequence (the framework) could be thousands or millions of instructions long.

The characteristics of the failure mode are determined by noting the sensitivities to different operating environment variables from the repeatability experiments above or through additional experiments at this stage. It is important to do some amount of debugging and failure characteristic determination to rule out most known failure modes to date.

To summarize, the task of bug identification is complete when we have accomplished all of the above and have made a reasonable effort to rule out known problems. We now have a new bug that is ready to be taken through the next task, debugging.

Debugging

The goal of the debugging effort is to determine the root cause of the failure and fix it on the chip in a new revision. The main steps to achieve this goal are gathering data about the failure, expanding the failure region, and hypothesizing the cause of the failure. These steps are all part of an iterative process that can lead us to our goal of complete understanding. As more data is gathered about the failure, a more complete and accurate hypothesis can be formed and a more accurate worst-case vector can be determined. On the other hand, new information may also prove that our initial hypothesis was incorrect. In that case, we go back to the data gathering step to acquire more information about the failure. When all of our data is consistent with our hypothesis, we have the root cause of the failure.

Gathering Data

Once we have determined from the bug identification process that the failure is one that we have not seen before, we need to gather as much data as possible about this new failure. If multiple chips fail in the same way, we may be able to correlate the failure with a specific wafer or lot or to a specific speed grade of the chip. For example, it is possible that only chips with extremely slow FETs will fail. Checking which revisions of the chip fail can tell us whether the failure is related to a recent change in the chip, or whether it has always been there.

The next step is to shorten the instruction sequence that will cause the failure. Often, failures occur in sequences of over 100 instructions, but the failure itself usually requires only a few specific data patterns and some specific instruction timing. Determining where in the instruction sequence the failure is occurring is one step toward isolating the failure. Occasionally, the failing instruction is easy to find. For example, if only one instruction in the case modified the failing register and the inputs to that one instruction did not change during the case, the failure has been isolated to the instruction sequence around that instruction. Usually, it is more difficult. If the failure causes an unrecoverable trap or reads bad data from the cache or main memory, we need more information before deciding where the failure occurred. For example, if a load from cache reads the wrong data, is it because another instruction stored bad data, or were the address lines incorrect during the read, or did the CPU corrupt the data after it was read?

Failures in the framework of a random code generator are quite difficult to debug, especially if the framework is written in a high-level language such as C++. If the failure can be localized to a specific code sequence, which could be quite long, it can usually be ported to a standalone case (no framework involved). From there, the same steps are taken as with any failure sequence to shorten the test case while maintaining the same failure mode.

Monitoring external events may be useful. Logic analyzers attached to external pins, such as the system bus interface and the cache interface, provide a picture of what the CPU is doing when the failure occurs and can help narrow down where in the code it is failing. We may see instruction fetches from main memory, which can tell us what area of the code the CPU is executing. Since the logic analyzer stores many, many previous states, we can look back through the execution of the case to see when bad data starts to appear. If the failure relates to an off-chip path, oscilloscopes can be used to verify the signal integrity of suspect paths. We have used this method when debugging noise-related problems and failures caused by imprecise impedance matching.

We would like to narrow down the code sequence to a very short sequence of instructions that will still fail in the same way as the original case. Creating a very short case such as this is not easy, especially considering that the PA 8000 uses out-of-order execution. Changing the original test case in any way may change the timing of certain events in the case such that it may not fail anymore. In gathering information about the failure, it is important to determine what events contribute directly to the failure and in what sequence the events must occur for the failure to occur. Just removing instructions starting at the beginning of the case may not help. Suppose that a load instruction, which normally would have caused a cache miss and a request to main memory, is removed from the beginning of the case. The behavior of the case will change because the next memory operation that accesses that cache line will cause the cache miss instead. This change in timing may cause two events in the CPU that were concurrent in the original case to be separated by many states in the modified case. Removing instructions may also have the effect of changing register data patterns that may have been required for the failure. If an add instruction that sets up a 0x0F0F0F0F data pattern in a register is removed, that register will contain its initial value instead—different from the pattern set up by the add—and the failure may not occur.

Experiments with the failing code are still very important. Removing irrelevant instructions and data can narrow the search for the failure. It is possible that a large number of instructions in the failing sequence can be removed without affecting the failure. The data patterns in the source registers for one or more instructions might be changed without affecting the failure. Each of these changes narrows the search for the failure mechanism. Very slight changes in the code sequence or data patterns can provide information on what events are necessary for the failure. If we change only one bit in one data pattern and the failure goes away, that is a big indication that the failure requires one bit to be set a certain way. Another useful step to narrow our search is to determine if any specific CPU features are involved in the failure. For example, if we can turn off a specific CPU feature, such as bypassing a register value from a pipeline stage, and the case now passes, we might say the failure is occurring in the bypass logic, or at least the timing of the case requires a bypass.

While we are doing the code experiments, we may use some of the on-chip test and debugging circuitry to get a better picture of the failure. By running the chip in both a passing region and a failing region and comparing the two runs, we can get a picture of where the failure starts.

Using all of the data gathered, we can begin to see the overall picture of the failure. We know under what conditions the failure will occur, including frequency, voltage, temperature, and process parameters. We know a short code sequence that will fail, and what CPU features and timing affect the failure. We have a partial picture of the internal state of the failure by comparing passing and failing runs.

At the same time that we are gathering data about the failure, we are developing a new test case for this failure. This new case will use the known requirements for the failure to occur, including specific instruction sequences and timing, specific register values, and cache hits and misses. When our new case fails, we have all the elements of the failure.

Expanding the Failure Region

In most instances, the particular failure that occurs in a random instruction sequence with random data patterns is not the worst-case failure. We would like to know how severe the problem is. One of our goals is to find the worst-case vector. Failures that were previously outside the operating region usually move into or very close to the operating region with a worse vector. For example, a speed failure may occur at a significantly lower frequency when a worst-case data pattern is used. Or maybe a failure that only occurred at 40°C will now occur at 20°C. Expanding the failure into more general shmoo conditions also helps in gathering more data. (It's not much fun to probe signals in a 40°C oven.)

Electrical problems are often heavily influenced by data patterns. For example, driving different data patterns on an internal bus may increase or decrease the capacitive coupling or delay of a signal that is causing the failure. It is unlikely that the random data pattern used in the original failing case is the absolute worst for this particular failure. Complicating the matter, it may not be clear what other signals could be affecting the failing signal.

There are some cases in which the failing signal may not be appreciably affected by any other signals. In other cases, the failing signal, which might be part of a bus, may be influenced by certain data patterns on that bus. For instance, some of the signals that make up the bus may capacitively couple to the failing signal in that same bus, slowing down the failing signal or inverting its value. Occasionally, the biggest influence on the failing signal is a bus that is functionally unrelated to the failing signal, but is in close proximity physically. The same type of capacitive coupling can occur in this case.

Changing obvious data patterns—instructions themselves, operands from registers, and data results from the ALU or the cache—is the first step. If none of these seem to affect the failure, the layout can be consulted to see what buses or signals run adjacent to or on top of the victim signal. Finding one or more data patterns that influence the failure also allows a better understanding of the failure.

Hypothesizing the Cause

In hypothesizing the cause of the failure, all of the information that has been acquired about the failure will be used.

The minimum code sequence is especially useful to the circuit debugger. First, it provides a list of code sensitivities that either turn the failure on or off or expand the failing region. Second, this code can be simulated by a switch-level simulator to give the debugger full observability into the on-chip circuits being exercised by the test case. By comparing simulations with and without the code sensitivities, the exact effect of the code on the circuits is observed.

The switch-level simulator is the first tool used by the circuit debugger. The exercised circuitry is compared together with the internal state differences from the passing and failing data captures to narrow down the circuits involved. This process yields several potential code experiments that will continue to narrow the playing field. At this stage in the debugging cycle, the circuit debugger is working side-by-side with the system debugger to isolate the failure.

Eventually, the circuit debugger makes a root-cause hypothesis as to the cause of the failing behavior. This hypothesis can frequently be supported by the switch-level simulator. For example, if the hypothesis is that a certain latch fails to make setup, this latch can be forced to fail in the simulator. The resulting simulated failure mode should match the actual failure mode. This is the point when the circuit debugger moves to SPICE as the main debugging tool.

SPICE is used to simulate the isolated failing circuitry in the appropriate failing conditions. In the latch example, the clock and data paths into the latch are accurately modeled in an attempt to reproduce the failure in SPICE under the same conditions as on real silicon. Differences are assumed to be either inaccuracies in the modeling or mistakes in the root-cause hypothesis. Obviously, these differences need to be explained before root cause is declared.

If the failure is frequency dependent, another way to validate the hypothesis is to stretch the specific clock phase during which we believe the failure occurs. By stretching a clock phase, we provide more time for the CPU to do the required work of that phase. For instance, if we have a speed failure related to one specific phase and we lengthen that one phase by 10%, the failure should get better.

We continue to gather data and hypothesize causes of the failure until our hypothesis passes the root-cause test.

Declaring the Root Cause

Often, by inspection of the failing case in the switch-level simulator or SPICE, sensitivities to local circuit behavior are predicted. This prediction can then be verified by targeting code to induce this behavior. For example, we may try to precondition a particular bus with a specific data pattern such that it no longer transitions as in the failing case. Or we may change the instruction timing of the case such that two events no longer occur in the same cycle. If we can turn the failure on and off (the light-switch test) by changing one of the known sensitivities, we have a good understanding of the failure.

Throughout the debugging cycle, all facts and observations are documented thoroughly in a bug database. This database is used to drive experiments to fill in data where it is missing or to explain observations. The entire weight of this data is compared to the root-cause hypothesis for consistency. Any data point in conflict needs to be explained before the root cause of the bug is considered known. This diligence to the data has avoided many premature and wrong root-cause analyses.

Once we have demonstrated a light-switch test and our hypothesis agrees with all of the data, we are at the root cause. We believe we fully understand the failure.

We then revisit the worst-case vector analysis one final time. Even if our final worst-case vector does not move the failure into the operating region, we may fix the problem anyway, because it is hard to determine if a small process shift later in this product’s life could move this failure close to or into the operating region.

The next step is to fix the problem. If we can fix the problem with only a change in the metal layers, the turnaround time for new chips is much shorter. To verify that the proposed fix will actually eliminate the failure, we may do a FIB (focused ion beam) experiment, in which one of the existing chips is modified (metal lines are cut and new ones are deposited) to include the change. The chip is characterized before and after the FIB change to determine how the failure was affected. If the failure was eliminated, we have good confidence in our fix, and we will put our fix into the plan for the next CPU revision.

Creating the Golden ROM

After a bug has been closed, its worst-case test is added to our ROM of all other worst-case tests that have exposed bugs. We call this ROM the *golden ROM.* We use the golden ROM for much of our volume shmoo testing, to serve a number of purposes. It shows where the current bugs were found and can show how a fix or certain chip characteristics could affect these failures. It also lets us know if a bug has been reintroduced, which happens on occasion. As the golden ROM grows in size, it naturally gives us more coverage. Many of our new bugs are found by running the old bug code in our golden ROM. If a test case has important coverage that we do not have in our tester screens, golden ROM tests can be converted into broadside vector tests for our package screens.

Updating the Methodologies

When the root cause or problem circuit has been identified, it often uncovers a flaw in our design methodologies. This implies that other similar circuits may be used on other parts of the chip but haven't been discovered yet. The aphorism "If there's one rat, there are many rats" becomes our motto. A "many rats" investigation is launched to find a tool-based method of extracting similar circuits from the chip database and fixing them if appropriate. Quite often, the failing circuitry has a unique topology that can be searched for with a tool. Finally, this flaw in the design methodologies is documented and the methodologies are updated.

Conclusion

We continue the electrical verification process until we have searched our matrix of variables—temperature, frequency, voltage, process, and test cases—and we can find no more failures that we believe could move into the operating region. This process spans multiple chip revisions, with each new revision fixing one or more failure mechanisms. This process ensures the long-term quality of the product throughout its lifespan.

In addition, we analyze the problems that we found and integrate the solutions to these problems into our design methodologies so that future products can avoid the same pitfalls and potentially reach high quality levels more quickly than previous products.

Acknowledgments

The authors would like to thank everyone who contributed to the successful electrical verification of the PA 8000 CPU. We especially want to acknowledge the hard work of the electrical verification team in the Engineering Systems Laboratory in Fort Collins, Colorado for finding and fixing some tough electrical bugs. We would also like to thank our system partners—the General Systems Laboratory in Roseville, California and the Fort Collins Systems Laboratory in Fort Collins, Colorado—for their help in sourcing electrical bugs.

Shmoo Plot Shapes

A shmoo plot is a graph that represents how a particular test passes or fails when parameters like frequency, voltage, or temperature are varied and the test is executed repeatedly. The shape of the failing region is meaningful and helps in determining the cause of the failure. Shmoo plots typically fall into familiar categories with descriptive names.A shmoo plot of normal circuit operation shows better high-frequency performance as supply voltage increases, as shown in Fig. 1a. However, other shapes frequently seen include the curlback (Fig. 1b), ceiling (Fig. 1c), floor (Fig. 1d), wall (Fig. 1e), finger (Fig. 1f), and breaking wave (Fig. 1g).

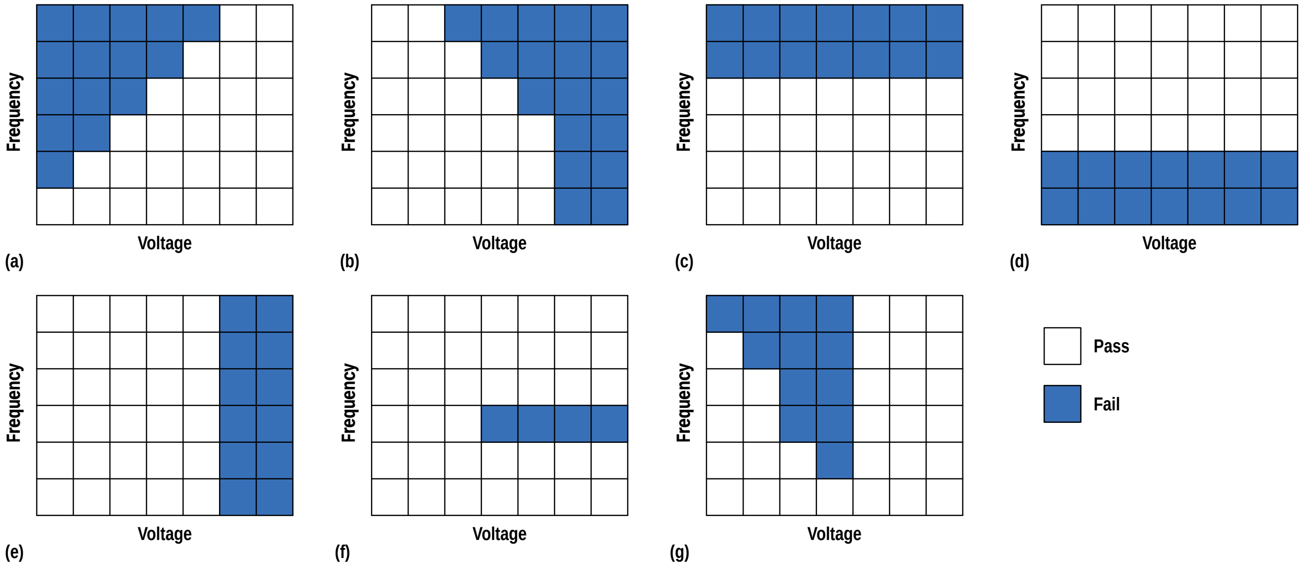


Fig. 1. *(a) Normal shmoo plot. (b) Curlback. (c) Ceiling. (d) Floor. (e) Wall. (f) Finger. (g) Breaking wave.*

Solving IC Interconnect Routing for an Advanced PA-RISC Processor

This paper discusses some important new block routing technologies that were required for the HP PA 8000 processor chip. These technologies are implemented in a new block routing system called PA_Route.

by James C. Fong, Hoi-Kuen Chan, and Martin D. Kruckenberg

The design complexities of today's microprocessors have grown significantly, with the number of transistors climbing to well over a million, silicon die sizes larger than 1.6 cm^2, clock speeds exceeding 150 MHz, and short design cycles caused by competition. These issues create tremendous pressure on design teams and the tools they use. The PA 8000 CPU design team used powerful design automation tools to achieve their design goals.

Layout of the interconnect metal on the chip is one of the key components of advanced designs. It is vital to address the increasingly complex layout problem to achieve smaller die sizes, higher performance, and quicker time to market. Since the early 1980s, HP has been working to solve the top-level IC interconnect problems associated with many of the larger HP-designed and HP-manufactured ICs. This paper will discuss some important new block routing technologies that were required to implement the HP PA 8000 microprocessor. These technologies are embodied in a new in-house block routing system called *PA_Route*.

Buy or Build Decision

Frequently we at the HP Integrated Circuit Business Division (ICBD) are approached by HP design teams who are about to embark on the design of a new chip to be manufactured by ICBD. We are asked to enhance our routing technology to address issues critical to the chip's successful routing.

This was the case when the PA 8000 design team approached us with some ideas for new features needed to take advantage of new technologies. Like most aggressive designs, they were pushing the limits of every technology where they thought they could get a significant return on investment. Block routing was one area they thought they could improve.

Our existing block router is called HARP (Hewlett-Packard Automatic Routing and Placement). HARP had been evolving for over a decade and had some legacy code that was becoming difficult to extend.

Using customer surveys to complement our own knowledge, we did a detailed analysis of various existing block routers. We wanted to see how they address this new class of block routing problems. Design teams are hesitant to switch layout tools unless alternatives can be found that match their design requirements well enough to justify the risks of switching tools and the cost of the new tool, not only the capital cost but also the cost of learning how to use the tool effectively. Using radar charts (see Fig. 1), we were able to determine that the less aggressive style of chips represented by the PA 7100LC processor

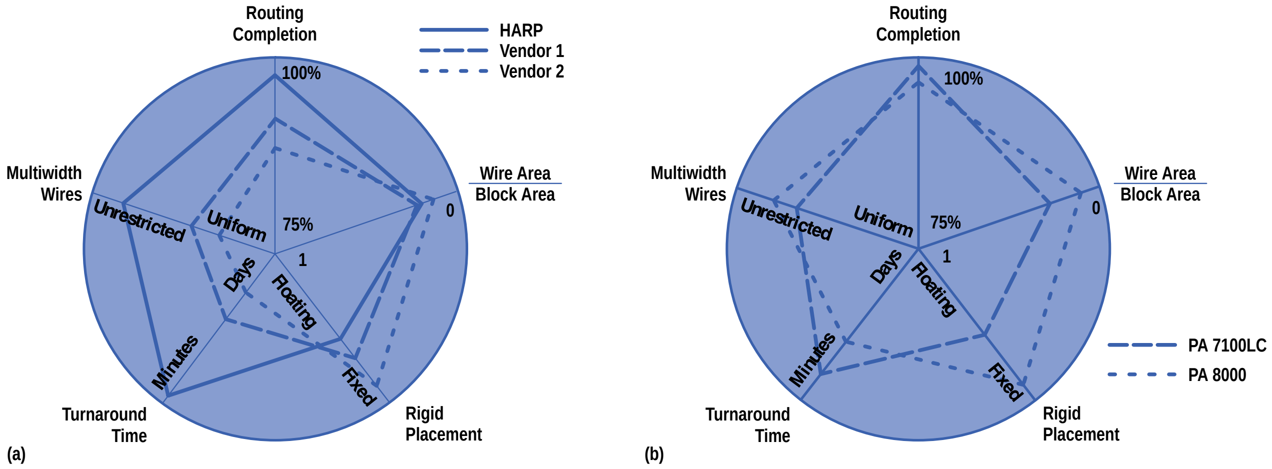


Fig. 1. *Radar charts showing (a) the capabilities of HP's existing HARP block router and third-party routers and (b) the needs of less aggressive chips like the HP PA 7100LC and more aggressive chips like the HP PA 8000.*

were well-suited for the existing HARP system. The more aggressive style of chips represented by the PA 8000, however, did not map to any existing block router offerings.

Significant changes in functionality and features usually entail a high amount of risk. On any design it is critical to manage the risk. Being an internal supplier, we are able to work much more closely with our customers by giving them greater visibility and control of the risks involved. This level of access is generally not available when dealing with third-party tool providers.

ICBD, as HP's internal chip supplier, is in the business of making and selling chips, not tools, so we face stricter requirements to justify any internal tool development. It is rarely cost-effective to build a router for a single chip. However, it has been our experience that the microprocessor chips have always pushed the limits of the technologies and the more general ASIC chips follow later after the bumps have been smoothed out. In looking at what was special about the PA 8000, we spotted several new technology trends that radically changed the block routing problem and might be adopted by future ASIC chips.

First, fabrication processes are starting to add many more layers of metal for interconnect. This change begins to invalidate the basic model used by traditional block routers of separate routing channels and blocks. Second, the need for higher off-chip connectivity is forcing a change in packaging technology. Solder bump packaging looks like the most viable means of addressing that need. Solder bump packaging is also being looked at for reducing packaging cost by mounting chips directly to boards. However, having solder bump pads in the middle of the chip breaks the traditional block router model of placing the pads at the periphery of the chip. Last but by no means least is a general trend of wiring delays becoming more significant than gate delays. Thus, the emphasis of routing is switched from minimizing chip area to minimizing interconnect delay.

Working with our PA 8000 customers, we prioritized the features and came up with a manageable subset needed for them to be successful. We then circulated a proposal to build the PA_Route block routing system. Given the time constraints and the ambitious goals, we had to take the drastic step of freezing the old system, HARP, with minimal support. We got agreement from all parties on the basis of strong support from the PA 8000 developers.

New Technologies Lead to New Constraints

The PA 8000 design team had decided that to be competitive, the PA 8000 chip would not only be more aggressive in its design, using superscalar, out-of-order instructions, but would also use a new process and new packaging. It is not uncommon for a microprocessor to use a new process, but this time they were moving from a three-metal-layer process to a five-metal-layer process. In addition, the increased I/O requirements of the design ruled out conventional packaging. The only mature packaging approach available was solder bump technology. With solder bump technology, the I/O pads are spread across the whole chip and are not just restricted to the periphery.

Analysis of the possibilities for extending or adapting the existing block router in the HARP system showed that its basic design intentions were not well-matched with the new requirements and could not be changed to make full use of the new technologies. HARP was based on the traditional channel routing paradigm, in which there are expandable routing channels between solid blocks. The channel router was limited to routing in three metal layers, while the new process had five. The solder bump I/O pads could be anywhere, but the block router could only attach to ports on the edges of a block.

Another more important restriction was that the solder bump port frame could not change and the blocks could not move with respect to the pads. There were two reasons for this. First, the board to which the PA 8000 was to be connected had a relatively long lead time, so its designers could not wait for the chip to be completed before getting started. Secondly, the solder bumps used to connect to the I/O pads emit alpha particles. If the placement were changed so that the pads became coincident with sensitive circuitry then unpredictable circuit behavior could result.

These requirements meant the block router could not grow the channels or move the blocks, a constraint for which our existing block router had only weak support. We jointly decided it was not feasible to attempt to automate the routing of the fifth layer of metal, since it was overly complicated by the requirements of the solder bump I/O pads.

In PA_Route we addressed as many of the new requirements as we could in the time available. We worked with the PA 8000 design team to pick the most important issues to address. This came down to two major features: being able to use the third and fourth metal layer resources over the top of some of the blocks and being better able to control the growth of the placement.

Our team is not often given the time to redesign our system, so we took advantage of the opportunity to add some long-desired capabilities. We added a more sophisticated port and net model, which we call *foliage*. With foliage we can describe the electrical characteristics of a port and a net. Pieces of artwork representing a port, for example, can be considered electrically equivalent (allowing stitching), electrically resistive (allowing connection to one of many but without stitching), or electrically open (specifying that all pieces of artwork must be connected). Foliage allows the router to be more flexible in using ports, since it uses this electrical model of the ports and it allows for more complex routing of nets in a channel. We also took the time to use more advanced software development techniques. We switched our design style from structured custom programming in the Ada language to object-oriented programming in the C++ language. This allowed us to attempt more complex algorithms and reuse existing component libraries.

The Building of PA_Route

The PA_Route system is composed of many components, including a netlist reader, an artwork reader (which models obstacles), a global router, a channel scheduler, and a detailed router. A viewer is used to examine intermediate and final results. Eventually the artwork is produced and then verified. Even though the design time of the PA 8000 is long compared to most ASIC chips, we did not have time to rewrite the whole system, so only three main parts were designed and implemented from scratch: the main database, the global router's over-the-block grid model, and the new over-the-block detailed router. We leveraged the rest of the system from the old HARP system with modifications to interface with the new database.

To minimize development time, we partitioned our development team into two parallel groups. One group started on the new database and started porting the old programs, while the other started implementing some of the new global router features in the old system. When most of the old HARP system was ported we ported the modified global router. This meant that the global router stayed in structured custom Ada code, which was the language used in HARP.

Database Changes

The capability needed by the PA 8000 design to route over the blocks required us to improve the expressiveness of the underlying database models used in routing regions. The new database allows us to model the obstacles and internal ports that we see in these over-the-block regions. The advanced port and net models (foliage) we implemented also required significant changes to the database. This not only allows us greater control and flexibility in routing, but also allows us to separate the act of global routing from the channel scheduler that calls the detailed router.

We developed automatic code generation technology to transform a graphical model of the database into code. The code generation technology was extended to support the C++ language and we began to work on the new input and output programs. With the database changes completed, we could begin porting the old HARP programs to the new database.

Global Routing

The general global routing problem is described at right. PA_Route incorporates a global router that understands rectangular blocks. The global router needed to be extended to support L-shaped blocks for the PA 8000. An L-shaped block is cut either horizontally or vertically into two rectangular components and special control is imposed on the channel between the cut components to keep the components linked together. The routing plane is divided into rectangular routing regions that meet only at T-intersections such that only two sides of a routing region have constrained ports. This somewhat restricted routing model comes from a conscious decision to avoid situations in which a routing region becomes a "switchbox" with ports on all four sides constrained to fixed locations. The more constrained switchbox routing problem generally requires more run time, creates more constraint cycles, demands clever rip-up and reroute strategies, and tends to leave more shorts for manual repair. We opted instead to concentrate our effort on providing more flexibility in the PA_Route global router for meeting user requirements and for achieving the smallest possible overall chip area.

The old HARP global router, like any traditional global router, assumes that blocks are black boxes, that the points for connections are on the edges of the black boxes, and that routing is confined to the channel areas between the blocks. That is, all routing resources inside the blocks are dedicated to the blocks' internal implementation only and therefore routing at the global level is not allowed to traverse through the blocks. This simplistic assumption was largely accurate in the days of two-layer and three-layer IC processes. With the advent of the HP CMOS14 process, which can have up to five routing layers, the assumption that routing resources inside a block are dedicated only to the block is no longer realistic. For the PA 8000, a good amount of metal 3 and metal 4 resources inside some child blocks are available for routing global nets.

Being able to use such over-the-block routing resources can lead to reduced signal timing, decreased channel congestion, and ultimately smaller overall routed chip size. Having judged that over-the-block routing was a critical factor to the success of PA 8000, the PA_Route team undertook a revolutionary change in the global router to support the routing of global nets over any block, provided that there are routing resources available over the block. The traditional global routing graph was augmented with a virtual grid model over each child block, a sophisticated net flow optimizer, and an efficient routing resource estimator. The grid model allows the lowest-cost path of a global net to traverse through any region over a block as long as there are free routing resources. The global router builds a detailed model of routing resources in each region (channel or block) and tracks free spaces in the regions based on a sophisticated density estimator that understands obstacles. The net flow optimizer minimizes jogging and distributes unavoidable jogs of different nets to different regions to reduce congestion. For connecting to the new solder bump I/O pads, which are inside some child blocks, the new global router was extended to support ports inside any block, with the restrictions that the ports be on selected port layers and that there be available routing resources in the block. The global router takes care of avoiding obstacles and ports on the edges of a block when inside ports are brought out of a block to form a lowest-cost path. The net flow optimizer also plays an important role in choosing an optimal exit point for the inside port so as to reduce unnecessary jogging.

The predetermined solder bump I/O pad locations for the PA 8000 force the placement to be unperturbed during routing. This is a hard problem for the global router. Not having the luxury of actually routing the nets during global routing, utilization of routing resources over the block as well as in the channel regions is controlled using a close estimate of detailed routing. A reasonably accurate and fast density estimator was incorporated into the PA_Route global router. Since routing over the block is allowed, the density estimator must understand prerouting and obstacles. A density check phase was introduced after the evaluation of the lowest-cost path of a net. If the path would exceed the routing capacity of one or more regions, the grid model in the global routing graph is modified to forbid further routing through the congested portion of the regions and an alternate lowest-cost path is sought. This checking process is repeated until either a clear path is found or no clear path is found, in which case the net is left unrouted to preserve the fixed placement. In addition to performing accurate density calculations, the global router also attempts to achieve minimal placement perturbation by automatically assigning nets to the less-dense layer wherever there is more than one routing layer available, and by optimizing net flow at region interfaces to reduce routing congestion.

For multipin nets, for which connectivity can have significant performance and density implications, port foliage was added to the PA_Route database to give the global router a model for determining port equivalency, while net foliage was introduced to allow the global router to generate more sophisticated physical connectivity for the shortest path. This combination of port and net foliage results in a high degree of control over the physical connectivity of a net. A designer can specify foliage explicitly or allow the PA_Route global router the freedom to optimize the global route by creating foliage as necessary to minimize the total wire length and to avoid congestion.

Over-the-Block Routing

To handle two important aspects of the PA 8000, a new over-the-block detailed router was required. The router had to handle obstacles in any routing layer and it had to be able to connect to ports located anywhere within the routing region.

Restrictions on the topology of the obstacles and ports were negotiated with the PA 8000 design team to relax the constraints of the over-the-block router to meet an aggressive schedule.

Child blocks were constructed by lower-level composition teams. Their design used the lower layers of metal to perform local interconnect and the upper levels of metal for intermediate levels of interconnect. The result was that partially used layers were made available to the over-the-block router to complete the intermediate and global level interconnect. The over-the-block router was given the responsibility of avoiding artwork created by the lower-level composition teams.

A review of existing detailed routing algorithms is presented ***Subarticle 5a***. The PA_Route over-the-block router is built on a new routing algorithm. It is based on a channel-like paradigm although it handles obstacles with arbitrary configurations. Layers are generally assumed to run in either the x direction or the y direction. Wires are routed assuming one direction is preferred, that is, in the preferred direction the wire runs for a longer distance and carries most of the current between a source and its sinks. By handling obstacles in arbitrary configurations, the over-the-block router extends channel routing concepts into an area-based routing regime. It retains many of the benefits of channel routing while being flexible enough to handle more complex routing topologies. This type of routing methodology will become more common as more layers are made available for routing. The over-the-block router can connect to ports not only on the sides of routing regions, but also in the middle of a routing region. The over-the-block router supports variable wire width and spacing, which gives the designers greater control over the timing delays of a signal. The over-the-block router reads the layout rules directly and does not abstract them into arbitrary routing constraints. Unlike other algorithms, our proprietary over-the-block algorithm does not require wires to be “binned” according to their width and spacing, and it does not rely on a compaction process to achieve optimal density.

The over-the-block router contains the features needed for high-speed, performance-driven critical designs such as the PA 8000. It handles complex via structures necessary for high-performance designs by allowing the intersection area to be expanded beyond the size dictated by individual metal connections. While it supports a high degree of manual control, the over-the-block router is also reasonably fast, making multiple design turnarounds feasible.

The over-the-block router models the routing problem as two-dimensional line segments that represent the largest wiring component of a net. This *trunk* is assumed to cause most of the parasitic delay and the overall goal of the algorithm is to find an optimal ordering of these trunks to generate a dense packing and avoid obstacles. Each trunk and each obstacle becomes a node in a graph. The edges in the graph model the vertical pin constraints of each wire and the horizontal constraints of that trunk’s placement relative to other trunks (see Fig. 2). The total graph contains the weighted constraints of all trunks in the routing region. Thus, each trunk is considered for placement during each phase of edge direction assignment, and the net ordering difficulties of other routing schemes are avoided. The general nature of the edge selection allows other constraints such as cross talk and delay to be modeled in future versions.

The algorithm can handle any number of layers and is not rigidly required to follow layer-per-direction constraints for vertical components (i.e., connections to ports) or trunk components. When constraints occur, the over-the-block router tries several schemes to alter the topology of the wire, such as removing the constraints. The scheme includes *jog insertion* and *wrong-side segmenting* in various forms.

If the over-the-block router cannot complete a route, it produces a spacing violation or short circuit along with associated diagnostics and completes the route. When this occurs, the user has the option of fixing the short manually or altering the routing problem for the region by such methods as growing the placement, constraining the global routing with capacity controls, or other means.

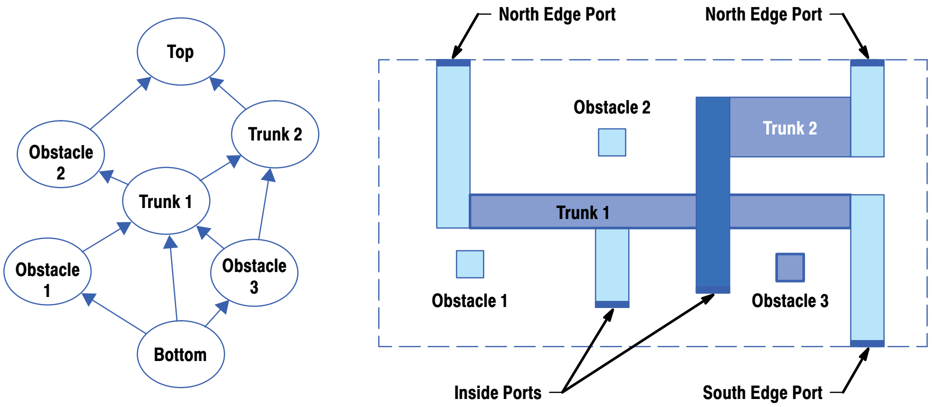


Fig. 2. *To the over-the-block detailed router, each wiring trunk is a node in a graph (a). The edges in the graph model the vertical pin constraints of each trunk and the horizontal constraints of that trunk's placement relative to other trunks. (b) Routing plan. Shades of gray represent different metal layers.*

Although specialized to handle the routing problems of the PA 8000, the over-the-block router was built to handle the general channel routing problem. No shortcuts were taken that would compromise robustness for the general case in the expectation that the router could be leveraged for other designs.

Conclusion

A new block routing system called PA_Route was built specifically to address the needs of high-performance, leading-edge IC designs. PA_Route contains significant features built on new technology while leveraging existing code to minimize risk. It was designed to be extendable to address future issues as they arise. It was used successfully to route the PA 8000 chip and did not impact its schedule despite the high levels of risk involved. Fig. 3 shows the areas of the PA 8000 chip where PA_Route performed block-level routing. The features and limitations of the system were carefully designed with close cooperation between the PA 8000 design team and the CAD development team. Many alternatives were analyzed using design-critical issues as the measurement criteria. Balancing immediate and future chip design needs was given high importance in the design of PA_Route so that the system can continue to be used for future designs.

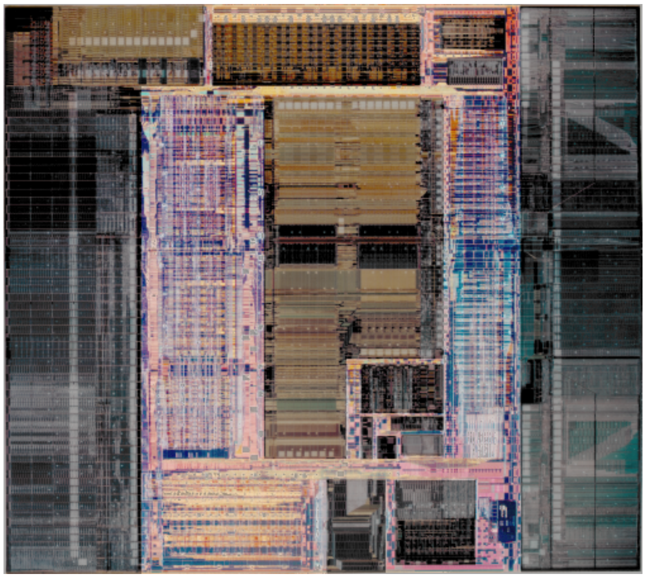


Fig. 3. *PA 8000 CPU chip with highlighted areas showing where PA_Route performed block-level routing.*

Acknowledgments

The PA_Route team would like to thank the members of the Engineering Systems Lab in Fort Collins with whom we worked closely, especially Julie van Gelder and Tom Asprey. Julie was invaluable in helping define features and requirements. Tom showed much patience and persistence with the early releases of PA_Route. We would like to acknowledge Dell Lee of the ICBD Design Technology Center, as well as Michaela Guiney and Eric Slutz, for supporting our efforts. Thanks to Sandy Fogle and Gaylord Tefft for providing a photomicrograph of the PA 8000.

PA_Route is available to designers within HP. The URL for PA_Route is http://emerald.dtc.hp.com/pa_route/general.html.

Global Routing—A Block-Level Problem

For a given level within a chip hierarchy, the routing plane is generally occupied by a number of blocks with ports that need to be connected by physical wires on nets. The blocks are usually restricted to be rectilinear in shape but are allowed to vary in size. The space between the blocks is generally reserved for routing and is usually subdivided into adjacent routing *regions* or *channels* so that the routing problem can be solved with a divide-and-conquer approach. Within each region, a certain number of layers are reserved for routing the signals at the given level. Global routing is the first step in the routing process. Its job is to generate a routing plan in which each signal is assigned to a number of routing regions. The objectives for the global router are to achieve 100% assignment of signals to available routing regions, to minimize the overall chip size, and to ensure that the timing requirements of the signals are met.

The global routing problem is generally represented by a *global routing graph*, which depicts the relationships between routing regions and ports to be connected. The edges of a global routing graph represent the routing regions and the nodes represent the intersections between regions and the ports. The edges are assigned *weights*, which can be the distance between two region intersections, the distance between a port and the nearest region, the cost for using a particular layer in the region, a penalty for switching routing layers, or a penalty for overflowing a region. Global routing is accomplished by implementing a lowest-cost path-finding algorithm on the global routing graph.

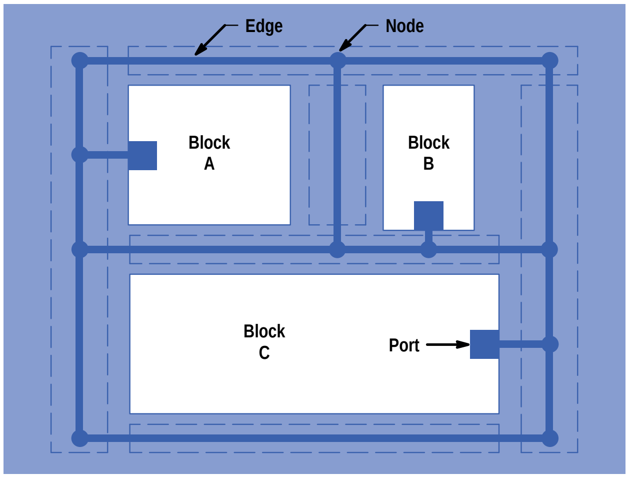


Fig. 1. *Global routing graph. Regions are indicated by dashed outlines.*

Detailed Routing Methods

Detailed routing has generally evolved out of four basic approaches: maze routing, line probe routing, left-edge routing, and greedy channel scanning. The problem is formulated as a routing area containing connection points or *pins* on a rectilinear (usually rectangular) region or *channel*. Pins can be located on any of the four sides of the region or within the region. The connection points are generally constrained to reside in certain layers to make them easier to connect to.

Even single-layer routing problems are NP-complete, which means that an optimal solution cannot be achieved in a reasonable time. For this reason, detailed routing solutions are heuristic in nature. The factors in determining a solution's usability are the number of terminals, net width, via restrictions, boundary shape, number of layers, and net types such as power, ground, and clock wires.

Maze routers abstract the channel routing problem with a grid-based model. Wires are restricted to follow paths along the grid lines. Routing is accomplished by laying down wires on the grid one at a time. Obstacles are modeled as disallowed portions of the grid. Therefore, maze routing can handle arbitrary obstacles.

Line probe routers scan in the x and y directions searching for line segments from either the source or the destination. Scan lines do not project beyond obstacles, so obstacles are avoided by a subsequent probe of the line segments orthogonal to the ones from the previous pass.

Left-edge routers sort wires by the boundary formed by the leftmost and rightmost pins. It orders wires one at a time using a greedy method that places segments into tracks. It fills tracks one at a time, packing segments to minimize unused space in a track. The route is complete when all wires have been assigned to a track.

Greedy channel routers divide the channel into horizontal tracks and vertical columns. This approach works on one vertical column at a time, scanning from left to right. The approach is termed "greedy" because each column is optimized individually, although the entire channel is not guaranteed to be optimal. The greedy router sweeps from column to column, trying to join segments of nets assigned to multiple tracks. The greedy channel scan is capable of providing fast solutions but cannot be easily extended to handle arbitrary obstacles.

The over-the-block detailed router used in PA_Route uses a completely different approach based on a graph. The graph represents horizontal and vertical constraints of the wires.

Intelligent Networks and the HP OpenCall Technology

The HP OpenCall product family is a portfolio of computer-based telecommunications platforms designed to offer a foundation for advanced network services based on intelligent network concepts. This article concentrates on the HP OpenCall *service execution platform*, *service management platform*, and *service creation environment.*

by Tarek Dehni, John O'Connell, and Nicolas Raguideau

Intelligent networks are an expanding area within the telecommunications industry. The adoption of intelligent network technology has been driven by its ability to allow telecommunications network operators to install and provision new, revenue-generating communication services in their networks. With these services installed within the network, the extra functionality they provide can easily and instantaneously be made available to the whole customer base. Examples of such services are the freephone services (the cost of the telephone call is paid by the called party), credit card calling, and the CLASS services (custom local area signaling services) in North America.

At the same time, the standardization of some key interfaces within the telecommunications network has allowed greater competition between network equipment providers, offering the possibility of genuinely multivendor networks. The opening up of previously proprietary switch interfaces has made it easier for network operators to add new functionality to their networks, since this functionality can now be implemented outside the switch, often on industry-standard computer platforms. Today, with the emergence of new fixed and mobile network operators in many areas of the world, two new drivers for intelligent networks have emerged. Firstly, there is the need for interoperability between these networks. Secondly, operators seek to differentiate themselves on their service offerings. Both imply an even stronger requirement to support extra intelligence in the network. This will ensure the continued demand for more open and flexible intelligent network solutions.

Hewlett-Packard's product strategy for the intelligent network market is based on the HP OpenCall product family, a portfolio of computer-based telecommunications platforms designed to offer a solid foundation for competitive, revenue-generating services based on intelligent network architectures. This article concentrates on the HP OpenCall *service execution platform*, *service management platform*, and *service creation environment*, with particular emphasis on the architecture and design of the service execution platform. The HP OpenCall *SS7 platform* is described in ***Article 7***.

In this paper, we introduce the key concepts in intelligent networks including the role of standardization, we explore the system requirements for a class of intelligent network elements (those elements targeted by the HP OpenCall platforms), and we highlight the key aspects of the design of the HP OpenCall platforms.

Intelligent Networks

The telephony service is very simple. Its objective is the transport of speech information over a distance in real time. Telephony networks were originally designed with the assumption that the same service would be offered to all users, and this held true for a long time. Users could select one of a range of destinations and be called by other users. Over the years the focus of telephony service providers has been to improve the technology to offer these basic services to a larger number of customers, and over longer and longer distances. At the same time, terminals have become mobile, with mobile phone users demanding the same levels of services.

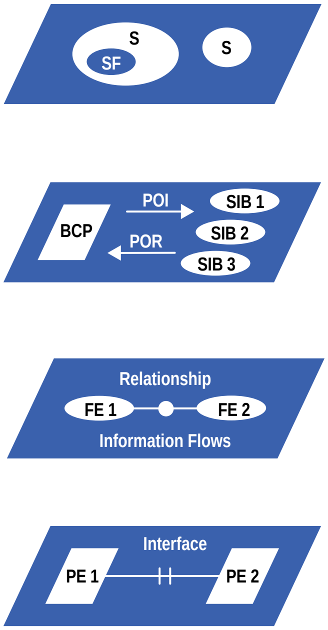
As a consequence of this evolution, today's telephony networks consist of a mix of more or less integrated technologies and networks that have been deployed over more than 30 years, forming a very complex and large-scale global infrastructure.

In this context, the task of provisioning a new service in an operator's network is extremely complex and may vary considerably depending on the network infrastructure. The conventional method of continually integrating these new functions into public exchanges is costly and lacks flexibility, making it difficult for network operators to compete effectively in an increasingly competitive environment.

This situation has led network operators and their suppliers to look for a better approach, in which control functions and data management linked to the creation, deployment, and modification of services can evolve separately from the basic switching or existing functions of an exchange. They assigned standards organizations (ITU-T, ETSI, BellCore) the responsibility of defining an architectural framework for the creation, execution, and management of network services.

Intelligent Network Conceptual Model

The ITU-T (International Telecommunications Union—Telecommunications Standardization Sector) developed the *Intelligent Network Conceptual Model* to provide the framework for the description of intelligent network concepts and their relations. The Intelligent Network Conceptual Model consists of four planes, each of which is a different abstraction of the telecommunications network. The ITU-T also planned the specification of the target intelligent network architecture through several study periods, thereby enabling incremental implementations. These successive standardized functions are referred to as *intelligent network capability sets* (see Fig. 1).



Service Plane:
- Extraction of service features to clarify service requirements
- S = Service
- SF = Service Feature

Global Functional Plane:
- Service execution description making use of SIBs and BCP
- BCP = Basic Call Process
- SIB = Service Independent Building Block
- POI = Point of Initiation
- POR = Point of Return

Distributed Functional Plane:
- Basic Call State Model (BCSM) models basic call process
- Description of service execution
- FE = Functional Entity

Physical Plane:
- Physical implementation scenarios for intelligent network functionality
- Define protocol: INAP (Intelligent Network Application Protocol)
- PE = Physical Entity

Fig. 1. *The Intelligent Network Conceptual Model of the ITU-T is a framework for describing and specifying intelligent network systems.*

Service Plane. The *service plane* describes services and the service features as seen from a user perspective. A service feature is the smallest part of a service that can be perceived by a user. The service plane does not consider how the service is implemented or provisioned in the network.

Global Functional Plane. The *global functional plane* describes the design of services as a combination of *service independent building blocks*. Service independent building blocks give a model of the network as a single entity, that is, there is no consideration of how the functionality is distributed over the network.

A specific service independent building block is the *basic call process*, which corresponds to the basic call service. It has *points of initiation* and *points of return*. An instance of a service logic can be called from a point of initiation, and after execution of the service logic, the basic call process is recalled in a point of return. Service logic corresponds to services or service features in the service plane.

Distributed Functional Plane. The *distributed functional plane* (Fig. 2) gives a distributed functional view of the network. *Functional entities* are groupings of functionality that are entirely contained in a physical entity. In other words, they cannot be split among several physical entities. The distributed functional plane describes the functional entities together with their relationships.

The identified functional entities are as follows:

- The *call control access function* models the interface with the end-user terminal.
- The *call control function* provides call and connection control, that is, basic call processing.
- The *service switching function* models the call control function as seen from the service control function.
- The *service control function* provides the logic and processing capabilities for intelligent network-provided services. It interacts with the service switching function to modify the behavior of the basic call, and with the two entities below to access additional logic or obtain service or user data.

- The *service data function* contains customer and network data for real-time access from the service control function.
- The *specialized resource function* provides additional specialized resources required for intelligent network-provided services, such as dual-tone multifrequency (DTMF) receivers, announcements, conference bridges, and so on.

Finally, on the management side, three functional entities are defined:

- The *service management function* allows for deployment and provisioning of intelligent network services and for support of ongoing operations. Its management domain can cover billing and statistics as well as service data.
- The *service creation environment function* allows new services to be safely and rapidly defined, implemented, and tested before deployment.
- The *service management access function* provides the interface between service managers and the service management function.

It is envisioned that the service independent building blocks specified in the global functional plane will be realized in the distributed functional plane by a sequence of coordinated actions to be performed by various functional entities.

Physical Plane. The physical plane describes the physical alternatives for the implementation of an intelligent network. The identified possible physical nodes include *service control points*, *switches*, and *intelligent peripherals*.

The information flows between functional entities contained in separate physical entities imply a need to specify and to standardize the interfaces and protocols between these separate physical entities.

The following protocols have been defined by the ITU-T:

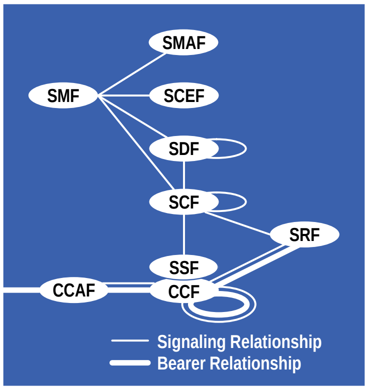
- The ISDN User Part (ISUP) and the Telephony User Part (TUP) are instantiations of the information flows between call control functions.
- The Intelligent Network Application Protocol (INAP) covers the information flows between service switching functions with a series of message sets.
- The information flow between the service control function and the service data function is based on the X.500 specifications.

Except for the TUP, these protocols are network services on top of the telephone companies' Signaling System #7 (SS7) signaling networks.

Intelligent Network Rollout

The general architectural concepts of intelligent networks are applicable to a wide range of telecommunications networks including plain old telephony services (POTS) networks, mobile communication networks (GSM, PCN, DECT), ISDN, and future broadband networks. Furthermore, these well-defined architectural principles can also be found in standards from other organizations that have defined equivalent or domain-specific architectures. BellCore's AIN (Advanced Intelligent Network) architecture shares many features of the ITU-T approach, while ETSI, for example, has identified various physical nodes (HLR, VLR, MSC) communicating via the MAP protocol in mobile networks.

Although it didn't deliver all of its original promises, the intelligent network concept is considered a success. Today freephone remains the major revenue-earning service for intelligent networks, and it is continuing to grow. Freephone, split-charge, and premium rate services still generate almost 50% of intelligent network revenue. Another service that is providing significant revenue to network operators is virtual private network service, which is currently experiencing the most rapid growth.



Functional Entities:
SMAF: Service Management Access Function
SCEF: Service Creation Environment Function
SMF: Service Management Function
SDF: Service Data Function
SCF: Service Creation Function
SRF: Specialized Resource Function
SSF: Service Switching Function
CCF: Call Control Function
CCAF: Call Control Access Function

Fig. 2. *Intelligent network distributed functional plane, showing functional entities.*

It is interesting that all of these services share the characteristic that they require data to be available throughout a network. This class of service clearly promotes a centralized intelligent network solution. In fact, the fundamental success of the intelligent network concept is that it has simplified data management of those services for which it has succeeded (although service management remains a relatively minor consideration in standards organizations). The full potential of other types of intelligent network services still needs to be realized.

Intelligent Network Element Requirements

Hewlett-Packard has developed the HP OpenCall service execution platform as an open, programmable, scalable, highly available, and easily manageable platform that can be used as a basis for implementing a range of different elements of an intelligent network. The platform provides a set of basic functionalities that are common to many intelligent network elements. By installing suitable user-defined applications or services on it, the HP OpenCall platform can be extended to provide the service control function, service data function, specialized resource function, and other functionality to other nodes of the SS7 network, such as switches. Thus, the aim of the HP OpenCall service execution platform is to provide a platform that can be easily extended to meet the requirements of different intelligent network elements. The common requirements of these different network elements are summarized in the following paragraphs.

Openness and Flexibility. One of the key goals of the intelligent network is to promote multivendor solutions to allow technology from different equipment providers to interwork. In contrast, many of the early intelligent network solutions were implemented on proprietary solutions. These applications are often not portable across platforms, so customers are often tied to a single equipment provider and cannot always benefit from the latest advances in hardware.

Furthermore, intelligent networks are seen as evolutions of existing networks, that is, new network elements, implementing intelligent network functionality, are expected to interwork with existing equipment. This implies that the new elements must support multiple entry points to ensure easy integration with other products, both at the SS7 network interface and at the interface with the operations support systems that manage the telephone network.

This need for multivendor solutions also drives the standardization activity in intelligent networks (see ***Subarticle 6a, "Standardization—A Phased Approach"***). In theory, if the interfaces between network elements are standardized and clearly defined, interworking should be easy. However, despite the existence of multiple standards, local differences make it necessary for a platform to adapt to many different environments in terms of connectivity, protocol support, and management. Furthermore, there is no standard environment in the telecommunications central office. Each central office often has its own collection of legacy systems. This implies a need to be able to add network-specific, protocol-specific, and environment-specific intelligence to meet customer requirements.

Rapid Service Deployment. In the increasingly competitive telecommunications market, network operators see the need to differentiate themselves on their service offerings. Operators want to be able to define, deploy, and customize services quickly, while still guaranteeing the high levels of quality and reliability that have traditionally been associated with telecommunications networks. There is a need to support all aspects of the service life cycle, including definition, validation, installation, and management.

Performance and Determinism. The round-trip real-time budget from a switch to a network element such as a service control point is typically quoted as 250 milliseconds (mean) with about 95% of responses within 500 ms. This includes the switch processing time, the transmission and queuing times of both the request and response messages, and the service control point processing time (encoding and decoding of messages, service activation and execution, query to database). Clearly, the faster the transmission links and the smaller the buffering in the system, the more time is available for service control point processing. For a simple freephone service, once the SS7 transmission times are subtracted, we obtain a requirement for a mean service control point processing time of 50 ms with 95% completing within 62 ms. The behavior must be controllable so that the system is deterministic. Transaction rates of up to 10,000 transactions per second for network elements have been requested.

High Availability. Network elements such as service control points and home location registers are critical components in an intelligent network. Very high system availability is required: no more than three minutes total downtime per year, including both scheduled and unscheduled downtime. This necessitates highly reliable hardware with no single point of failure and software that allows mated applications to back each other up in the event of a natural disaster disabling one particular site.

Furthermore, in the event of a failure at a site, during the failure recovery phase, the network element must be responsive to other network elements, taking less than six seconds to resume service. If not, the network considers that a total failure has occurred at the site.

The availability requirements also pervade the scalability and functional evolution aspects. The system must be capable of expansion and addition of new functionality without disruption of service.

Similar kinds of requirements apply to service management systems, albeit not as severe as for network elements. Service management systems are typically allowed 30 minutes of total downtime per year.

Scalability. A network element must be scalable in terms of processing power, memory, persistent data storage, and communications without compromising system availability. On the hardware side, this means support for online upgrades of processor boards, memory boards, disk subsystems, communication controllers, and links. It must be possible to perform such operations without service interruption. On the software side, it means bringing additional resources into service smoothly and safely. If anything goes wrong, the software must automatically fall back to the last operational configuration.

In general, network elements such as service control points and home location registers are classified in terms of transactions per second (TPS) and customer database size. Scalability then translates into the ability to add new hardware and/or software elements to increase either the maximum supported TPS rate or the maximum customer database size.

Functional Evolution. The ability to add new applications or platform capabilities or simply upgrade existing ones without impacting the availability of the system is vital. This means that such things as updating the operating system, upgrading the application, adding a new communication protocol stack, or changing the firmware on a communication controller must be achieved without disrupting real-time performance or system availability. This ability should not impose too many constraints on software development and installation. In all upgrade situations, a fallback must be possible.

Manageability and Operation Requirements. There are detailed and rigorous requirements concerning installability, physical characteristics, safety, electromagnetic and electrical environments, maintenance, reliability, and so on. Remote management interfaces to large and complex network systems with demanding performance and scalability requirements are needed.

To allow easy management, complex distributed or replicated network elements must be capable of providing a single-system view to operations centers. At the same time, it is also very important to provide per-system views, to allow management of specific systems and to act upon them (typically for fault management or performance management). Newly installed network elements often need to be integrated into existing management systems.

HP OpenCall Platforms

HP OpenCall Service Execution Platform

The HP OpenCall service execution platform is an open, scalable, programmable, highly available, and easily manageable platform. It is implemented as a layer of functionality on top of the HP OpenCall SS7 platform (see ***Article 7***). Given the general network element requirements listed above, the following architectural principles and high-level design decisions were adopted when developing the HP OpenCall service execution platform.

The software runs on the HP-UX* operating system, allowing it to benefit immediately from advances in hardware speed and CPU processing power.

All critical hardware and software components are replicated, giving the platform the ability to tolerate any single failure. An active/standby replication model was chosen at the software level, with an instance of the platform software running on two independent systems. Besides providing a high degree of fault tolerance, such an approach also provides the basis for most online upgradability tasks, such as upgrading hardware and the operating system.

Fig. 3 shows a typical site configuration, with two instances of the HP OpenCall service execution platform software executing on two independent HP 9000 machines, but with a single connection to the SS7 network. This site will appear as a single network node to other elements of the SS7 network. The configuration in Fig. 3 is a standard duplex configuration of the platform. Other configurations are possible, such as the mated-pair configuration, described later. In the standard duplex configuration, the two machines are connected via a dual LAN and both hosts are connected to a set of signaling interface units. The HP OpenCall service execution platform software runs on both hosts in active and standby modes. The active host controls the signaling interface units and responds to all incoming requests from the SS7 network. Both hosts are capable of being active (i.e., there is no default active host).

The platform is network independent. It makes no assumption about the structure of the SS7 network. It has the ability to support multiple message sets, but all message set dependent decisions are made at the application level.

Some customization is necessary before an instance of the HP OpenCall service execution platform software can be installed in or connected to an SS7 network. By default, the platform supports no message set, and it offers no service to other network elements. Minimally, users of the platform must install one or more message sets and provide one or more applications to respond to requests coming from other network elements or to send requests to other network elements. Furthermore, to ensure that the resulting solution can be monitored and managed, it should be integrated into an existing management system or a set of management tools should be implemented.

A set of APIs (application programming interfaces) are provided to access platform functionality, to be used either locally or remotely. This provides flexibility with respect to integration with operations support systems and legacy management systems. A set of management tools using these APIs are also provided, and can be used to provide a first level of platform management. Further levels of management (for managing the platform, installed services, customer data, etc.) can be provided by integrating the platform with other external management systems via the provided APIs.

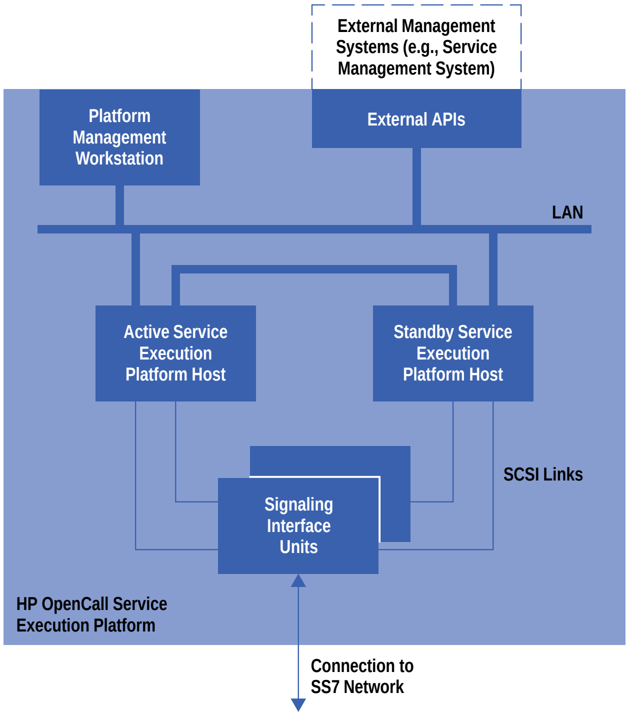

External Management Systems (e.g., Service Management System)
Platform Management Workstation
External APIs
LAN
Active Service Execution Platform Host
Standby Service Execution Platform Host
SCSI Links
Signaling Interface Units
HP OpenCall Service Execution Platform
Connection to SS7 Network

Fig. 3. *Duplex configuration of the HP OpenCall service execution platform. Two instances of the platform execute on two independent HP 9000 host machines with a single connection to the SS7 network.*

Applications, or *services*, executing on the platform are interpreted. New services or new versions of existing services can be installed at run time without interrupting processing of TCAP (Transaction Capabilities Application Part) traffic and without affecting other running services. Because services execute in a virtual machine with no direct access to the operating system, services cannot affect the availability of the platform. Furthermore, the service execution environment can monitor service instances, ensuring that instances do not interfere and that resources are not permanently consumed.

Services are independent of the operating system, protecting them from changes and upgrades to operating system. No knowledge of the operating system is required to write the service. Services are written in SLEL (Service Logic Execution Language). Most of the basic concepts in SLEL have been adapted from SDL (Specification and Description Language), enhancing it with some features specific to intelligent networks. This has the advantage that SDL is well-known in the telecommunications industry, and many telecommunications standards are specified in SDL.

A replicated in-memory relational database is provided as part of the service execution environment. The structure and contents of this database are under the user's control. By holding all customer-related data in RAM, services can respect the real-time response time requirements imposed by switches, since there is no disk access to retrieve call-related data. To achieve data persistency, a copy of the database is maintained on a standby host.

A Management Information Base (MIB) collects information on the state of the platform, making this information available both via an API and via a set of management tools, and allows external applications to manage and configure the platform. All management operations are directed to the active system—the standby system replays all management commands—thus presenting a single-system view to external applications.

The platform is implemented as a set of UNIX® operating system processes, allowing it to profit from multiprocessor hardware.

HP OpenCall Service Creation Environment

The HP OpenCall service creation environment allows easy definition, validation, and testing of services. Services are defined as finite-state machines, using a graphical language. The service creation environment provides a set of tools to allow the validation and simulation of services before they are deployed on the HP OpenCall service execution platform. The same service execution environment as described above exists in the service creation environment to ensure that the same behavior is observed when the service is installed in the HP OpenCall service execution platform.

HP OpenCall Service Management Platform

The HP OpenCall service management platform is capable of managing multiple HP OpenCall service execution platform sites. Such a distributed configuration introduces an extra degree of scalability, both in terms of transaction throughput capacity and database capacity.

Platform Design

This section discusses five key aspects of the HP OpenCall service execution platform solution: the service execution environment, platform flexibility, high availability, database replication, and scalability.

Service Execution Environment

The HP OpenCall service execution platform provides an execution environment for telecommunications services. These services are usually developed in response to new telecommunications requirements, and typically provide additional functionality to other elements in the SS7 network.

Programs defined in the Service Logic Execution Language (SLEL) are interpreted by the platform at run time, and can use language primitives to access the functionality of the underlying platform. The primitives enable them to send and receive TCAP messages, read and write message attributes, access and update the in-memory database, communicate over other external connections, send and receive events, set and reset timers, manage shared resources, log data to disk, and perform other functions.

Programs written in SLEL define finite-state machines, that is, a service waits in any given state until a recognizable input stimulates a transition to the next state. During the transition the service can carry out processing and output messages. Fig. 4 summarizes the functionality of the SLEL virtual machine.

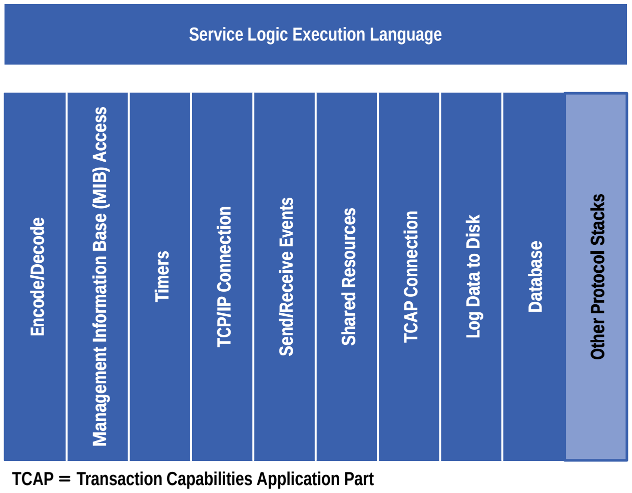


Fig. 4. *Services are written as finite-state machines in the Service Logic Execution Language (SLEL) and run on the SLEL virtual machine, which provides the functionality shown here.*

The following principles have been adopted in the development of the HP OpenCall service execution environment:

- Services are not aware of the operating system. They are isolated from the HP-UX interface. This allows easy migration between HP-UX versions. Furthermore, the application developer only needs to provide service-specific logic and need not be concerned with the startup, switchover, and failure recovery aspects of the platform, since these are transparent to the service logic.
- Services are not aware of replication. Replication of services and restart of services after a failure are handled automatically by the platform. No failure recovery code needs to be written by the service developer. The service programmer can assume a single-system view. To maintain this illusion, services can only access the local MIB and can only receive locally generated events. Furthermore, the service execution environment on the standby node is an exact replica of the active node's, providing the same services, same resources, same MIB structure, and so on.
- Real-time response to switches. The service execution environment gives the highest priority to the processing of TCAP messages and other service-related events (e.g., popped timers, received events). All other activities such as database or MIB accesses by external applications are run as background tasks. Service execution cannot be interrupted. A single state transition runs to completion. All local access by a service is synchronous (even to the database). A service only blocks if it explicitly requests blocking, for example to wait for the next TCAP message, wait for a timer, or wait for an event.
- Services cannot crash the platform. The service execution environment presents a virtual machine as its upper interface. Services can only access the platform functionality from SLEL. No pointer manipulation is available. Resource allocation and deallocation are done automatically by the platform. There is no possibility for core dumps or memory leaks.

- Online upgradability of services is possible. Because services are interpreted, services can be enabled, disabled, installed, or removed at run time without stopping the platform. Multiple versions of a service can be installed simultaneously, although only one can be enabled at any time. Instances of the previously enabled version are allowed to run to completion, so that TCAP traffic is not interrupted.
- The platform manages and monitors service instances. A single service cannot block other services from executing. There is a limit on the number of instructions that can be executed before a service is terminated. This prevents infinite loops and prevents one service from blocking out all other services. Resources are automatically reclaimed once the service instance has completed. There is also a limit on the total lifetime of a service, as a way of garbage collecting unwanted service instances. Both limits can be set on a per-service basis, and can be altered at run time.

Platform Flexibility

There is a strong requirement for the HP OpenCall service execution platform to be flexible. Obviously, it should make as few assumptions as possible about the applications that are installed and executing on it, and it should be able to integrate into any central office environment, both in terms of its connection with the SS7 network and its links with management applications.

Multiple Message Sets. The platform by default supports both a TCAP/SS7 connection and a TCP/IP connection. It makes no assumption about the protocols that are used above these well-standardized layers. In practice, any message set defined in ASN.1 (Abstract Syntax Notation One) can be loaded into the platform, and multiple message sets can be supported. The platform can encode and decode messages belonging to one of the installed message sets.

The message set customization tools take as input an annotated ASN.1 definition of a message set. The output is a *message set information base*, which contains a concise definition of the message set. Multiple message set definitions can be stored in a single message set information base. The HP OpenCall service execution platform's service execution environment loads the message set definitions from the message set information base at startup time. Services running in the execution environment can request the platform's encode/decode engine to process any incoming or outgoing message with respect to the installed message sets.

The same message set information base can be loaded into the HP OpenCall service creation environment. The message set definitions are available to service developers as part of the help facility. The message set information base is also available in the validation environment, allowing the user to submit message-set-specific message sequences to test the logic flow of the developed services. The traffic simulation tool uses the message set information base to encode and decode the user-supplied message sequences.

Flexible Service Instantiation. When a new TCAP transaction request is received by the platform, service instances must be created and executed to process the message. To offer a high degree of flexibility, the policy for choosing the correct service instances is programmable, that is, a user-supplied piece of service logic is executed to choose the most appropriate service. The decision can be based on any criterion, such as priority of the request, customer-specific data held in the database, overload status of the platform, and so on.

To allow tracking of resources, only one service instance controls the TCAP transaction, although it can be passed between instances. In this way, if the instance that currently controls a transaction exits unexpectedly, the platform can close the associated transactions and free the associated resources.

Flexible Service Structure. The intelligent network market has traditionally taken a service independent building block-based approach to service implementation. That is, a set of service independent building blocks are provided by the underlying execution platform, allowing applications to merely link these building blocks together to provide services to the SS7 network. The disadvantage of this approach is that the only functionality available to programmers is the set of available service independent building blocks. These are often message-set-specific and difficult to customize.

The HP OpenCall service execution platform does not provide a default set of service independent building blocks. Instead, it provides the means to structure applications as a set of components. Thus, if required, a user can implement the ITU-defined set of standard service independent building blocks and then use those components to implement applications to provide higher-level services. Of course, the user can also decide to implement an entirely different set of service independent building blocks.

Furthermore, the platform does not distinguish between a service independent building block (or component) and an application. Instead, it views both as services. Both can be arbitrarily complex pieces of logic, defined and installed by the user. How they interact and what functionality they provide are entirely under the user's control. To provide a single service to the SS7 network might only involve a single instance of service logic, or it might involve the creation and interworking of multiple such instances.

Platform Management. The platform exports information on its state via a Management Information Base (MIB). The MIB can be used to both monitor and control the platform. For example, installing a new service or a new version of an existing service onto the platform is performed via the MIB. Similarly, adding a new TCP/IP connection or supporting a new subsystem number is also achieved via the MIB. Both cases involve creating a new object in the MIB. Statistics on CPU use, TCAP traffic, service execution, database memory use, and so on are all held in the MIB and can be retrieved by external applications by issuing a request to the appropriate objects.

The information is presented as a hierarchy of objects, with each object representing a part of the platform's functionality. The hierarchical structure provides an intuitive naming scheme, and this also allows easy integration into standard CMIS (Common Management Information Service) or SNMP (Simple Network Management Protocol) management frameworks. Users can create new objects, delete existing objects (obviously changing the functionality of the platform in the process), update existing objects, or just retrieve information from individual objects. As mentioned previously, APIs that can be used remotely are provided, along with a set of management tools that provide a first level of platform management.

Customized Overload Policy. One of the most important responsibilities of any SS7 network element is to respond in a timely manner to incoming requests. The response time on requests must appear to be bounded, and 99% of replies must be generated within a fixed time interval. This implies that the network element must react to overload situations.

With the HP OpenCall service execution platform, the overload policy is programmable. That is, user-defined logic specifies how the network element reacts when the load is high. The platform itself collects statistics on a number of overload indicators such as CPU use, transaction rate, number of unprocessed messages, and average queuing time for such messages. These values are available in the MIB and can be viewed both by the overload service (the logic implementing the overload policy) and by external management applications.

The programmability of the overload policy offers a high level of flexibility. For example, under heavy load, the overload service may decide to:

- Reject new incoming requests but continue to accept messages relating to ongoing transactions.
- Request that other network elements reduce the number of requests. Obviously such a policy is network-specific and message-set-specific, requiring knowledge of both the SS7 network configuration and the message sets supported by remote switches.
- Reject new requests that require complex processing (obviously application-specific), or reject requests for low-revenue-generating services (again, application-specific).

The platform also provides a set of hooks to control the traffic flow. The overload policy can, for example, request the platform to reject new transaction requests (i.e., only accept messages relating to ongoing transactions), to limit the number of instances of a given service, or to reject traffic from a particular remote network element.

High Availability

All critical components of the HP OpenCall service execution platform are replicated (see Fig. 5). The core set of software processes operate in active/standby mode. This forms the basis both of the platform fault tolerance and of its online upgradability policy. It uses the HP OpenCall SS7 high availability platform, with every critical process being a client of the fault tolerance controller (see ***Article 8***). For simplicity, not all of the processes are shown, and not all of the interprocess links are shown.

Besides replication of processes, the platform also uses mirrored disks, duplicate signaling interface units, and dual LAN connections.

The principles that form the basis for the high availability policy are discussed in the following paragraphs.

Active/Standby Model. An instance of the HP OpenCall service execution platform platform runs on each of two independent machines. One instance, the *active*, is responsible for responding to all incoming requests, whether from the SS7 network or from management applications. The other instance, the *standby*, maintains a copy of the active's state, always ready to take over processing of such requests. The decision to adopt an active/standby model brings the following benefits:

- The code is simpler, resulting in fewer run-time errors.
- It is easy to provide a single-system view (that is, the active instance defines the state of the platform).
- It isolates errors because the two hosts are not performing the same tasks or the same types of tasks.
- The standby host is available for online upgrades, configuration changes, and so on.

The alternative, to adopt a load-sharing model (with requests being processed in parallel on two or more hosts), would have required more complex communication protocols between the instances (to ensure synchronization and a single-system view) as well as a greater possibility of an error simultaneously impacting more than one host.

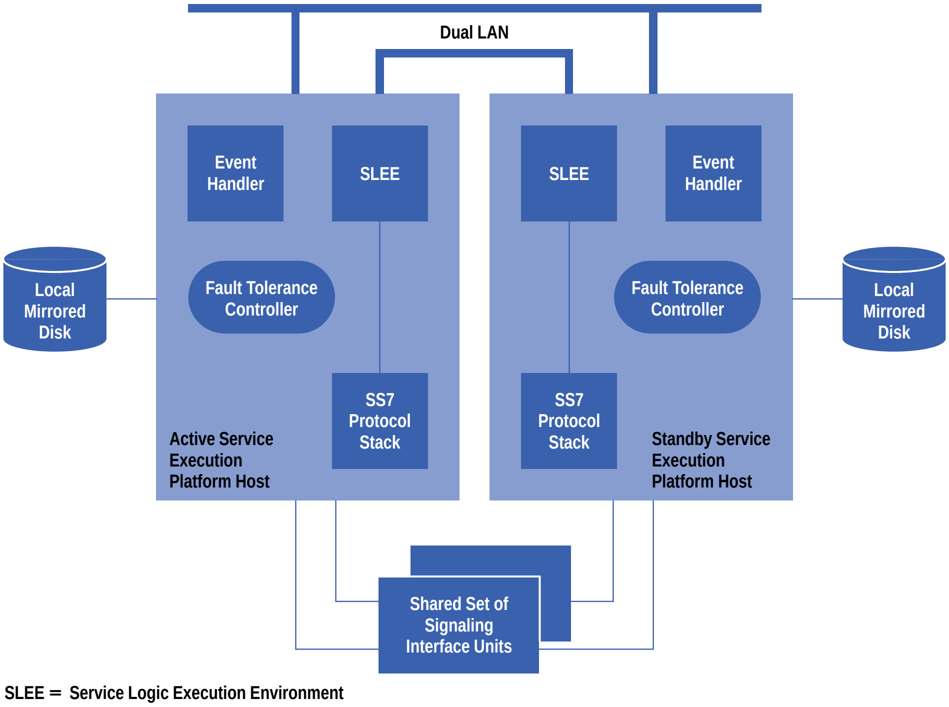

Dual LAN
Event Handler
SLEE
SLEE
Event Handler
Local Mirrored Disk
Fault Tolerance Controller
Fault Tolerance Controller
Local Mirrored Disk
SS7 Protocol Stack
SS7 Protocol Stack
Active Service Execution Platform Host
Standby Service Execution Platform Host
Shared Set of Signaling Interface Units
SLEE = Service Logic Execution Environment

Fig. 5. *Replication of critical components in the HP OpenCall service execution platform.*

High Service Availability. The solution guarantees high service availability to the SS7 network in the event of any single hardware or software failure. To achieve this, the standby host must always be ready to take over in the event of a failure on the active host. For this reason, the standby maintains a copy of the in-memory database and of the service execution environment. All relevant changes to the state of the active (e.g., changes to the active database) are immediately propagated to the standby. In the event of a switchover, the state of the standby instance defines the current state. That is, the standby does not need to retrieve information from either the failed instance or from other external applications to become active. Thus, the transition to the active state is instantaneous, guaranteeing high service availability.

Centralized Fault Recovery Decisions. All switchover and process restart decisions are made by a centralized process, the *fault tolerance controller,* on each node. These two processes continuously exchange information on the state of the two hosts. All other processes obey the centralized decision-maker. This greatly simplifies failure recovery and error handling. In the event that both LANs go down, the signaling interface units are used as a tiebreaker. The node that controls the signaling interface units remains active (see ***Article 8***).

Online Upgradability and Maintenance. Because all of the critical components are replicated, the existence of the standby host is used as a basis for all online upgradability operations, such as changing the operating system, installing new versions of the platform, reconfiguring services or the service execution environment, performing rollback operations on the database, and so on. Because most upgrade operations can effectively be performed online in this way, the platform meets its downtime-per-year requirements.

Replication Does not Impact Services. Service programmers do not need to be aware of the replication of the platform. Furthermore, propagation of data to the standby is performed as a background task, implying a minimal impact on response time. Algorithms for resynchronizing the standby after a failure are also run in the background.

Respawnability. It is important that a failed host or a failed process be restarted quickly. If the standby host is not available, this introduces a window of vulnerability during which a second failure could cause the whole network element to be unavailable, directly impacting service availability. This respawnability feature is possible because of the continuous availability of an active system. The failed instance will restart, rebuilding itself as a copy of its peer. Because the active instance is assumed to be consistent, the respawned instance is also guaranteed to be consistent. However, to avoid rebuilding an instance to a nonconsistent state, instances are not respawned if the peer is not active (which might happen in the rare case of a double failure). In such cases, manual intervention is required to restart the complete platform. This respawn policy ensures rapid failure recovery in the case of a single failure but prevents erroneous failure recovery in the case of a double failure.

Support for Complete Restart. Although the platform is designed to tolerate any single failure, in certain cases it may be necessary to stop and restart both hosts (either because of a double failure or for operational reasons). Disk-based copies of both the MIB and the in-memory database are maintained, and these can be used to rebuild both hosts. However, some data loss may occur. This is considered to be acceptable, since double failures are rare.

Database Replication

At the core of the service execution environment is an in-memory replicated database. The database is in memory to guarantee rapid access and update times. To ensure high availability, copies of the in-memory database are kept on both the active host and the standby host. Critical updates to the active database are propagated to the standby.

The structure and contents of this database are under the user's control. When defining the database structure, the user must also define the replication policy for every field. Of course, propagating updates to the standby will impact performance, but because the user is allowed to specify the replication policy, the user can also control the impact.

Traditionally, database systems achieve persistency by maintaining a copy of the database on disk or on mirrored disks. In the HP OpenCall service execution platform, the primary standby copy is held in memory on a standby host. This offers a number of advantages over the traditional approach:

- Writing to a remote in-memory copy is quicker than logging to disk, and therefore has a smaller impact on SS7 traffic.
- The degree of availability is higher. In the event of a failure on the active host, the standby copy is immediately available. It is not necessary to recreate a copy of the database.
- The standby copy can be taken offline and used to restructure the database or to roll back to a previous version of the database.

Periodically, the active host generates a disk-based copy of the database. This *checkpoint* of the database serves a number of purposes:

- The checkpoint ensures that the platform can recover from double failures.
- The checkpoint is used to reinitialize a standby host if and when it is restarted after a failure or shutdown.
- The checkpoint can be used for auditing purposes by external database management systems.

Three algorithms are critical to the database replication scheme: the data replication algorithm, the resynchronization algorithm, and the rollback/restore algorithm.

Data Replication Algorithm. As mentioned above, the user specifies the replication policy for every field in the database. For certain data in the database, it may not be necessary to replicate changes to the standby host. Typical examples are counters or flags used by services.

Consider a set of fields that collect statistics on SS7 traffic. These would typically be incremented every time that a new request is received, and would be reset to default values periodically. For many services, it may be acceptable not to propagate these traffic statistics to the standby database. This implies that some data will be lost in the event of a failure on the active host (all traffic statistics since the last reset), but it also implies that less CPU time is consumed replicating this data to the standby. This trade-off is application-specific, so the decision to replicate is left to the user.

For fields that are marked for replication, the active database instance will generate an update record called an *external update notification*, containing both the old value and the new value, for every update. This record is then propagated to the standby database instance. By comparing the old value with the new value, the standby database can verify that it is consistent with the active. In the case of an inconsistency, the standby database shuts itself down. It can then be restarted, rebuilding itself as a copy of the active database.

The flow of an external update notification is shown Fig. 6.

To handle double failures, the external update notification is also written to disk on both the active and standby hosts. This is performed as a background task on the active host to minimize the impact on the transaction processing activity. The active host does not attempt to maintain a replica copy of the database on disk. Instead, a log is maintained of all of the updates that have been performed. To rebuild the in-memory database, the log must be replayed.

To manage the disk space required for these external update notification logs, the active host takes regular checkpoints. This task is treated as a low-priority background activity, and the frequency of the checkpoints can be controlled by the user (obviously as a function of the available disk space). When a checkpoint operation has completed, redundant external update notification log files can be removed. By default, the active host keeps the two most recent checkpoint files and intermediate external update notification log files on disk (following the principle of replicating all critical components).

Because database updates must be allowed during the checkpointing operation, each checkpoint file has an associated sequence of external update notifications that were generated while the checkpoint was being taken. To rebuild a copy of the in-memory database, both the checkpoint file and the associated external update notifications are required. Typically, the contents of the disk containing these database-related files will be as depicted in Fig. 7. The external update notifications are stored in a sequence of files. The rate at which one file is closed and a new file in the sequence is opened can be controlled by the user (either as a function of time or file size, or on demand). Thus, multiple external update notification log files can be created between two checkpoints, or indeed during a checkpoint operation. A copy of the in-memory database can be rebuilt either from a checkpoint file and associated external update notification log files (those files generated during the checkpoint) or from a checkpoint file, associated external update notification log files, and subsequent external update notification log files. To hold a copy of the database on a remote storage device, the user should close the current external

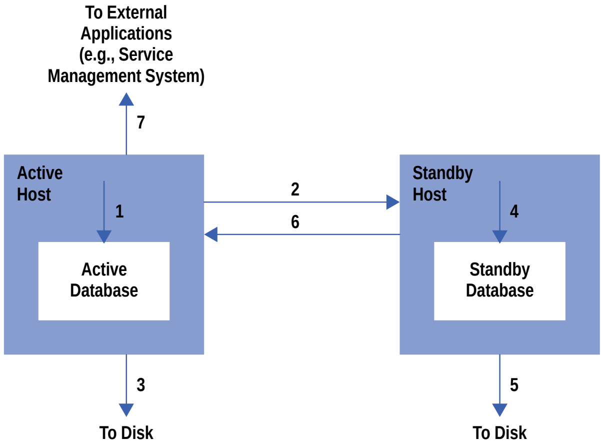


Fig. 6. *External update notification flow. (1) Update performed on active database. (2) Update record (external update notification) sent to standby. (3) External update notification logged to disk on active host. (4) Update performed on standby database. (5) External update notification logged to disk on standby host. (6) Acknowledgment sent to active host. (7) External update notification forwarded to interested external applications.*

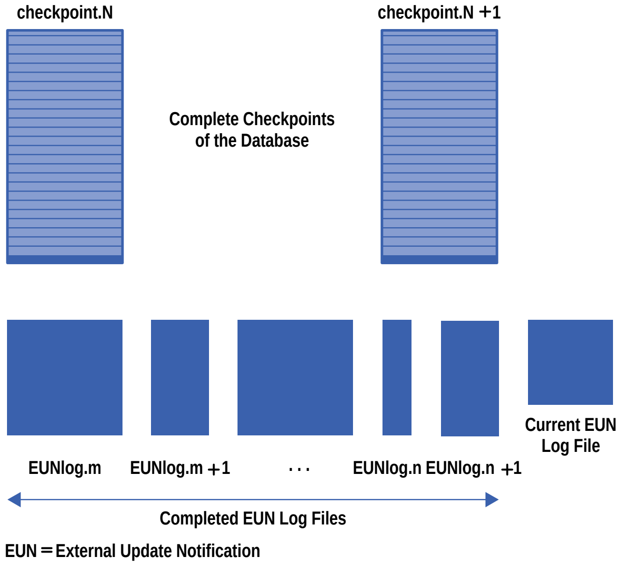


Fig. 7. *Checkpoint and external update notification files on disk.*

update notification log file (the active database will close the current file, assign it a file name in the sequence, and open a new current external update notification log file), and the user should then copy to stable storage the most recent checkpoint file and the external update notification log files generated during and since that operation.

Resynchronization Algorithm. Failures will happen. Therefore, a failure recovery algorithm is required. One critical component is the recovery of the standby copy of the in-memory database. The checkpoint and external update notification log files also play a critical role in this algorithm. The algorithm is complex because updates are permitted on the active database while the standby database is being rebuilt. The resynchronization process can take a long time (in the order of minutes), so it would not be acceptable to disallow database updates during that period.

The sequence of steps in resynchronization is as follows:

1. The standby host copies the most recent database checkpoint file and external update notification log files from the active file system to its local file system.

2. The standby host rebuilds the in-memory copy of the database from this consistent set of files.

3. When this operation is complete, the standby database asks the active database to close the current external update notification log file. It then retrieves that file and replays the external update notification records.

4. Step 3 is then repeated. The assumption is that, because the standby host is not performing any other tasks, with each iteration of step 3, the size of the current external update notification log file will be reduced. In effect, the state of the

standby database is moving closer to that of the active database. Eventually the size of the current external update notification log file will be small enough for the algorithm to move to the next step.

5. The standby database again asks the active database to close the current external update notification log file. At this point, a connection is also established between the two database copies. Now, while retrieving and processing the latest external update notification log file, the standby database is also receiving new external update notifications via that connection.

6. Once the latest external update notification log file is processed, the standby database starts to replay the queued external update notifications. At this point, with the establishment of the real-time connection between the two database copies, the two copies are considered to be synchronized, and the standby database is considered to be *hot standby*.

Rollback/Restore Algorithm. As with all database systems, it is also possible to roll the in-memory database back to a previous state. Again, the checkpoint and external update notification log files play a critical role in this algorithm.

This operation can be achieved “online” by using the standby copy of the database, which can be taken offline without impacting the active host. While the latter continues to process TCAP traffic, the rollback algorithm can be applied to the standby database. In effect, it rebuilds itself from a checkpoint file and associated external update notification log files. Once this operation is completed, the user can request a switchover of the two hosts. The previously offline standby host will now begin to receive and to process TCAP traffic. The peer host should then be rebuilt as a copy of this new active host, using the synchronization algorithm described above.

Scalability

The HP OpenCall service execution platform is implemented as a set of processes running on the HP-UX operating system. This gives it a degree of scalability, since it can run on a range of HP machines and can also benefit from multiprocessor hardware. Low-end configurations are executed on a single-processor, 48-MHz machine with 128M bytes of RAM. The high-end configurations currently execute on a dual-processor, 96-MHz machine with 768M bytes of RAM. The migration to HP-UX 10.20 increases the capacity of the platform both in terms of the maximum supported TPS rate and the maximum database size.

An extra degree of scalability is provided with the support of *mated-pair* configurations. Fig. 3 showed the standard single-site duplex configuration, that is, an active/standby configuration running at one geographical location with a single connection to the SS7 network (multiple links are used for redundancy and load sharing, but a single address is shared by these links). A distributed solution, with multiple active/standby configurations running at a set of sites (each with its own SS7 address) provides a number of benefits:

- Extra TPS capacity, since sites can process traffic in parallel
- Increased database capacity
- Tolerance of site failures or outages. This is critical if and when it is necessary to shut down a complete site for maintenance or operational reasons (or in the unlikely case of a double failure hitting both the active and standby hosts).

The HP OpenCall service management platform can be used to manage a distributed configuration, as shown in Fig. 8. Although this is referred to as a mated-pair configuration, it is not limited to two sites; multiple sites can be supported. Furthermore, each site is not required to have the same database, that is, the contents of the databases at different sites can be completely different. However, in general, sites will be paired; hence the name mated-pair. For a given site, a copy of its database will also be maintained at one other, geographically remote site. This remote site then has the ability to take over if the original site is shut down. The role of the HP OpenCall service management platform is to maintain consistency across these multiple sites.

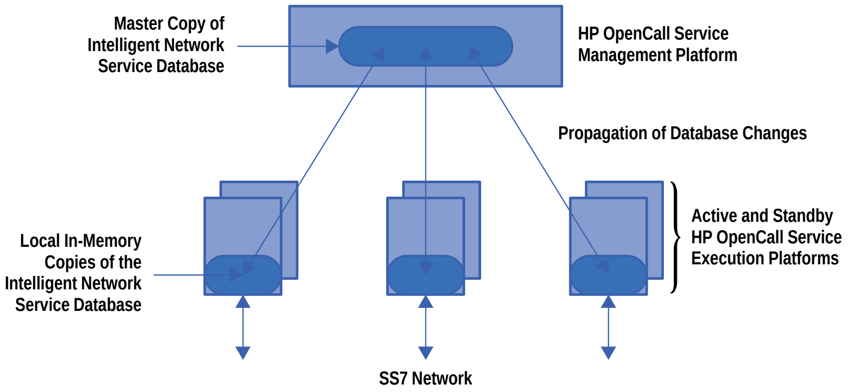


Fig. 8. *Centralized management of multiple HP OpenCall service execution platform nodes.*

The centralized HP OpenCall service management platform maintains a disk-based copy of each in-memory database. It receives notifications (external update notifications) from each site when the in-memory database is changed. Such external update notifications are then propagated to all other sites containing the altered data.

If an operator or system administrator wishes to change the contents of the in-memory database, the request should be sent to the service management platform. It will then forward the request to all sites holding a copy of the altered data.

Sites will typically process TCAP traffic in parallel, so it is possible that the same data may be changed simultaneously at two separate sites. Both will generate external update notifications and both external update notifications will be propagated to the service management platform. The platform is responsible for detecting this conflict and for ensuring the consistency of the replicas. It uses the old value field of the external update notification to detect that simultaneous updates have occurred. Consistency is ensured by rejecting any external update notification in which the old value does not match the current value held by the service management platform, and by applying the current value at the site from which the erroneous external update notification was received. In effect, this establishes the service management platform's database as the master database. In the event of a discrepancy, the service management platform ensures that its view is enforced and that all copies will become consistent.

The HP OpenCall service management platform also supports an auditing function. The service management platform orders a checkpoint of the HP OpenCall service execution platform's in-memory database, and then compares the contents of the resulting checkpoint and external update notification log files with the contents of its own disk-based database. Discrepancies are reported to the system administrator.

Conclusion

The intelligent network architecture allows the telecommunications industry to move to a more open solution, in which the creation, deployment, and modification of services is independent of a particular network equipment supplier and equipment from different providers can interwork to provide new and innovative services. Having independent and neutral intelligent network platform providers can enforce implementation of standards, hence ensuring better interconnectivity.

The HP OpenCall service execution platform is an open and flexible platform. It has been installed in a large number of different networks throughout the world, and has shown its ability to interwork with equipment from a host of other network equipment providers. It has been used to implement mission-critical network elements such as service control points and service data functions using standard, off-the-shelf computer technology. This use of standard hardware and a standard operating system will allow operators to benefit from the evolution of information technology with few additional costs and limited engineering risk.

HP-UX 9.* and 10.0 for HP 9000 Series 700 and 800 computers are X/Open Company UNIX 93 branded products.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

X/Open is a registered trademark and the X device is a trademark of X/Open Company Limited in the UK and other countries.

Standardization—A Phased Approach

Because of the complexity of intelligent networks, the number of unresolved technical issues, and the significant financial investments, the development of an intelligent network architecture supporting all possible telecommunications services and technologies, called the *target intelligent network*, will take many years. Standardization bodies have chosen to adopt a phased approach to intelligent network development that takes advantage of the technological capabilities at a given point in time and that guarantees backward compatibility between the different phases.

The International Telecommunications Union—Telecommunications Standardization Sector (ITU-T) has addressed this phased approach in its Recommendation Q.1211.

Table I shows the different phases in terms of capability sets and their descriptions. Each capability set gives a set of definitions of capabilities that are of direct use to both manufacturers and network operators.

Table I
Phased Approach to the Target Intelligent Network

Phase (Capability Set)	ITU-T Recommendation	Time Frame	Description
CS1	Q.121x	Finalized in 1995	First standardized stage
CS2	Q.122x	To be finalized in 1997	CS1-compatible. Handling of multiparty calls.
CS3	Q.123x	Work started at end of 1996	CS2-compatible. Handling of broadband aspects and integration with the TMN.
CSn	...	...	Evolving towards target intelligent network

TMN = Telecommunications Management Network.

Capability Set 1 (CS1)

CS1 is the first standardized stage of intelligent network evolution based upon the existing technology. It is a subset of the target intelligent network architecture. CS1 defines the functional entities (see Fig. 2 in ***Article 6***) and the interface between these entities. It also defines the generic model of two-party call processing functionality, the Basic Call State Model (BCSM). CS1 limits end-user access to service processing capabilities to the following types: analog lines, ISDN basic and primary rate interface (BRI/PRI), and analog and SS7 trunks.

The target set of services for CS1 includes universal personal telecommunication (UPT), freephone, virtual private network (VPN), credit card calling, user-defined routing, and others. All of these services are considered immediately marketable and highly profitable. The common characteristic of all CS1 services is that they apply only to one party of the call (either the originating or the terminating party), and generally only during the call setup phase.

The protocol used by the different CS1 functional entities to communicate is called the Intelligent Network Application Protocol (INAP). This protocol relies on existing underlying transport protocols (e.g., SS7/TCAP) to convey the intelligent network application layer protocol messages.

Capability Set 2 (CS2)

CS2 is the second standardization stage and is a superset of the CS1 recommendations. CS2 aims to support enhanced services in addition to the ones supported by CS1. It introduces new capabilities that allow handling of multiple parties that are or will be involved in the same call, such as conference calling. Other capabilities will be included in CS2 to support personal mobility (UPT) and terminal mobility (DECT, GSM) functionality. These new enhancements and capabilities are achievable by extending the existing CS1 call-processing model and functional model. INAP operations are also extended and new ones are to be defined. Standardization activities are going on at ITU-T and ETSI (European Telecommunications Standards Institute). A complete revision of the CS2 protocol is expected at the end of 1997.

The target set of services for CS2 includes call completion to busy subscriber, conference calling, call transfer, call waiting, mobility services (UPT, GSM), and others. The common characteristic of all these services is that they require call party handling functionality that is not supported in CS1.

Future Capability Sets

CS1 and CS2 do not cover all possible user accesses and network capabilities. According to the phased approach, ITU-T plans to introduce CS3 (and maybe others later) to cover broadband network aspects (intelligent network/B-ISDN integration), intelligent network/TMN integration, and full support of mobile communications systems. Requirements are being set up and specifications might come in 1998.

HP Approach

Because the needs, in terms of network operation, vary from one network operator to another (operator-specific charging and billing, implementation limitations), and because the INAP standard will continue to evolve following the different capability sets, network equipment providers have to work with a large number of INAP variants.

To meet this need and to be able to respond to its customers' requirements rapidly, HP has developed a flexible service execution platform (see the accompanying article) that is able to rapidly follow the evolution of the INAP and support different customers' specific variants. Tools are provided to automate the support of a message set's syntax. The implementation of the message set's semantics has been pushed to the application level, leaving the platform itself independent of any supported message set. This has the benefit that a single platform can be maintained for a varied and evolving customer base.This independence from the INAP message set also allows the HP platform to easily support similar message sets defined by other standards bodies such as MAP (defined by ETSI for mobile networks) and AIN 0.1 and 0.2 (defined by BellCore).

The HP OpenCall SS7 Platform

The HP OpenCall SS7 platform allows users to build computer-based signaling applications connected to the SS7 signaling network.

by Denis Pierrot and Jean-Pierre Allegre

Today's telecommunications operators need to offer more and more services to their customers. Because of deregulation and the resulting competition, network operators have to be able to bring to the market useful value-added services to differentiate themselves from the competition. To support new functionalities, telecommunications networks have undergone an important restructuring starting in the mid-1980s. This restructuring resulted in the separation of the signaling functions from the voice transmission functions.

Signaling includes all of the necessary procedures to set up, tear down, and control calls. Before this split was made, the networks were using inband signaling—the signaling information was conveyed over the same channel as the voice with some predefined tones (see Fig. 1a). This technique had many drawbacks, including:

- Long call setup times. Addressing information needed to be outpulsed one digit at a time for each intermediate switch in the voice path.
- Security problems. Billing fraud was possible by faking the inband signaling and billing tones.
- Limitations on the amount of new services that could be provided.

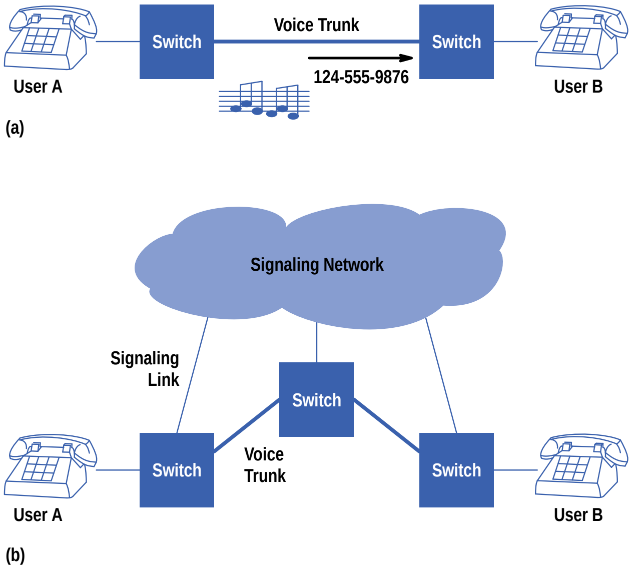


Fig. 1. *(a) Before signaling was separated from voice transmission, networks used inband signaling—the signaling information was conveyed over the same channel as the voice. (b) After the split, all of the connection setup, teardown, and control is effected via the signaling network and the voice trunks are dedicated to transporting voice only.*

The Signaling Network

With the separation of signaling and voice transmission, the concept of the *signaling network* was introduced. The signaling network is a digital, robust, packet network with built-in redundancy to achieve a high degree of availability. Fig. 1b shows the typical topology. All of the connection setup, teardown, and control is effected via the signaling network and the voice trunks are dedicated to transporting voice only.

The creation of the signaling network, often called *common channel signaling* or CCS, makes it possible to implement an important set of new services because of the global control it provides over the transmission network. The current implementation of the signaling network is called Signaling System #7, or SS7.

The signaling network is the foundation for the *intelligent network*, which makes it possible to deliver new services to network operators' customers in a timely and cost-effective manner. The intelligent network is programmable so that new services can be easily provisioned. It uses vendor independent interfaces so that multivendor networks can be built. It allows rapid introduction of new services, and it distributes the intelligence in the network into a few intelligent elements. For more information on intelligent networks, refer to ***Article 6***.

Elements of the Signaling Network

The signaling network is a packet network built using the following elements:

- The *service switching point* is a switch that is able to interact with the signaling network.
- The *signaling transfer point* is a packet switch that routes messages between end points of the SS7 network. Signaling transfer points are often compared to IP routers. Signaling transfer points have no connections to voice trunks or telephone lines.
- The *service control point* is the place of execution of value-added services. Historically, service control points were seen as databases only. The Advanced Intelligent Network architecture describes them more as the place of execution of the service logic.
- *Signaling links* are the physical connection between elements of the SS7 network. They provide a full-duplex 64-kbit/s digital path conforming to the V.35, DS0A, or T1/E1 standards. A group of signaling links connecting the same two elements can be grouped logically into a *linkset*. The SS7 protocol provides procedures for redundancy and load sharing between links of the same linkset. For example, if a link within a linkset fails, the protocol will automatically move the traffic to the nonaffected link and will try to restore the failed link.

Fig. 2 shows typical elements of an SS7 network. The signaling transfer points are provisioned in pairs. Service control points and service switching points always connect to two different nodes. Similarly, links are redundant such that messages can always find an alternate route in case of a failure.

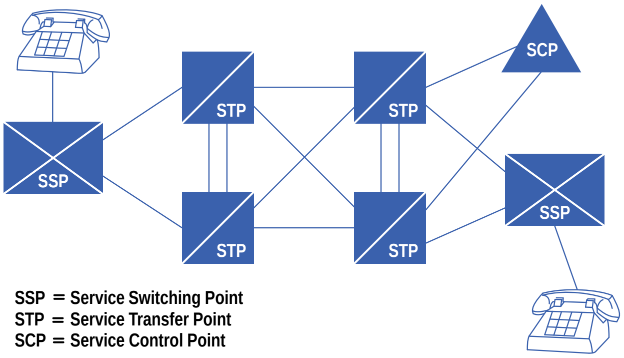

SCP
STP
STP
STP
STP
SSP
SSP
SSP = Service Switching Point
STP = Service Transfer Point
SCP = Service Control Point

Fig. 2. *Elements of the signaling network.*

Use of the Signaling Network

The signaling network is used for many purposes: It is used for regular calls, allowing rapid setup and secure operation. It is used in the wireline fixed network to provide additional services requiring specific service logic and databases (800 numbers, alternate billing, etc.). It is used in the mobile network to manage mobility information. For example, when a mobile phone is switched on, the *home location register* containing the subscriber profile is queried using the signaling network.

The signaling network is now the basic infrastructure for the global telecommunications network. SS7 networks are deployed in almost all countries now, with variable coverage. North American networks are defined by ANSI and Bellcore, while the rest of the world usually follows the ITU standard. The two flavors of the standard are similar, but (of course!) incompatible. The ITU version is used at the boundary of the international networks.

The SS7 Protocol Stack

The SS7 reference model is based on the Open Systems Interconnection (OSI) reference model of the International Organization for Standardization (ISO), following similar principles with layers of protocols. However, the SS7 model is more specialized, being designed for signaling information transfer with a specific focus on low latency and built-in robustness.

Fig. 3 shows the SS7 protocol stack. MTP stands for Message Transfer Part. It represents the three lower layers of the protocol stack. The Signaling Connection Control Part (SCCP) is built on top of MTP Level 3. The ISDN User Part (ISUP) sits on top of MTP (and potentially SCCP also). The Transaction Capabilities Application Part (TCAP) resides on top of SCCP. Let's look at each layer's functions.

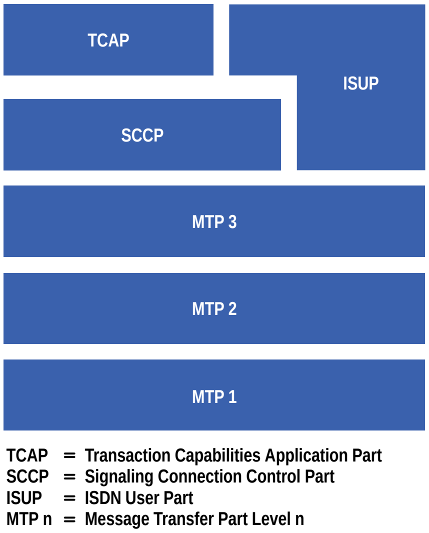

TCAP
ISUP
SCCP
MTP 3
MTP 2
MTP 1
TCAP = Transaction Capabilities Application Part
SCCP = Signaling Connection Control Part
ISUP = ISDN User Part
MTP n = Message Transfer Part Level n

Fig. 3. *The SS7 protocol stack.*

MTP Level 1. The physical layer of the SS7 protocol is based on digital transmission channels known as signaling links, connecting two digital elements with a rate of 56 kbits/s (ANSI) or 64 kbits/s (ITU). The physical network can be composed of V.35, DS0A, or T1/E1 links.

MTP Level 2. MTP Level 2 maps onto layer 2 of the OSI seven-layer model and provides a basic message exchange with an error correction mechanism based on the retransmission of unacknowledged messages. An alignment procedure ensures, if successful, that links are able to convey messages between two points.

Unlike most level 2 protocols, MTP 2 has some unusual features. For example, it keeps filling the available bandwidth by sending small messages (fill-in signaling units or FISU), especially when there is no user traffic. This allows it to promptly detect any physical link failure and to react accordingly. From an implementation point of view, this is a very stressful feature and usually requires specific hardware and firmware. Another interesting feature of MTP 2 is its ability to return unacknowledged frames stored in its buffers to the upper layer (MTP 3) in case of failure. This allows MTP 3 to retrieve the frames from a failed link and send them again on another link without any data loss.

MTP Level 3. MTP Level 3 handles the routing functions and network management procedures of the SS7 network. MTP 3 is the key contributor to the built-in robustness of the SS7 network. The network management functions are the most complex features of the SS7 protocol. They are in charge of maintaining the integrity of the signaling network. These functions can be split into three areas: link management, traffic management, and route management.

Link management is responsible for the integrity of one link. It uses services (especially counters) provided by MTP 2 to monitor the quality of a link. If the link is considered to be in error (excessive error rate, for example), then the link is removed from service, messages are rerouted to alternate links, and the adjacent node is notified to do the same. MTP 3 then starts an alignment procedure to try to restart the link in a clean state.

Traffic management handles traffic on the links within a linkset. It is in charge of load-sharing the traffic over all the active links and of rerouting traffic from a failed link to another.

Route management is in charge of maintaining information on the network topology and the availability or unavailability of certain paths to reach destination nodes. The interesting feature of MTP 3 is that it only knows adjacent nodes and destination nodes and not other intermediate nodes (Fig. 4). It maintains a table of available routes to reach a destination node via an adjacent node.

The other functions of MTP 3 are message routing, discrimination, and distribution. On the outbound path, this means finding the right link to reach the target destination node. On the inbound path, it has to determine for which higher-level protocol the packet is intended (ISUP, SCCP, etc.). It can reroute the packet via the network if it determines that it is not the intended target.

SCCP. SCCP is built on top of the MTP 3 layer and provides additional end-to-end services such as connectionless or connection-oriented service, extended addressing, and network management functions.

SCCP Users are assigned a specific address called a *subsystem number* which, along with the MTP 3 address (called a *point code*), makes it possible to uniquely address an SCCP user in the SS7 network. The extended addressing feature of SCCP allows the use of a label or *global title* in place of the subsystem number and point code to address an application. This allows for symbolic addressing and provides a level of indirection with respect to the physical structure of the SS7 network.

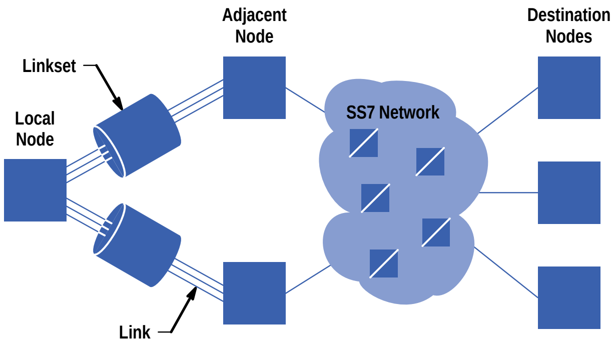


Fig. 4. *The SS7 network as seen by the Message Transfer Part Level 3 (MTP 3) on the local node.*

The translation of the global title to the subsystem number and point code is accomplished either in the signaling transfer points or in the endpoints. Very often, the dialed digits are used as the global title.

In connectionless mode, SCCP operates a bit like UDP (User Datagram Protocol) operates in the Internet world: messages are sent to a target address (either a global title or a subsystem number and point code) and are transmitted from node to node by MTP 3 to the final destination. There is no guarantee of delivery of the message, nor is it guaranteed that the messages will arrive in the order in which they were sent. The connectionless mode is the most widely used, especially because TCAP uses it.

In connection-oriented mode, SCCP operates a bit like X.25. A virtual circuit must first be opened before data transfer can take place. Once the circuit is open, there is a guarantee that messages are delivered and in the right order.

SCCP also has built-in network management functions. Each node in the network maintains the state of its SCCP users identified by their subsystem number. The SCCP layer is in charge of broadcasting the state of its own subsystem numbers to the other nodes, so that at any time, an SCCP user knows about the state of the remote subsystems.

TCAP. The objective of the Transaction Capabilities Application Part is to provide the means of transferring noncircuit-related information (unlike ISUP, which handles circuit-related information) between different nodes of the SS7 network. TCAP is especially used to access service control points in the fixed network or to access home location registers, a short message service center (SMSC), or an equipment identification center (EIC) in the mobile network.

The TCAP layer is divided into two sublayers. The first, the *transaction sublayer*, deals with the exchange of TCAP messages. A transaction, called a *dialogue* at the user level, can be *unstructured* (composed of one unidirectional TCAP message) when no explicit initiation or termination is needed. For more interaction, a *structured dialogue* is used with a beginning, an exchange, and a termination or an abortion. This sublayer uses the SCCP connectionless service.

The upper sublayer is the *component sublayer* and is dedicated to operations. An *operation* is an action (with parameters) to be performed by the remote end. Each operation is encoded into *components*, which are part of a TCAP message payload. Components convey either an operation request or an operation response. Simultaneous operations are allowed inside a transaction and TCAP is able to support multiple simultaneous transactions with different remote TCAPs. The addressing for each TCAP user is the addressing provided by SCCP (point code and subsystem number or global title).

Fig. 5 shows a TCAP interaction with the separation of the transaction layer and the component layer.

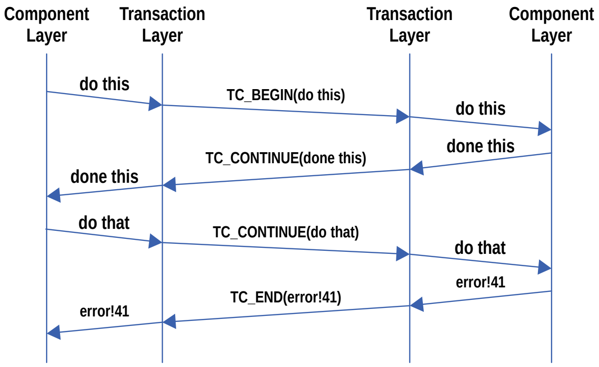


Fig. 5. *A TCAP (Transaction Capabilities Application Part) transaction, showing the transaction layer and the component layer. (ITU-T terminology is shown. ANSI has equivalent services).*

ISUP. The ISDN User Part (ISUP) is a circuit-related protocol, which means that it defines and transports the necessary messages to set up, tear down, and control voice and data circuits. It uses the MTP 3 services to transport messages from switches to switches.

A typical ISUP interaction is shown in Fig. 6. User A takes the receiver off the hook and dials 555-xxxx. The local switch (333) looks up its routing table and finds out that it should route the call to switch 444, which is an *access tandem* (not the final destination). It then sends an *initial address message* to switch 444 via the SS7 network and reserves a voice circuit to switch 444. When switch 444 receives the initial address message, it reserves the other end of the voice circuit, finds out that the call should be routed to switch 555, and sends another ISUP initial address message via SS7 to switch 555. Switch 555 accepts the call, reserves the voice circuit with switch 444, and sends back an *address complete message* to switch 444, which forwards it to switch 333, triggering the ring-back of user A (via the voice path). Switch 555 also rings the destination phone. When user B takes the receiver off the hook, switch 555 sends an *answer message* over the SS7 network to switch 444, which forwards it to switch 333. The call can now proceed.

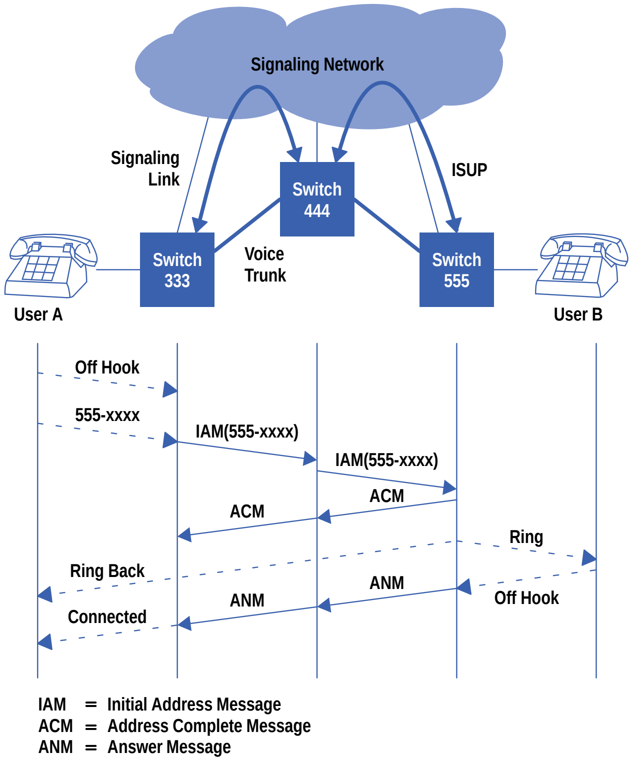


Fig. 6. *A typical ISUP (ISDN User Part) interaction.*

The release phase uses the same kind of message interaction. ISUP also allows many other supplementary services.

The HP OpenCall SS7 Platform

The HP OpenCall SS7 platform allows users to build computer-based signaling applications connected to the signaling network. Using computers to achieve some of the intelligent network functions is one of the key benefits of the intelligent network architecture. Compared to modifying switch software, it is less expensive, faster, and easier to program computers in the intelligent network. The HP OpenCall SS7 platform provides the hardware and the middleware necessary to use a computer in a signaling network.

The main characteristics of the HP OpenCall SS7 platform are:

- It provides the protocols to connect to the SS7 network. This mostly consists of specialized hardware (for MTP Levels 1 and 2) and protocol stacks (MTP, SCCP, TCAP, ISUP) for various flavors (ANSI, ITU, Chinese, hybrid).
- It provides high availability—most of the target applications are mission-critical (see ***Article 8***).
- It provides the necessary components for the computer to be integrated in a central office. This means, for example, support of a −48Vdc power supply, antiseismic capabilities, compliance with established standards, and so on.
- It provides open application programming interfaces (APIs) for users to write applications.

The HP OpenCall SS7 platform is a platform in the sense that it does not provide the application itself but rather allows users to build the application. The platform can be instantiated under several options that will be described later.

Core Protocol Implementation

The network connection is made by a dedicated communications unit called the *signaling interface unit* (see Fig. 7). Each signaling interface unit has a SCSI interface card for the host connection and three slots for signaling link interface cards. These cards provide two links each, with different options for each supported type of link (V.35, DS0A, and T1/E1). These cards come from the HP 37900 SS7 protocol analyzers. The signaling interface unit can also be expanded by means of a dedicated expansion box, which provides four additional slots (eight more links) for a single SCSI attachment. Signaling interface units can be chained on the SCSI bus and the platform can currently support up to 64 links.

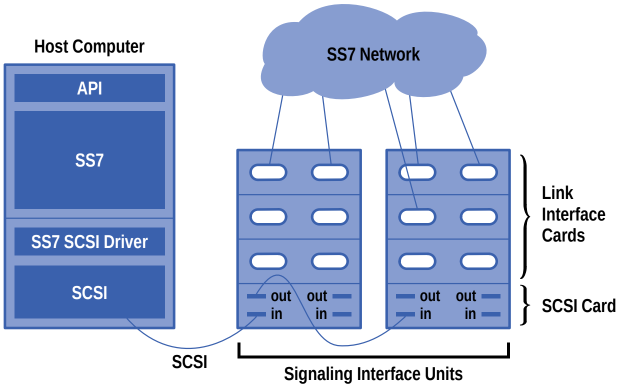


Fig. 7. *The network connection is made by a dedicated communications unit called the signaling interface unit. Each signaling interface unit has a SCSI interface card for the host connection and three slots for signaling link interface cards.*

Each signaling link interface card runs the MTP 1 and MTP 2 protocols and sends and receives messages to and from the host via the SCSI interface.

On the host side, messages are read by an SS7 driver built on top of the SCSI driver in the HP-UX* kernel. On top of these, a single user space process implements MTP 3, SCCP, and TCAP, sending and receiving messages to and from the SS7 driver.

APIs are provided as a library to be linked with the user process. The library is in charge of managing the interaction with the user application, implementing interprocess communication between the application and the SS7 stack, and supporting the flow control.

For each layer of higher-level protocols, an operation, administration, and maintenance (OA&M) programmatic access is provided (Fig. 8). This allows an application developer to control the state of the protocol stack or to implement management applications (monitoring, configuration, etc.).

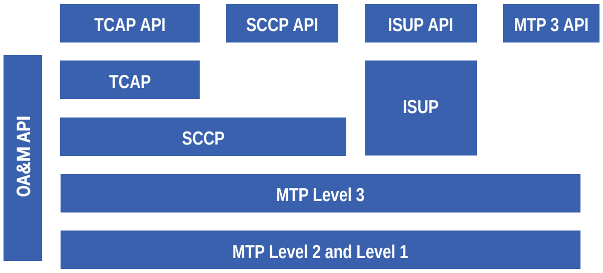


Fig. 8. *For each layer of higher-level protocols, an operation, administration, and maintenance (OA&M) programmatic access is provided.*

Each layer is directly accessible via a direct API. Some APIs are simple wrappers that get the user parameters and marshall them to the stack. Other APIs, such as the ISUP and TCAP APIs, implement some part of the protocol in the library itself, allowing wider distribution of the processing load. All of the APIs are asynchronous to allow for high transaction rates.

High Availability

As explained above, one of the key aspects of the platform is high availability. The SS7 network has built-in high availability capabilities and it is important that the end node also provide these capabilities.

Our solution is based on the *active/standby model* (see ***Article 8*** for more details). To eliminate any single point of failure, every element is replicated (see Fig. 9). Two hosts are used, one being the active host and the other the standby host. Only

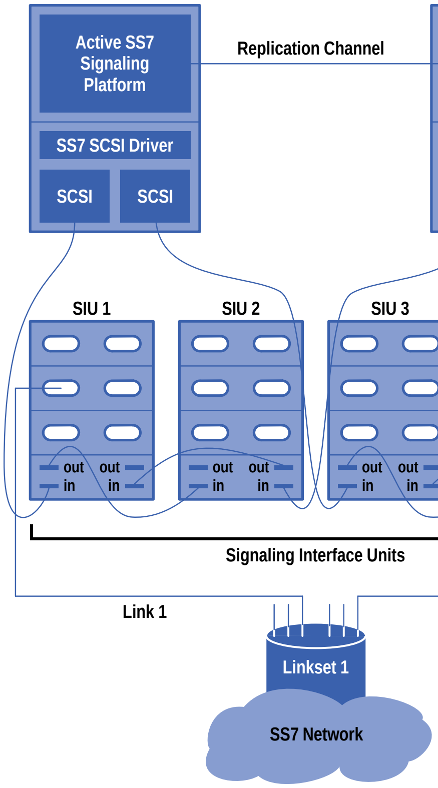


Fig. 9. *High availability of the HP OpenCall SS7 platform is based on the active/standby model.*

the active host processes the traffic, while the standby just keeps its state up to date. For network attachment, we use dual-ported signaling interface units, and each unit is connected to two different SCSI chains terminating at each host. The dual-ported signaling interface unit has built-in logic such that one and only one SCSI bus can be active at any point in time.

Each host has two SCSI interface cards, each connected to one half of the signaling interface unit set. Fig. 9 also shows how signaling interface units are chained. The active SS7 stack, running on the active host, uses the two SCSI chains terminating at it. The standby stack controls its two SCSI chains but does not have control of the dual-ported signaling interface units. The active stack has control of the signaling interface units via its two SCSI interfaces. In case of a failure of the active side, the standby side will take over and will take control of all the signaling interface units by using its two SCSI buses. The switchover happens in less than six seconds to be transparent from the SS7 network point of view. Refer to ***Article 8*** for more details on the mechanism.

The SS7 links of a given linkset must be connected to two different signaling interface units so that if one signaling interface unit happens to fail, the SS7 traffic will be routed transparently by the network to the surviving signaling interface units. Therefore, from a network attachment point of view, the architecture is more a load-sharing architecture, whereas it is an active/standby architecture at the host level.

From an application point of view, the API hides the fact that there are in fact two SS7 stacks running. Application developers are free to use their own high availability mechanism, either load shared or active/standby.

Distribution

Another important aspect of the HP OpenCall SS7 platform is its ability to support distributed applications. The key concept here is a *front-end/back-end mode.* The front-end computer supports the SS7 connection and protocol and the back-end computer supports the application. A typical configuration is shown in Fig. 10.

The SS7 stack is able to distribute the traffic among several instances of the application running on *back-end nodes*. The application instances can run on several nodes and several instances can run on the same host. The API completely hides the distribution and the active and standby instances of the stack. Thus, an application can be configured to run either on a simplex node (no high availability), on a duplex node (active/standby), or on a back-end node without modifying anything in the code. All connections between the various systems are made over a dual LAN (possibly FDDI for high-end systems) to eliminate any single point of failure.

This flexibility allows users to use their own high availability and distribution schemes.

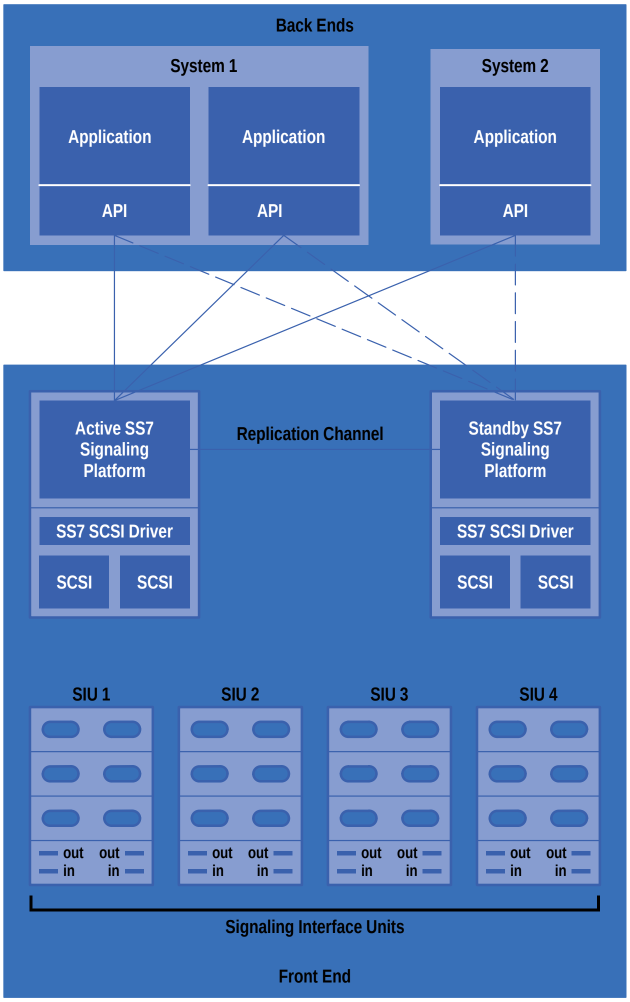


Fig. 10. *A front-end/back-end mode allows the HP OpenCall SS7 platform to support distributed applications.*

Stack Implementation

As mentioned earlier, MTP 3, SCCP, and TCAP are implemented in a single user space process. The protocol implementation started in 1988. The SS7 stack uses object-oriented technology and a message passing bus for interobject communication. Fig. 11 shows the stack implementation.

Each layer has a software bus instantiated. Entities can dynamically register on the bus, specifying what kind of messages they are interested in. Entities are object classes that model elements of the protocol. Typical objects are MTP 3 links, SCCP remote subsystem numbers, and so on. Each object instance is associated with a unique key (object identifier), usually extracted out of the protocol information, which allows very efficient dispatching by the bus. Entities can send messages on the bus to be multicast to the target entities, calling one of their base class methods.

This method has proved to be very efficient in terms of encapsulation and coupling between objects. (Note that the MTP 2 layer does not implement the MTP 2 protocol but rather provides the interface with the MTP 2 implemented in the signaling interface units).

Message Set Customization

To extend the capabilities of the SS7 platform, it is necessary to provide more built-in protocols as they are adopted. These new protocols are built on top of TCAP and are used in the intelligent network or in the mobile network. Although standardized, the flavors of these protocols vary from network to network and from vendor to vendor. The same applies for ISUP, for which there is about one version per country.

To deal with the diversity of message formats at the product level without having to do a special version for every new flavor of the protocols, we have developed a *message set customization* technology to automate the customization of a protocol. This consists of a message set compiler in conjunction with a generic run-time engine to process the encoded messages (see Fig. 12).

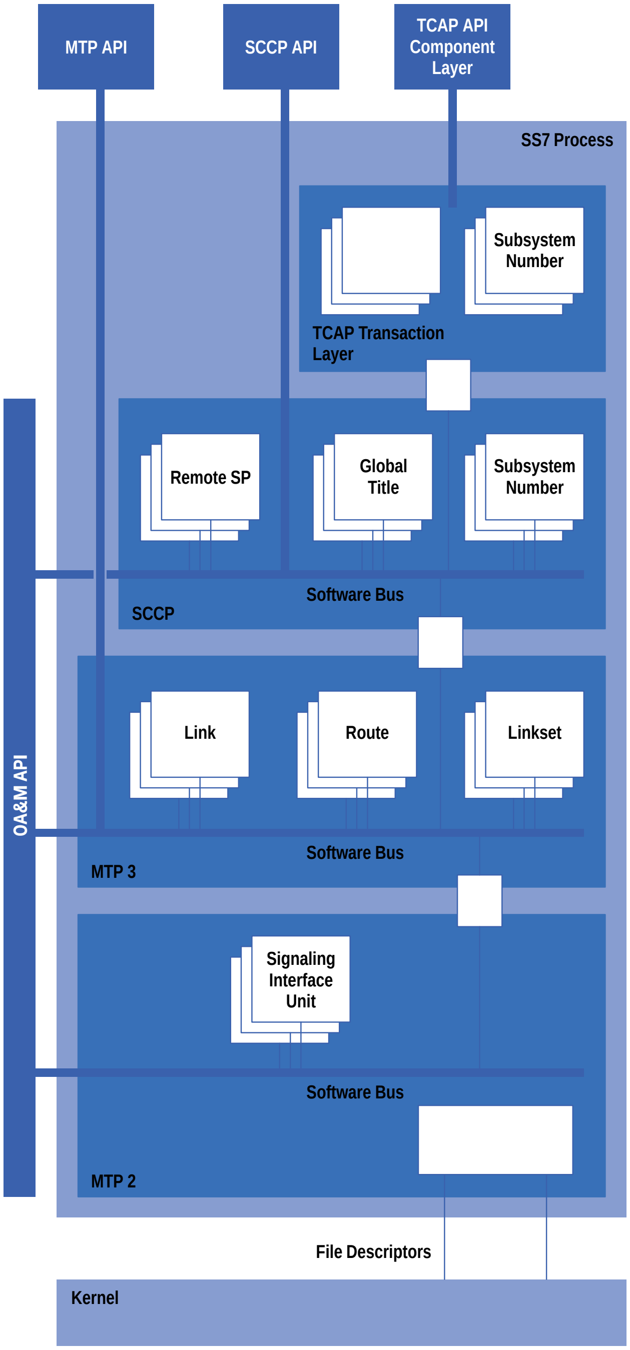


Fig. 11. *SS7 stack implementation in the HP OpenCall SS7 platform.*

The format of the messages used by the protocol is defined in Abstract Syntax Notation 1 (ASN.1) with some specific annotations. ASN.1 is the OSI standard for defining data structures and is used by most of the protocols that we implement. However, the message set compiler technology is not restricted to ASN.1. A protocol such as ISUP, which is not defined in ASN.1, can also be accommodated.

The compiler generates a *metadata file*, which contains the definition of the messages. The run-time engine loads these metadata files and can immediately encode and decode new definitions of messages without impacting the API or requiring recompilation or relinking. The benefit of this technology is that it can adapt the product to the user's exact specifications at the latest possible stage, without impacting the core product.

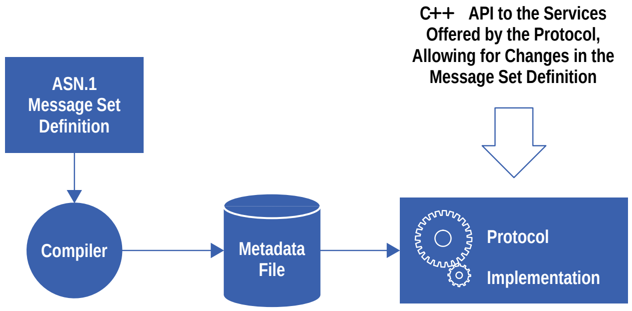

C++ API to the Services Offered by the Protocol, Allowing for Changes in the Message Set Definition
ASN.1 Message Set Definition
Compiler
Metadata File
Protocol Implementation

Fig. 12. *Message set customization technology consists of a message set compiler in conjunction with a generic run-time engine to process the encoded messages.*

Performance

The HP OpenCall SS7 platform can handle more than 2100 SS7 transactions per second, a transaction being defined as one message into a dummy application and one message out from the application. These figures were measured on an HP 9000 Model K420 host computer. The implementation is CPU-bound, so its capacity automatically increases whenever more powerful system hardware becomes available.

The constraints set up for the development were rather stringent and very similar to in-kernel development. For example, no file system access is allowed except at startup, and all calls must be asynchronous. We are closely watching the I/O traffic to avoid falling into I/O bottlenecks. One of the reasons why we do not get I/O bottlenecks is that we group as many SS7 messages as possible together before doing any transfer. For SCSI, this is mandatory because SCSI is architected for rather large data transfers whereas SS7 handles very small messages (∼100 bytes). Therefore, the signaling interface unit and the SCSI driver purposely introduce some latency to transfer larger data blocks.

High Availability in the HP OpenCall SS7 Platform

Fault tolerance in computer systems is discussed and high availability is defined. The theory and operation of the active/standby HP OpenCall solution are presented. Switchover decision-making power is vested in a fault tolerance controller process on each machine.

by Brian C. Wyld and Jean-Pierre Allegre

Our lives are increasingly dependent on our technology. Some things are trivial, like being able to watch our favorite TV program, while some are much more important, like medical equipment. When you start to look at the difference technology makes to our lives, starting from the convenience, you begin to appreciate the problems we would have if it were to break down.

Some breakdowns are merely irritable. Not being able to phone to arrange a night out isn't going to kill anyone. But when you can't phone for help in an emergency, then a lack of the service we all take for granted is a lot more serious. Being able to ensure that the technology we use every day keeps working is therefore a part of the functionality, as much as providing the service in the first place.

Although it would be nice if things never broke down, we all know that this is impossible. Everything has a flaw—entropy always gets us in the end. The unsinkable ship sinks, the uninterruptible power supply gets cut, the unbreakable plate proves to be only too breakable (usually with the assistance of a child).

We all depend in one way or another on the continued functioning of our technology, so we all have some dependence on the tolerance of that technology to the faults that will inevitably strike it. When a computer malfunction can disrupt the lives of millions of people, fault tolerance is not just nice, but absolutely necessary.

Computer Fault Tolerance

Computer fault tolerance covers a range of functionality. The important aspects to consider are the speed of recovery in the presence of a fault, the perception of the users in case of a fault, and the consequences to the application of a fault. With these in mind, the following degrees of fault tolerance are often defined.

Reliable Service. The system is built to be as reliable as possible. No effort to provide tolerance of faults is made, but every part of the system is engineered to be as reliable as possible to avoid the possibility of a fault. This includes both the hardware, with either overengineered components or rigorous testing, and the software, with design methods that attempt to ensure bug-free code and user interfaces designed to prevent operator mistakes. Reliability rates of 99.99% can be achieved.

High Availability. The emphasis is on making the service available to the user as much of the time as possible. In the event of a fault, the user may notice some inconsistency or interruption of service, but will always be able to reconnect to the service and use it again either immediately or within a sharply bounded period of time. A reliability rate of 99.999% is the target.

Continuous Availability. This is the pinnacle of fault tolerance. When a fault occurs, the user notices no interruption or inconsistency in the service. The service is always there, never goes away, and never exhibits any behavior that leads the user to think that a fault might have happened. Needless to say, this level is both difficult and expensive to obtain. The reliability rate is 100%.

In general, a fault tolerant system will be highly available in most parts, with touches of continuous availability.

Achieving Fault Tolerance

Fault tolerance is usually achieved by using redundant components. Based on the assumption that any component, no matter how reliable, will eventually either fail or require planned maintenance, every component in the system that is vital is duplicated. This redundancy is designed so that the component can be removed with a minimal amount of disruption to the operation of the service. For instance, using mirrored disk drives allows a disk to fail or be disconnected without altering the availability of the data stored on the disk.

The way the redundancy is designed varies according to the paradigm used in the system. There are several ways to build in the redundancy, depending on the use made of the redundant components and how consistency is maintained between them.

Multiple Active. The redundant components may in fact be used simply to provide the service in a load-sharing way. In this case, data and functionality are provided identically by all the components. The load from the users of the service is spread across the components so that each handles a part of the load. In the event of a component failure, the load is taken up by the others, and their load increases correspondingly.

Active/Standby. In this paradigm, there exist one or more active components, which provide the service to the users, and in parallel, one or more standby components. These provide a shadow to the actives and in the event of an active failing, change state to become active and take over the job. Variations on the theme involve one-to-one active/standby pairs, N actives to one standby, and N actives to M standbys.

Coupling

How the service is affected by the failure is also important. The redundant component, whether it was also providing the service or not, can be loosely or tightly coupled to the other components.

When loosely coupled, the redundant component only has a view of the state of the active at certain times—at the end of a transaction, for instance. This has the effect that a failure of the component while processing a user's request will lose the context. Any finished work will be unaffected, but work in progress is lost and the user must restart. However, the effort required to keep the standby coupled is low.

A tightly coupled component will remain in step with the component processing the request, so that it can take over seamlessly in the event of the fault. The workload is much higher because many more messages must be exchanged, and the speed of the operation may be slower to ensure that at every stage the standby is in step with the active.

Of course, many shades and granularities of loose versus tight coupling are possible in a single system.

Traditionally, hardware fault tolerant systems have been exponents of the tight coupling paradigm: two or more processors executing exactly the same instructions in synchronization, with the outputs selected on either an active/standby basis or by a voting system. Software systems have leaned more towards the loose coupling method, at various levels of granularity. For instance, there are database transactional paradigms in which user database accesses are bundled into transactions, and only once a transaction is committed does that transaction become certain and unaffected in the event of a failure.

Predicting and Measuring Fault Tolerance

Various statistical methods exist to measure the fault tolerance of a system in a quantitative manner. These usually use the standard measures of system failure such as MTBF (mean time between failures) and MTTR (mean time to repair), and are combined to give a forecast of application downtime. However, the values of downtime produced by such methods can be inaccurate, and sometimes bear little resemblance to the true values.

The main reason for this is that failures may not be isolated and uncorrelated, and this is very difficult to take into account. Simply predicting from the MTBF and MTTR that the chance of a single failure bringing down the entire system is very small is not realistic when the single failure will often provoke subsequent related failures, often in the part of the system trying to recover from the fault. Most fault tolerant systems and related statistical analysis are based on an assumption of a single failure, and systems are built to avoid a single point of failure. In practice, and in the often inconvenient real world, failures can happen together, and can cause other failures in turn.

The other assumption that causes trouble is that of silent failure. It is often assumed that when a component fails, it does so silently, that is, in a failure mode that doesn't affect other components. For instance, having a dual LAN between several computers to avoid the LAN's being a single point of failure doesn't help when a crashed computer decides to send out nonsense on all of its LAN interfaces, effectively preventing use of any LAN.

Downtime Causes

Things that cause downtime on systems can be grouped into several main categories. First is the obvious computer hardware failure. This may be caused by a component's lifetime being exceeded, by a faulty component, or by an out-of-specification component. Often, hardware failures in one component can cause other components to fail. Many computer systems are not constructed to allow a single component to fail or to be replaced without affecting other components. For instance, a failed disk drive on a SCSI bus will force the entire system to be halted for its replacement even though only one component has failed.

This often implies that avoiding the single point of failure means adding more hardware than might seem reasonable —a second SCSI controller card and chain for instance, so that the backup disk drive can be on a separate SCSI bus. Reliable hardware, coupled with a system built to allow hot-swappable components can do a lot to eliminate this source of downtime.

The second obvious cause of downtime is software failures. No software will ever be entirely bug-free. Even formal methods, quality reviews, and all the rest of the trappings of computer science cannot keep those elusive problems from slipping in. The main problem with bugs is that the ones that escape to released systems are usually in code that has not been well-tested. This may often be the code designed to recover from failures, since this is difficult to cover fully with testing. Recovery may also often involve a higher load than normal as standby processes become active, load files, and so on, and this can often expose lurking bugs. The net effect is that when your application crashes and burns, your standby application, ready and waiting to continue the service, promptly follows it down in flames instead.

Another not so obvious but very real source of downtime is operator intervention. Although in theory, operators of a system will always follow procedures and always read the manual, in practice they are prone to typing rm * (deleting all of the files on the disk), and pulling out the wrong power plug. Even when the mistake is not so obvious, mistakes such as badly configured systems, too much load on a critical system, or enabling unneeded tracing or statistics can bring the system down.

No amount of clever fault tolerant algorithms or mathematically proven designs will help here. However, a carefully planned system configuration, with working defaults and a user interface that is designed to help the user make the correct choices by presenting the correct information in a timely and obvious fashion, can go a long way towards avoiding these sorts of problems. This is, unfortunately, an often neglected part of the system design, with attendant problems. When building a system that end users will use as a platform, with programming interfaces, the importance of providing usable interfaces becomes even greater.

Finally, there are the disasters. The absolutely unforeseeable, one-in-a-million, couldn't happen in a lifetime chances. Who could predict that the earthmoving equipment would knock down the pylon supplying the mains electricity to the computer center, which would fall onto the telecommunications lines, blowing every piece of data communications equipment, after which it would careen into the hole where the water main was being repaired, breaking it and flooding the basement where the batteries for the UPS are installed, completely destroying them? “Impossible,” you might say, but it has happened. In such cases, geographically separated sites can prove to be the only possible solution if a 100% available system is really required. This does rule out certain forms of fault tolerance—any form of dual-ported hardware, for instance, or lockstep processors—but is possible with software fault tolerance techniques.

Telecommunications Fault Tolerance

The requirements on a fault tolerant system vary with the application. In telecommunications, we see different requirements being demanded depending on the element being addressed. On the billing services side, the requirements are biased towards ensuring that no data loss occurs. Limited application downtime is acceptable but any billing data should be safe. This sort of system is similar to the requirements of any database-oriented application, and technologies such as mirrored disks and reliable systems are usually sufficient.

For operations services, which provide the management of the network, certain essential administration and management functions should always be available so that control over the network is always maintained. In the service provision environment for which the HP OpenCall SS7 platform is designed, the essential requirements are to avoid disruption to the network, to have a continuously available service, and to avoid disruption to calls in progress in the event of a fault.

To avoid disruption to the network, the SS7 protocol provision has to avoid ever being seen as down by the network. This essentially means that in the event of a fault, normal protocol processing must resume within six seconds. Any longer than this and the SS7 network will reconfigure to avoid the failed node. This reconfiguration process can be very load-intensive and can cause overloads in the network. This effect is to be avoided at all costs.

To provide continuous availability requires that the application that takes over service processing from a failed application must be at all times in the same state with respect to its processing. This is also required to ensure that current calls in progress are not disrupted. The state and data associated with each call must be replicated so that the user sees no interruption or anomaly in the service.

HP OpenCall Solution

To fulfill all of these requirements for a telecommunications services platform is not an easy task. We chose to implement a simple active/standby high availability fault tolerance model that is capable of providing most customer needs.

To achieve high availability, we need to replicate all the hardware components (see Fig. 1). We have defined a platform as being a set of two computers (usually HP 9000 Series 800) interconnected by a dual LAN, equipped with independent mirrored disks and sharing a set of SS7 signaling interface units via redundant SCSI chains (see ***Article 7*** for more details).

The highest constraint on the system is to be able to perform a switchover in less than six seconds. The SS7 protocol MTP Level 2 running on the signaling interface unit can tolerate a 6-s interval without traffic. If this limit is exceeded, the SS7 network detects this node as down, triggering a lot of alarms, and we've missed our high availability goal.

In a nutshell, the high availability mechanism works as follows. One system is the active system, handling the SS7 traffic and controlling all the signaling interface units. In case of a failure on the active side, the standby system gets control of the signaling interface units and becomes the new active. During the transition, the signaling interface units start buffering the data. When the buffers are full (which happens rapidly), the signaling interface units start sending MTP Level 2 messages to the other end to signal a transient outage. If this outage lasts more than 6 s, the SS7 network detects this node as down, so it is critical that in less than 6 s, a new active system take over. The failure detection time is the most crucial one. We need to detect failures in less than four seconds to be able to perform a safe switchover.

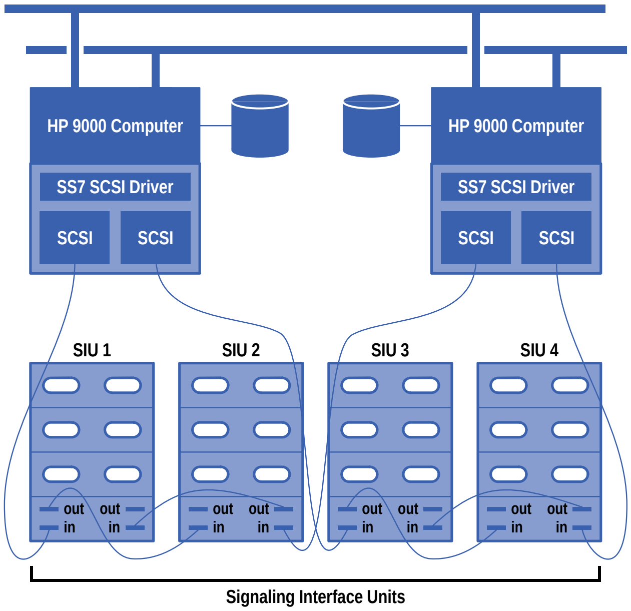


Fig. 1. *The high availability solution in the HP OpenCall SS7 platform calls for replication of all hardware components.*

The architecture goal is to protect the platform from failing in case of a single failure. In general, the dual-failure case leads to a service loss, even though some cases may be recovered.

Software Model

The HP OpenCall SS7 platform is based on an active/standby model, with a pair of UNIX® processes providing a service. Only one of the pair is actually doing the job at any time (the active) while its peer process (the standby) is idle, waiting to take over in the event of a failure.

The service provided by the process may be the SS7 protocol stack, centralized event management, or a telecommunications application. The standby process is not completely idle. It must be kept up to date with the state of the active process to be able to resume processing from the same point if a failure occurs. When in this state, it is *hot standby* (see Fig. 2).

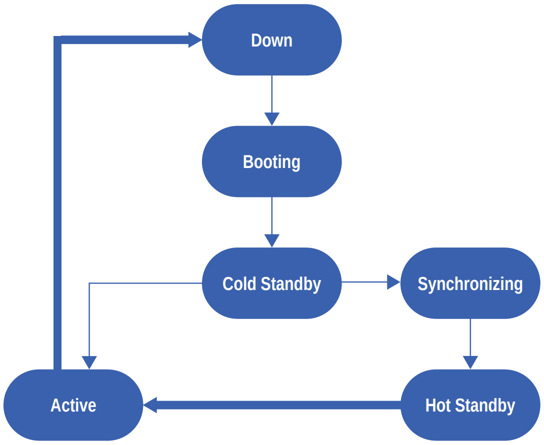


Fig. 2. *A process starting up with an active process running is in the down state. When it reaches the hot standby state, it is ready to become the active if the active goes down.*

Consider a process starting up when an active process is already running. A process is initially *down*, that is, not running. When it is started, it performs whatever startup process is required (*booting*), and then is *cold standby*. In this state, it is correctly configured and could perform the service if required, but all current clients would see their states being lost.

This would be enough to give a highly available service, but would not fulfill the requirement to avoid disruption to current clients. The process must therefore now synchronize itself with the active process, during which time it is *synchronizing*.

Once it is completely up to date, it is hot standby. In this state, current clients should see no disruption if the active process fails.

Obviously, if no active is running, the process goes to active from cold standby, since there are no current clients.

Once there exists this pair of processes, one active providing the service and one standby providing the backup, the system is ready to deal with a failure. When this occurs, the failure must first be detected, and then a decision on the action to be taken must be made.

Fault Tolerance Controller

The HP OpenCall SS7 platform centralizes the decision-making process into a single controller process per machine, which is responsible for knowing the states of all processes controlled by it on its machine. It has a peer controller (usually on the peer machine) which controls all the peer processes. These two *fault tolerance controllers* make all decisions with regard to which process of the pair is active. Each high availability process has a connection to both the fault tolerance controller and to its peer.

The fault tolerance controllers also have a connection between them (see Fig. 3). The A channel allows the two fault tolerance controllers to exchange state information on the two system processes and to build the global state of the platform. The A channel also conveys heartbeat messages. The B channels allow the fault tolerance controllers to pass commands governing the state of the processes, and to receive their state in return. A process cannot change state to become active without receiving a command from the fault tolerance controller. Because the fault tolerance controller has information on all high availability processes and on the state of the LAN, CPU, and all peer processes, it can make a much better decision than any individual process. Finally, the C channels are replication channels. They allow peer processes to replicate their state using an application dependent protocol.

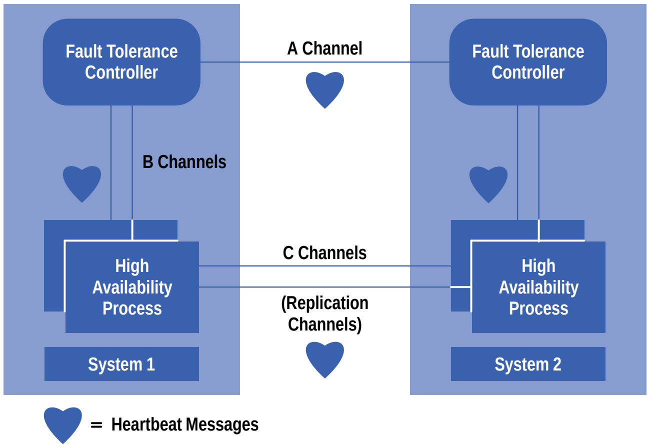

Fault Tolerance Controller
A Channel
Fault Tolerance Controller
B Channels
C Channels
High Availability Process
(Replication Channels)
High Availability Process
System 1
System 2
♥ = Heartbeat Messages

Fig. 3. *The fault tolerance controllers make all decisions with regard to which process is active. Each high availability process has a connection to both the fault tolerance controller and to its peer. A heartbeat mechanism helps detect failures.*

Failure Detection

For the success of any fault tolerant system, failures of any component must be detected quickly and reliably. This is one of the most difficult areas of the system. The HP OpenCall SS7 platform uses several mechanisms to detect various kinds of faults.

To detect a failure of one of the high availability processes, a heartbeat mechanism is used between the fault tolerance controller and the high availability process via the B channel. UNIX signals are also used to detect a failure of a child process (the fault tolerance controller is the parent of all high availability processes), but they provide information only when a process exits. To detect more subtle process faults such as deadlocks or infinite loops, the heartbeat mechanism is required, but it has the drawback of mandating that every high availability process be able to respond to heartbeat messages in a timely fashion, usually around every 500 ms. This is not so critical in our environment since we expect our processes to behave in a quasi-real-time manner, but it rules out using any potentially blocking calls.

To detect system hang, we use a specific mechanism implemented in the fault tolerance controllers. Each fault tolerance controller, running in HP-UX* real-time priority mode (rtprio), uses a real-time timer that ticks every 2 s (typically). At every tick, the fault tolerance controller checks the difference between the last tick and the current time. If this difference exceeds a certain tolerance, this means that the system has been hung for a while, since the fault tolerance controller is configured to have the highest priority in the system and should therefore never be prevented from receiving a real-time timer. Upon occurrence of such an event, the fault tolerance controller exits after killing the other high availability child processes. As strange as it may sound, this is the safest thing to do. If the system has been hung for a while, the peer fault tolerance controller should have also detected a loss of heartbeat and should have decided to go active. If we were to let the fault tolerance controller of the hung system keep running once it wakes up, we would have two active systems, with all the possible nasty effects this entails.

The two fault tolerance controllers also exchange heartbeat messages (along with more detailed state information). Should a heartbeat fail, the fault tolerance controller of the active side will assume that the peer is down (for example, because of a dual-LAN failure, a peer system panic, or a peer fault tolerance controller failure) and will do nothing (except log an event to warn the operator). If the fault tolerance controller of the standby side detects this event, the fault tolerance controller will assume that something is wrong on the active side. It will decide to go active and will send an activate command to all of its high availability processes. If the old active is indeed dead, this is a wise decision and preserves the service. In the case of a dual-LAN failure (this is a dual-failure case that we are not supposed to guard against), we may have a split-brain syndrome. In our case, we use the signaling interface unit (see article, page 1) as a tiebreaker. If the active SS7 stack loses control of the signaling interface unit (because the peer stack has taken control of it), it will assume that the other system is alive and will exit, asking the fault tolerance controller not to respawn it. Operator intervention is necessary to clear the fault and bring the platform back into its duplex state.

Dual-LAN Support

At the time the HP OpenCall SS7 platform project started, no standard mechanism existed to handle dual LANs, nor did we want to implement a kernel-level dual-LAN mechanism. We therefore selected a user space mechanism provided by a library that hides the dual LAN and provides reliable message-based communication over two TCP connections (Fig. 4).

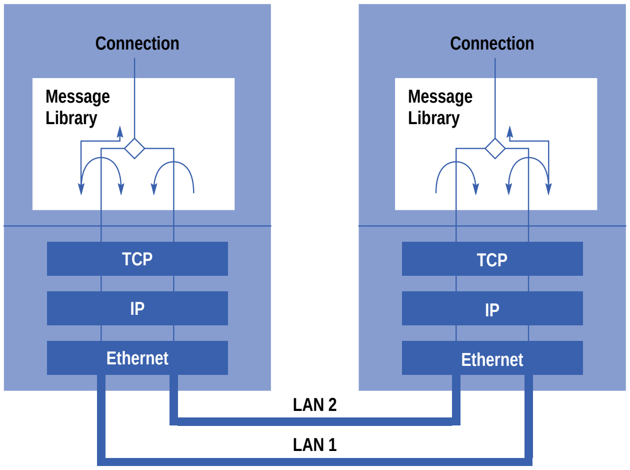


Fig. 4. *The dual-LAN mechanism is a user space mechanism provided by a library that hides the dual LAN and provides reliable message-based communication over two TCP connections.*

A *message library* provides the dual-LAN capability and message boundary preservation. The message library opens two TCP connections, one on each LAN. Only one LAN is used at a time (there is no attempt to perform load sharing). On each of the TCP connections, we maintain a small traffic of keep-alive messages (one every 500 ms), which contain just a sequence number. On each TCP connection, the message library monitors the difference between the sequence numbers on each LAN. If the difference exceeds a given threshold, one LAN is assumed to be either broken or overloaded, in which case the message library decides to switch and resume traffic on the other LAN. No heartbeat timer is used. Only differences in round-trip time can trigger a LAN switch. The benefit of this solution is that it is independent of the speed of the remote process or remote machine and scales without tuning from low-speed to high-speed LANs. It also allows very fast LAN switching time.

A drawback is the sensitivity of this mechanism to a loaded LAN, which is perceived as a broken LAN. For this, we recommend that an extra LAN be added to the system dedicated to application bulk traffic. Another problem is that when switching LANs, we have no way of retrieving unacknowledged TCP messages to retransmit on the new LAN, so we end up losing messages upon a LAN switch. Some parts of the platform guard themselves against this by implementing a lightweight retransmission protocol.

Access to High Availability Services

An important objective of the HP OpenCall SS7 platform is to shield the application writer from the underlying high availability mechanisms. We came up with the scheme illustrated in Fig. 5 to access the high availability processes.

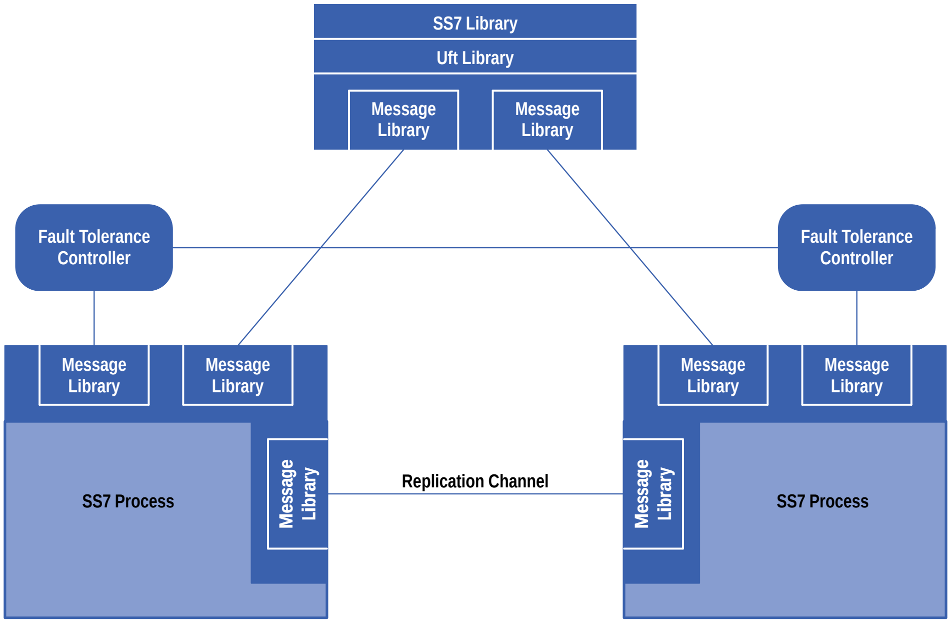


Fig. 5. *The method of accessing the high availability processes is designed to shield the application writer from the underlying high availability mechanisms.*

Let's take the example of the SS7 process. The *SS7 library* maintains two message library connections (four TCP connections because of the dual LAN): one with the active instance and one with the standby. The *Uft library* (user fault tolerance library) transparently manages the two connections and always routes the traffic to the active instance of the stack. Upon switchover, the SS7 process informs its client library via a surviving message library connection that it is now the new active and that traffic should be routed to it. From an API point of view, the two connections (four sockets) are hidden from the user by exporting an fd_set as used by select() instead of a file descriptor. The application main loop should be built along the following lines:

```
while(1) {
  API-pre-select(&rm,&wm,&em,&timeout);
  // application possibly adds
  // its own fd in the mask here
  // FD_SET(rm,myFd);
  select(&rm,&wm,&em,&timeout);
  API-post-select(&rm,&wm,&em,&timeout);
  // application possibly checks
  // its own fd here
  // if (FD_ISSET(rm,myFd)) {}
}
```

The application main loop must continuously call the preselect function of the API to get the accurate value of fd_set (sockets can be closed and reopened transparently in case of failure), then call select(), possibly after having set some application-specific file descriptor in the mask, then call the API postselect function with the mask returned by select(). In the postselect phase, the library handles all necessary protocol procedures to maintain an accurate view of the state of the high availability process, along with user data transfer.

State Management

One of the key aspects of the active/standby paradigm is the definition of the process state and how precisely it must be replicated. The framework described above does not enforce how state management should be performed. It provides a replication channel between the two peer processes and information about the processes, but no specific semantics for state. Different schemes are used depending on the nature of the state to be replicated. A key element to consider when designing such a system is the state information that must be preserved upon switchover and its update frequency. For instance, blindly replicating all state information in a system targeted at 4000 messages per second would be highly inefficient, because the replication load would exceed the actual processing.

For these reasons, we have not set up a generic state replication mechanism, but rather build ad hoc mechanisms depending on the nature of the state. For example, on the SS7 stack, the MTP 3 protocol has no state associated with data transfer (such as window value, pending timer, connection state), but has a lot of network management information that must not be lost in case of switchover.

The policy has been to intercept MTP 3 management messages coming from the network or from the OA&M (operation, administration, and maintenance) API and send them to the standby via the replication channel or the API. The standby stack processes the MTP 3 management messages in the same way as the active and the computed states are identical.

TCAP transactions are not replicated because of the high rates of creation and deletion and the amount of state information associated with the component handling. The effect is that opened TCAP transactions are lost in case of switchover. Work is progressing on a scheme that preserves transactions by letting the user of the transaction decide when the transaction becomes important and should be replicated.

An alternative to replicating messages is to replicate the state after it has been computed from the message. The usual algorithm for this scheme is to do the computation on the active side, use an ad hoc protocol to marshall the new state to the standby, and let the standby update itself. If the standby fails to replicate the state, it decides to go to the down state and will be restarted.

Another important design aspect for the high availability system is the synchronization phase. A starting cold standby system has to perform the cold standby to hot standby transition by getting all the state information from the active and rebuilding it locally. This operation should disturb the active as little as possible, but care must be taken that the algorithm converges. If the state of the active changes faster than the standby can absorb, there is a risk that the standby may never catch up. This is usually addressed by assuming that the standby has much more CPU available than the active, and if necessary, slowing down the active. In the case of SS7 signaling, the amount of state information is rather small and the information is stable, so we use a relatively simple algorithm. The synchronization is performed by having the standby fork a helper process that, via the SS7 OA&M API, dumps the content of the state information to disk and then replays it to the standby via the API. To check that the state is correct and that the standby can go to hot standby, the standby stack initiates an audit phase that checks that the two configurations are identical. If this is not the case, the process is resumed. Otherwise, the state of the standby goes to hot standby. This is a simple implementation, but has proven to be sufficient for SS7.

Technical Challenges

Developing a high availability platform on the HP-UX operating system has been a great challenge, but we've obtained a very stable and operational product, deployed in hundreds of sites worldwide.

One of the technical challenges was that HP-UX is not a real-time operating system and we need determinism to handle the high availability aspects, especially with the very small reaction time (<6 s) that we are allowed. We've addressed this by forbidding some operations (like file system access and potentially blocking system calls) in time-critical processes such as the SS7 stack, by slicing every long-lived operation into small operations, and by trying to stay below the saturation limit (where response time starts to increase rapidly). For example, we recommend keeping overall CPU utilization below 85% and staying far below the LAN maximum bandwidth.

Another challenge was time synchronization between the various hosts. We do not need time synchronization between the hosts for proper operation, but some of our customers request it. We've used the NTP package (Network Time Protocol), which has proved to work reasonably well except when the clock drift between the hosts was too large to be compensated smoothly and NTP decided to suddenly jump the system clock to catch up. This caused problems for synchronization of events, and also fired our failure detection mechanisms. We resolved these problems using external clocks and configuring NTP in a controlled manner to avoid such time jumps.

A Benchtop Inductively Coupled Plasma Mass Spectrometer

The HP 4500 is the first benchtop ICP-MS. It has a new type of optics system that results in a very low random background and high sensitivity, making analysis down to the subnanogram-per-liter (parts-per-trillion) level feasible. It can be equipped with HP's ShieldTorch system, which reduces interference from polyatomic ions.

by Yoko Kishi

Inductively coupled plasma mass spectrometry (ICP-MS) is an analytical technique that performs elemental analysis with excellent sensitivity and high sample throughput. The ICP-MS instrument employs a plasma (ICP) as the ionization source and a mass spectrometer (MS) analyzer to detect the ions produced. It can simultaneously measure most elements in the periodic table and determine analyte concentration down to the subnanogram-per-liter or part-per-trillion (ppt) level. It can perform qualitative, semiquantitative, and quantitative analysis and compute isotopic ratios.

The schematic diagram of an ICP-MS instrument is shown in Fig. 1. Basically, liquid samples are introduced by a peristaltic pump to the nebulizer where a sample aerosol is formed. A double-pass spray chamber ensures that a consistent aerosol is introduced to the plasma. Argon (Ar) gas is introduced through a series of concentric quartz tubes, known as the *ICP torch*. The torch is located in the center of an RF coil, through which 27.12-MHz RF energy is passed. The intense RF field causes collisions between the Ar atoms, generating a high-energy plasma. The sample aerosol is instantaneously decomposed in the plasma (plasma temperature is in the order of 6,000 to 10,000K) to form analyte atoms, which are simultaneously ionized. The ions produced are extracted from the plasma into the mass spectrometer region, which is held at high vacuum (typically 10^{-6} Torr, 10^{-4} Pa). The vacuum is maintained by differential pumping.

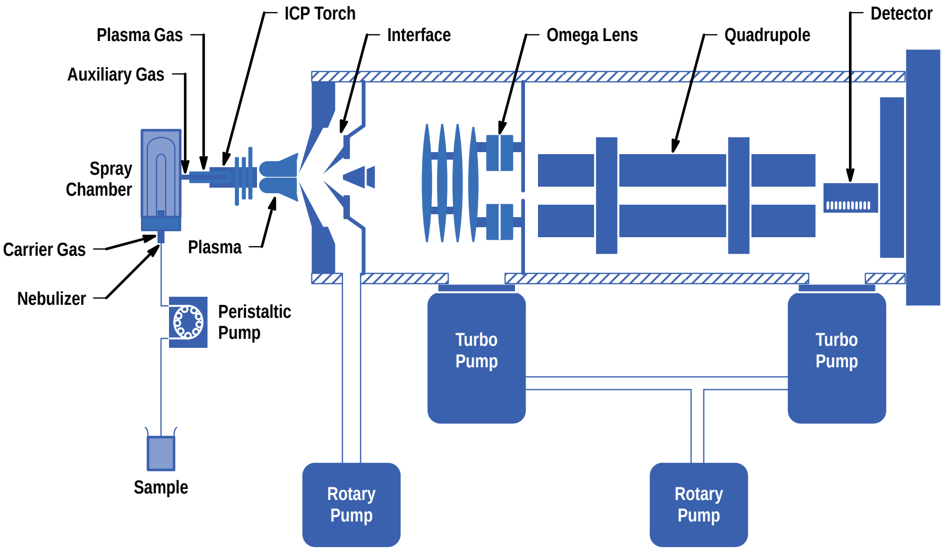


Fig. 1. *HP 4500 ICP-MS schematic diagram.*

The analyte ions are extracted through a pair of orifices, approximately 1 mm in diameter, known as the *sampling cone* and the *skimmer cone*. The analyte ions are then focused by a series of ion lenses into a quadrupole mass analyzer which separates the ions based on their mass/charge ratio (m/z). The term quadrupole is used because the mass analyzer is essentially four parallel molybdenum rods to which a combination of RF and dc voltages is applied. The combination of these voltages allows the analyzer to transmit only ions of a specific mass/charge ratio. Finally, the ions are measured using an electron multiplier, and data at all masses is collected by a counter. The mass spectrum generated is extremely simple. Each elemental isotope appears at a different mass (e.g. ^{27}Al would appear at 27 amu) with a peak intensity directly proportional to the initial concentration of that isotope. The system also provides isotopic ratio information.

New Benchtop ICP-MS

The HP 4500 is the world's first benchtop ICP-MS (see Fig. 2). The reduction in instrument size is dramatic: the size of the previous model is 1550 by 900 by 1450 mm, while that of the HP 4500 is 1100 by 600 by 582 mm. Previous generations of ICP-MS instruments had requirements—space, utilities, and environment—that dictated that a special room be dedicated for the instrument. Installing an ICP-MS could be particularly difficult, since major construction changes were often required.

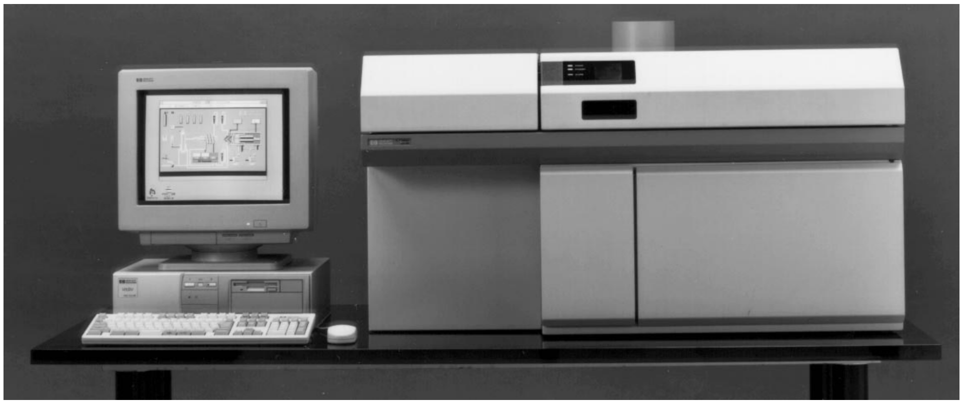


Fig. 2. *HP 4500 ICP-MS system.*

The HP 4500 is smaller and lighter so that it can be installed on an existing bench. The layout of the instrument is designed to make user interaction with the sample introduction system, the interfaces, and the ion lenses routine. All parts can be accessed from the front and connected or disconnected easily. These and other new features and technology introduced and used by the HP 4500 help to make ICP-MS a more routine and therefore a more accessible technique.

Ion Lens System

The configuration of the ion lens system is one of the key design issues because it directly affects the ion transmission efficiency of an ICP-MS system. Various ion lens configurations were produced and evaluated to determine the optimum configuration and operating conditions for the HP 4500. Ion trajectories through each ion lens system were predicted mathematically.

The HP 4500 is equipped with a new type of optics system, as shown in Fig. 3a. The *omega* lens consists of a pair of crescent-shaped lenses that resemble the Greek letter Ω. The optics system contains two omega lenses, the omega+ and omega– lenses, which bend the ion beam, allowing the quadrupole and detector to be mounted off-axis. This prevents photons from reaching the detector (which would increase random background noise), and also focuses the ions very efficiently. The result is a very low random background and high sensitivity, making ultratrace analysis down to the subnanogram-per-liter level feasible. In contrast, other ICP-MS systems employ a photon stop lens system as shown in Fig. 3b.[1] Ions are defocused after extraction into the main vacuum chamber and then refocused, while photons are blocked

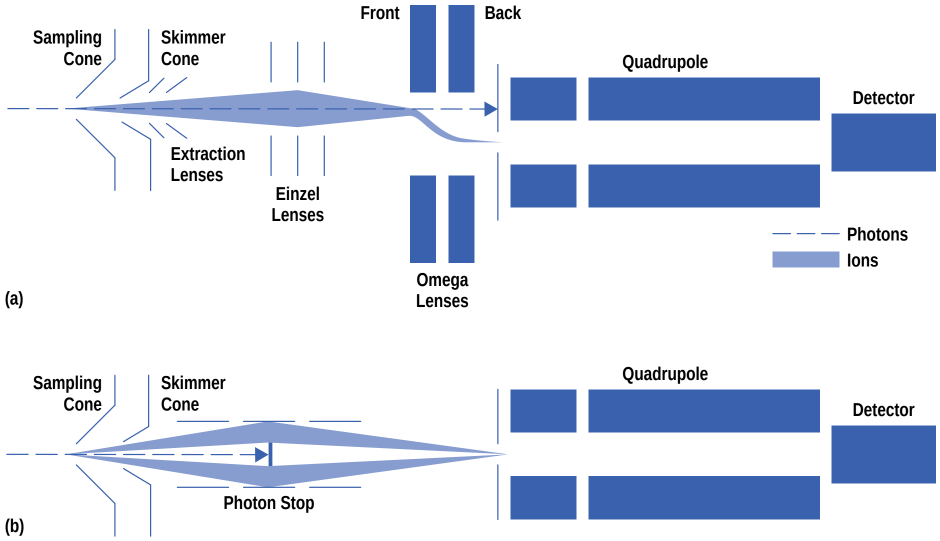


Fig. 3. *Ion lens system. (a) HP 4500 omega lens system. (b) Photon stop system.*

by the photon stop. With this design, some ions inevitably collide with the photon stop and are lost, so overall transmission is reduced.

An example of ion trajectory mapping for the optics system of Fig. 3a is shown in Fig. 4. In this example, the initial ion energy was estimated at 10 eV and the space-charge effect[2] was ignored. The broad trace in the center shows the ion trajectories for the lens voltage settings shown. Starting from the left, the lenses and their voltages are: skimmer cone (no voltage), extraction lens 1 (–160V), extraction lens 2 (–70V), einzel lens 1 (–100V), einzel lens 2 (8V), einzel lens 3 (–100V), omega bias lens (–35V), omega+ lens (4V), omega– lens (–5V), quadrupole focus and plate bias lenses (–10V). The einzel lenses are a traditional electrostatic lens system in which the voltage on the center lens is different from the voltage on the other two lenses.

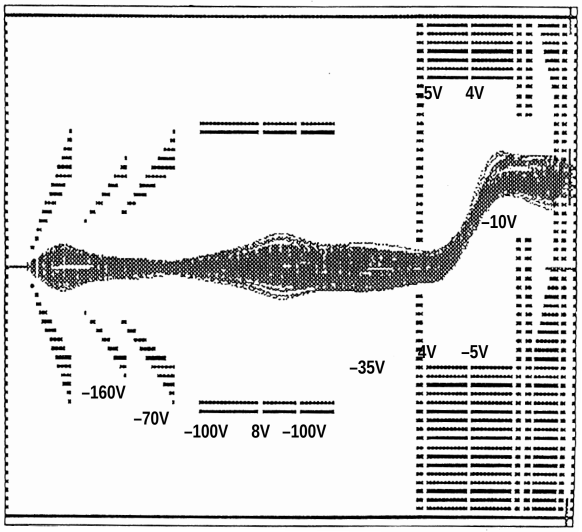


Fig. 4. *Example of ion trajectory mapping.*

Dual-Mode Detection System

The dynamic range of the ICP-MS system is extended from six to eight orders of magnitude in the HP 4500 by a newly developed dual-mode detection system. The electron multiplier used in the dual-mode system is a discrete dynode type operated in both pulse count and analog modes.

The block diagram of the dual-mode system is shown in Fig. 5. When an ion enters the electron multiplier, it hits the first dynode and a shower of electrons is generated. These electrons hit the next dynode, generating more electrons. Finally, the pulse generated is detected by the collector. This small signal is amplified and a measurable pulse signal is obtained. At this point, the output signal from the amplifier contains both electrical noise and the pulse signal. After the amplifier, the electrical noise is eliminated by a discriminator circuit and pulse signals higher than the discriminator voltage are converted to an ideal pulse shape. This pulse is measured as one count.

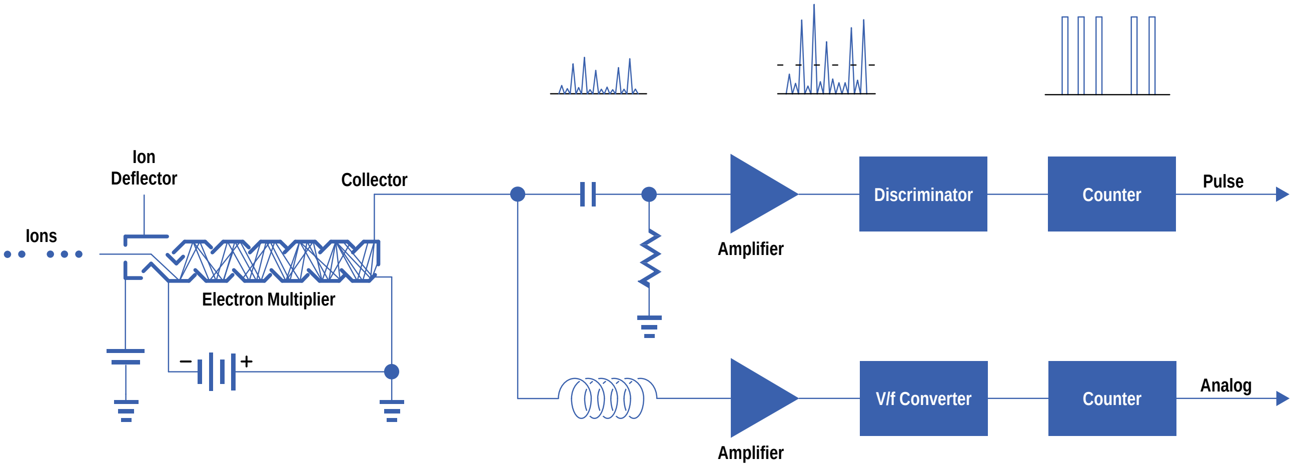


Fig. 5. *Block diagram of the HP 4500 dual-mode detection system.*

At very high analyte concentrations (>1 mg/l in the sample solution), detector saturation occurs, so the dual-mode system is automatically switched to analog mode and the ion current is measured. The ion current is converted to a frequency by a voltage-to-frequency converter and measured as counts per second.

The dual-mode detector system extends the maximum working range of the instrument up to approximately 100 mg/l. The appropriate mode for each isotope is selected automatically by the HP ChemStation operating software, and dual-mode data is acquired simultaneously, which is another first for ICP-MS. The great benefit is that samples containing a range of analytes at different concentration levels can be analyzed in a single analysis.

Without dual-mode operation, dilution, preconcentration, or other complicated sample preparation and steps would be involved. It is inevitable that as the process for sample preparation gets more complex, an increasing number of errors and contamination will occur. Contamination during sample preparation is always of concern when analyzing elements at trace levels.

The ShieldTorch System

Although the ICP-MS generates essentially monatomic, positively charged analyte ions, there are still several polyatomic ions such as ArO, ArC, and ArH, which arise mainly from the combination of the argon gas used to generate the plasma with oxygen, carbon, and hydrogen from the air and the samples. The main interferences are shown in Table I.

Table I
Typical Interferences in ICP-MS

Analyte	m/z	Interferant
K	39	$^{38}Ar^{1}H$
Ca	40	^{40}Ar
Ca	44	$^{12}C^{16}O_2$
Cr	52	$^{40}Ar^{12}C$
Fe	56	$^{40}Ar^{16}O$

The HP 4500 can be equipped with HP's proprietary technology called the ShieldTorch system, which reduces interference from polyatomic ions.[3] The electrical model of the plasma and the interface region is shown in Fig. 6. When the plasma is coupled with the RF coil inductively, the plasma has only a slight dc potential. However, there is capacitive coupling between the plasma and the RF coil, which creates a positive plasma potential oscillating at the radio frequency of the plasma source.

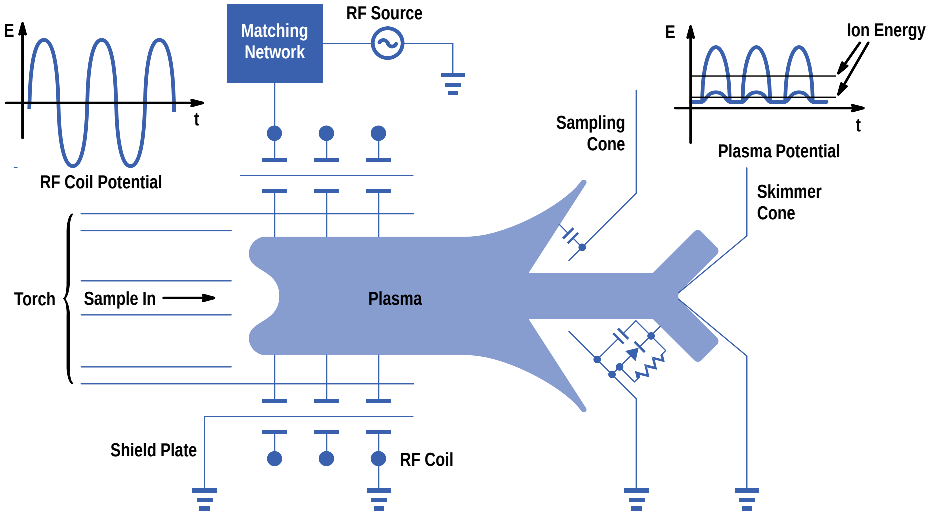


Fig. 6. *Electrical model of the plasma and the interface region.*

Within the plasma, positive ions and electrons exist, since the plasma temperature is high (6,000 to 10,000K). The numbers of positive ions and electrons are essentially equal, so the plasma is electrically neutral. Since the sampling cone is cooled by water, the plasma temperature decreases dramatically when the plasma comes close to the cone. Positive ions and electrons do not exist any more and the neutral Ar atom becomes dominant, creating a "sheath" between the interface and the plasma. Since the plasma potential is grounded to the interface and the vacuum chamber through the sheath, it acts as a condenser and the charge buildup around the sampling cone results in the formation of a discharge inside the first vacuum stage, commonly called the *secondary discharge*. The secondary discharge ionizes molecules such as ArO, ArH, and ArAr inside the first vacuum stage, giving rise to interferences with analyte ions at the same nominal mass.

When the ShieldTorch system is used, a shield plate is inserted between the torch and the RF coil, eliminating the capacitive coupling between the plasma and the RF coil so that the plasma potential is effectively reduced to zero. As a result, there is no longer a secondary discharge and polyatomic ions are not ionized behind the sampling cone. To reduce the polyatomic ions even further, the plasma temperature is reduced, since these polyatomic ions are also generated in the plasma itself. By lowering the plasma temperature, the ShieldTorch system reduces these interferences dramatically, resulting in improved detection limits down to ng/l or ppt levels for elements such as Fe, Ca, and K—typically three orders of magnitude better than without the ShieldTorch system. Typical spectra with and without the ShieldTorch system are shown in Fig. 7.

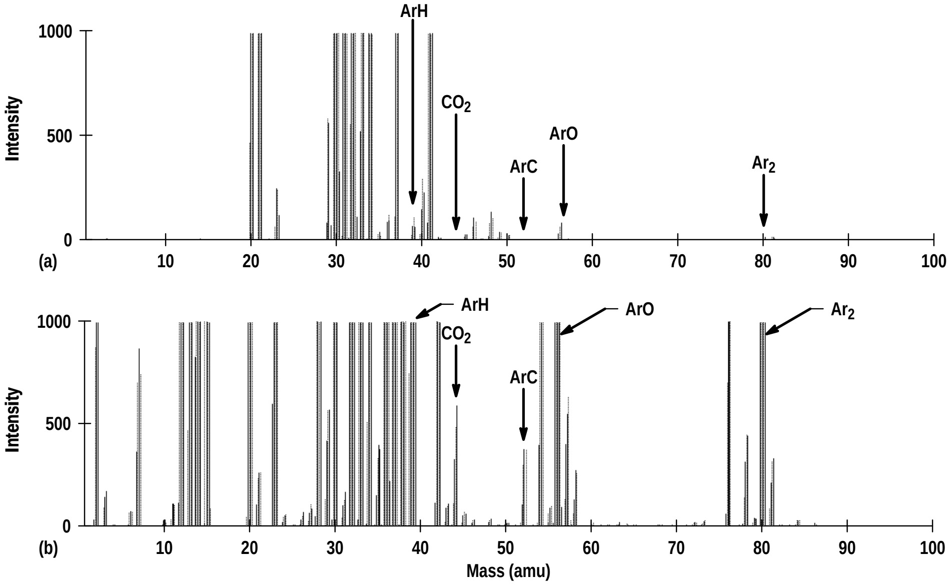


Fig. 7. *Typical spectra of deionized water (a) with and (b) without the ShieldTorch system.*

HP ChemStation Operating Software

The HP ChemStation operating software is easy to learn and use. All instrument parameters are controlled via the HP ChemStation, unlike traditional ICP-MS systems which were completely manual before the introduction of the HP 4500. An example screen from the HP ChemStation is shown in Fig. 8—this is the instrument control screen. A single click of the mouse starts the entire system, while system status is displayed in real time.

The HP ChemStation automates day-to-day operation by employing a suite of autotuning routines. Autotuning automatically optimizes the sensitivity, background level, and mass resolution and performs mass calibration. In the tuning screen, the user can select the tuning actions to be performed and the target values for sensitivity, oxide and doubly charged ions, and background. Three masses (typically one each at low, middle, and high mass) are simultaneously adjusted using a proprietary algorithm based on the simplex method.[4] Each ion lens voltage is changed to increase the signal of the element that has the weakest relative response (ratio of actual signal to target value) among the three masses until all the signals satisfy the target values. This allows less experienced operators to operate the instrument to its full potential.

Applications

The HP 4500 ICP-MS offers high-throughput multielement analysis with ng/l (ppt) or better detection limits, very small sample volume requirements, robustness, and ease of use. Therefore, the application areas for the HP 4500 are very wide, from the semiconductor industry in which the concentration of analytes is extremely low, to the environmental, geological, and clinical fields in which high-matrix or “dirty” samples are analyzed.

Semiconductor Sample Analysis. The trend towards pattern miniaturization and ultra large-scale integration (ULSI) in semiconductor devices requires the lowering of the level of metallic impurities present. In recognition of the need for higher-purity chemicals to meet the needs of submicrometer device production, the SEMI Process Chemicals Committee has proposed several grades for each chemical.

Hydrogen peroxide, H_2O_2, is widely used to remove metallic, organic, and particulate contaminants from wafer surfaces during the semiconductor manufacturing process. The H_2O_2 must be of extremely high purity to avoid contamination of the wafer surface by the cleaning solution itself. The specification for H_2O_2 (30-32%) in the SEMI Tier C Guidelines (the quality needed to produce ICs whose critical dimensions lie in the range of 0.09 to 0.2 μm or greater) stipulates that the maximum

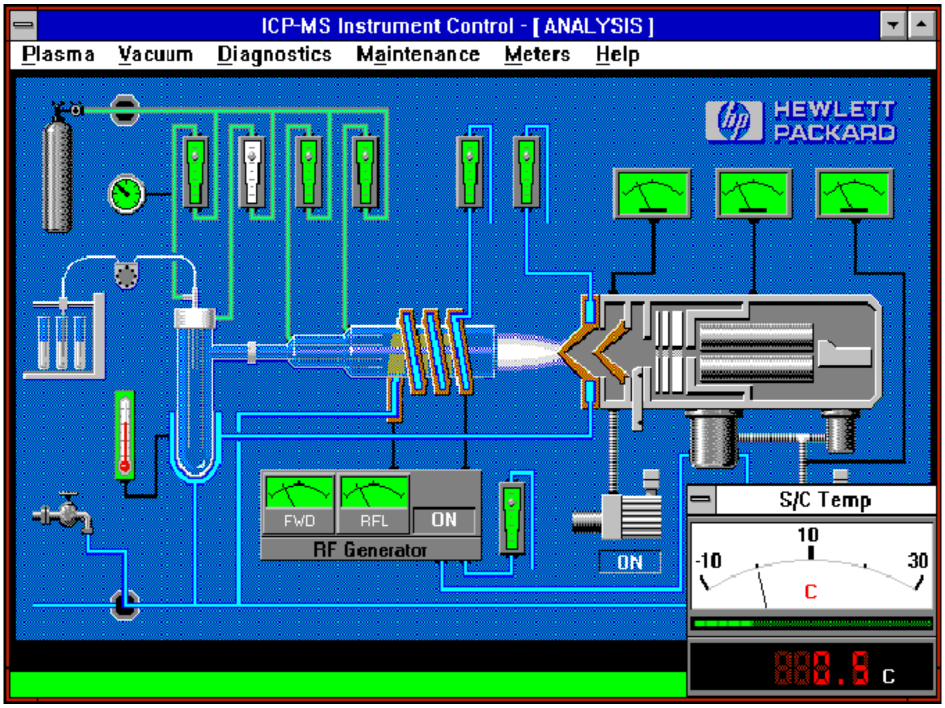


Fig. 8. *Example screen from the HP 4500 ChemStation.*

concentration of impurities should be 100 ng/l (ppt) for a suite of 18 metals. Table II shows the results of a quantitative purity analysis of H_2O_2 (30%).

Until now, recovery data presented to the Process Chemicals Committee by member companies has involved the use of ICP-MS followed by graphite furnace atomic absorption spectroscopy (GFAAS) for Ca and Fe. The HP 4500 with the ShieldTorch system can determine even Fe, K, and Ca at low ppt levels not normally possible by quadrupole ICP-MS because of interferences from polyatomic ions and isobars such as ArO, ArH and Ar.

Table II also shows the recovery results at the 50 ng/l (ppt) level. The recoveries of all of the elements were well within SEMI Tier C Guidelines, which stipulate that recovery data must be obtained showing 75 to 125% recoveries for all metals.

Environmental Sample Analysis. Concerns regarding safe levels of contaminants in the environment, particularly heavy metals, continue to grow. The requirement for analysis of more elements at ever-decreasing concentrations is exposing the limitations of currently used analytical techniques. ICP-MS is the only technique that offers the improvements in sensitivity that will be demanded in the near future. ICP-MS is approved for several environmental analytical methods including those developed by the U.S. Environmental Protection Agency (EPA).

Fig. 9 demonstrates the qualitative spectrum of river water standard reference material (SLRS-3). A large number of elements, ranging from lithium (Li) at low mass to uranium (U) at high mass can be clearly observed, even though the total analysis time was only 100 seconds. Table III shows HP 4500 ICP-MS quantitative results, which are in excellent agreement with the certified values. The dual-mode detection system allows the user to quantitate the analytes from a few tens of ng/l (ppt) to the mg/l (ppm) level.

Clinical Sample Analysis. The determination of toxic elements such as mercury (Hg), lead (Pb) and cadmium (Cd) in humans has been a critical issue in the field of clinical chemistry from the toxicology viewpoint. In addition, since recent biomedical research has shown that some elements at trace levels have specific functions in the biochemistry of living organisms, the determination of trace element concentrations in human beings has also become a major issue in the field of nutritional study. As a result, the analysis of toxic elements and also many trace elements in biological samples is required. The analyte concentration range is large, ranging from the trace levels normally found in the body to the high levels resulting from industrial exposure. Since medical treatment regimes for hospital patients depend on the analytical results reported, the analysis of biomedical samples is critical. Therefore, the need for fast and reliable analytical methods and instrumentation is paramount.

Table IV shows the HP 4500 ICP-MS quantitative results for human hair standard reference material (NIES No. 5) which was decomposed by a microwave sample preparation system. The concentrations of 11 elements analyzed were in good agreement for all the elements that had certified values (there is no certified value for As).

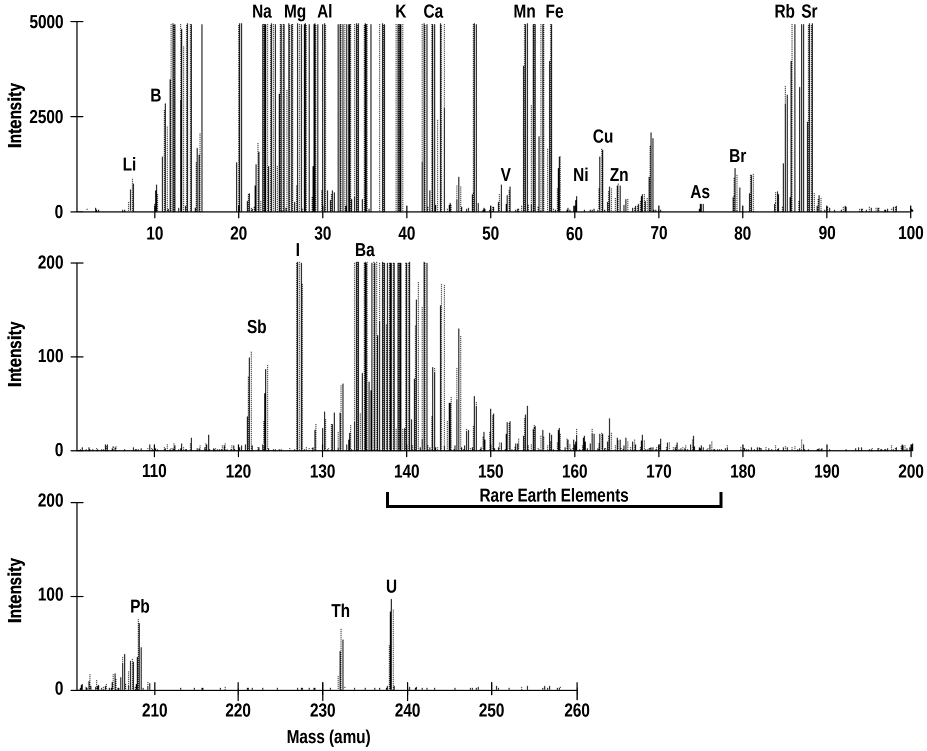


Fig. 9. *Qualitative spectrum of river water reference material.*

Table II
Quantitative Results for Hydrogen Peroxide (30%)

Element	Concentration (ng/l)	Detection Limit (ng/l)	Recovery (%)
B	188	4	96
Na	4.7	0.5	102
Mg	8	2	97
Al	9	3	102
K	not detected	0.02	101
Ca	34	4	109
Ti	14	2	98
Cr	2	1	101
Mn	1.3	0.1	102
Fe	9	1	106
Ni	6.5	0.6	98
Cu	2.6	0.4	102
Zn	8	1	101
As	12	0.7	115
Sn	4.4	0.5	102
Sb	3.1	0.5	104
Au	7	2	100
Pb	3.4	0.3	98

Table III
Quantitative Results for River Water

Element	Certified Concentration (µg/l)	Measured Concentration (µg/l, N = 3)
Be	0.005 ± 0.001	0.0051 ± 0.0004
Na	2300 ± 200	2260 ± 30
Mg	1600 ± 200	1450 ± 10
Al	31 ± 3	32.3 ± 0.5
K	700 ± 100	700 ± 30
Ca	6000 ± 400	5720 ± 10
V	0.3 ± 0.02	0.303 ± 0.004
Cr	0.3 ± 0.04	0.303 ± 0.003
Mn	3.9 ± 0.3	3.70 ± 0.07
Fe	100 ± 2	98.7 ± 0.5
Co	0.027 ± 0.003	0.0288 ± 0.0002
Ni	0.83 ± 0.08	0.769 ± 0.003
Cu	1.35 ± 0.07	1.39 ± 0.02
Zn	1.04 ± 0.09	1.01 ± 0.00
As	0.72 ± 0.05	0.697 ± 0.007
Sr	28.1*	30.1 ± 0.2
Mo	0.19 ± 0.01	0.193 ± 0.005
Cd	0.013 ± 0.002	0.0125 ± 0.0002
Sb	0.12 ± 0.01	0.127 ± 0.001
Ba	13.4 ± 0.6	13.3 ± 0.1
Pb	0.068 ± 0.007	0.060 ± 0.003
U	0.045*	0.0413 ± 0.0008

* Not certified, information value only.
N is the number of repetitions.

Table IV
Quantitative Results for Human Hair

Element	Certified Concentration (µg/g)	Measured Concentration (µg/g)	Detection Limit (µg/g)
Al	240*	220 ± 6	0.003
Cr	1.4 ± 0.2	1.72 ± 0.07	0.004
Mn	5.2 ± 0.3	5.47 ± 0.13	0.001
Fe	225 ± 9	219 ± 5	0.9
Ni	1.8 ± 0.1	1.87 ± 0.06	0.004
Cu	16.3 ± 1.2	16.7 ± 0.6	0.002
Zn	169 ± 10	171 ± 4	0.004
As	**	0.18 ± 0.02	0.02
Se	1.4*	2.4 ± 0.3	0.004
Cd	0.2 ± 0.03	0.21 ± 0.03	0.0002
Hg	4.4 ± 0.4	4.52 ± 0.15	0.003
Pb	6.0*	5.98 ± 0.11	0.0007

* Not certified, information value only.
** Not certified.

Solid Sample Analysis. Solutions and liquids are the normal sample types measured by ICP-MS. Solid samples are normally digested using mineral acids and analyzed as solutions. However, solid samples such as glass can be analyzed directly using the laser ablation system. The schematic diagram of this system is shown in Fig. 10. A sample is placed in the sample cell and ablated by the beam from a Nd:YAG laser operating at 266 nm. The fine aerosol generated is carried directly to the plasma by Ar carrier gas. Fig. 11 shows qualitative data for glass standard reference material (NIST 614). Group 1 and 2 elements, transition metals, rare earth elements, and actinides can be clearly seen from a two-minute analysis, even though the concentration of most elements was at the mg/kg (ppm) level or lower in the glass.

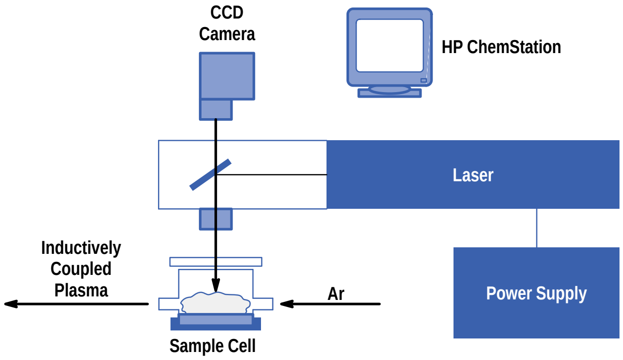


Fig. 10. *Schematic diagram of laser ablation system.*

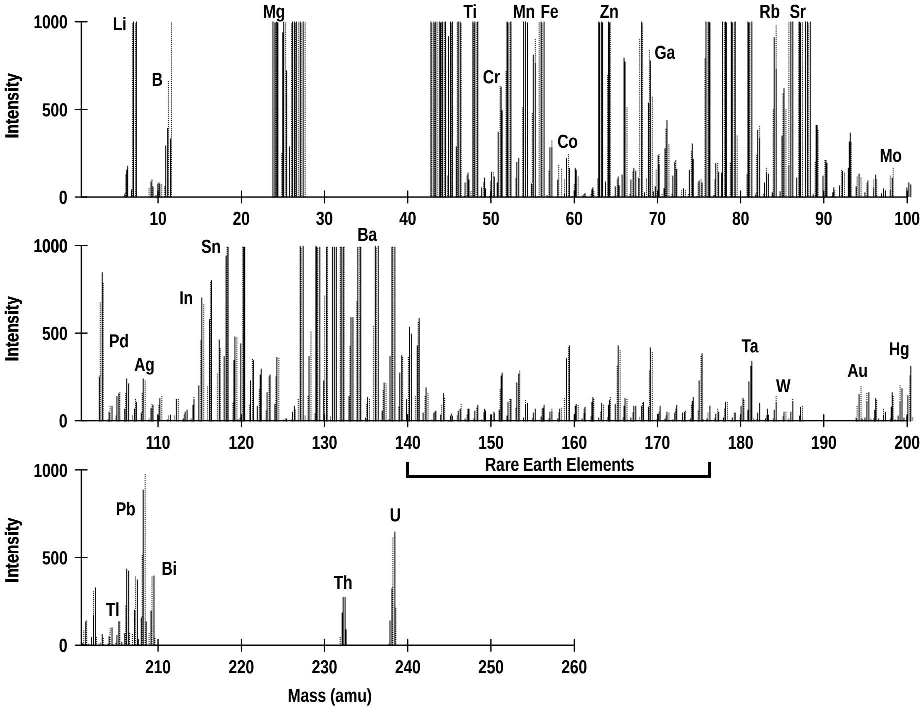


Fig. 11. *Qualitative spectrum of glass reference material.*

In addition to the bulk analysis capability shown, this technique also has the capability to analyze sample features and inclusions as small as 10 µm in diameter.

Speciation Analysis. Organotin compounds have been widely used for a variety of commercial applications. Trialkyltin compounds have been used for antifouling paints for ships and fish traps. Dialkyltin has been used for polymerization catalysts. Currently, there is growing concern about their effects on the environment. Methods to determine the species of tin (Sn) and the total amount of Sn present are required, since the toxicity of organotin compounds varies widely with the number and types of organic groups attached to the Sn atom. The combination of ICP-MS and chromatography has the ability to perform speciation analysis with high selectivity and sensitivity. Fig. 12 shows a chromatogram of six organotin compounds obtained by the HP 4500 ICP-MS combined with the HP 1050 liquid chromatograph. Each organotin compound was separated clearly within a total run time of 20 minutes. Detection limits obtained were 24 to 51 pg as Sn.

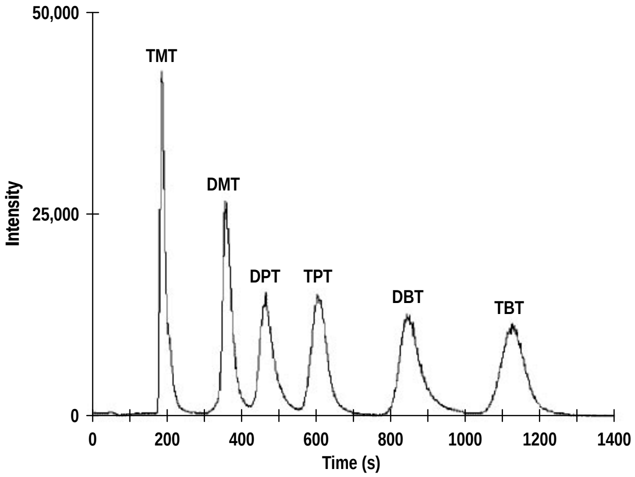


Fig. 12. *Chromatogram of six organotin compounds: trimethyltin (TMT), dimethyltin (DMT), diphenyltin (DPT), triphenyltin (TPT), dibutyltin (DBT), tributyltin (TBT).*

Summary

The HP 4500 ICP-MS offers high sensitivity, low background, a wide dynamic range, and the reduction of polyatomic ions, even though its benchtop size is only one fifth the size of the previous model. It is designed for routine use, easy operation, and easy maintenance. With these features, the HP 4500 is ideal for a wide range of applications in the semiconductor industry, environmental studies, laboratory research, plant quality control, and other areas.

Acknowledgments

The author wishes to acknowledge all of the different teams who contributed to the development of the HP 4500. Thanks to Don Potter and Toshiaki Matsuda for reviewing this paper and providing many helpful suggestions.

References

1. A.L. Gray, "The Origins, Realization, and Performance of ICP-MS Systems," *Applications of Inductively Coupled Plasma Mass Spectroscopy*, A.R. Date and A.L. Gray, editors, Blackie and Son Ltd., 1989.
2. K.E. Jarvis, A.L. Gray, and R.S. Houk, *Handbook of Inductively Coupled Plasma Mass Spectrometry*, Blackie Academic and Professional, 1992.
3. K. Sakata and K. Kawabata, "Reduction of fundamental polyatomic ions in inductively coupled plasma mass spectrometry," *Spectrochimica Acta*, Vol. 49B, no. 10, 1994, pp. 1027-1038.
4. M.A. Sharaf, D.L. Illman, and B.R. Kowalski, *Chemometrics*, John Wiley & Sons, Inc., 1986.

Audit History and Time-Slice Archiving in an Object DBMS for Laboratory Databases

Development of an object database management system allows rapid, convenient access to large historical data archives generated from complex databases.

by Timothy P. Loomis

The requirements for laboratory databases include many of the same features specified for other types of databases, including enforcement of a rigorous transaction model, support for concurrent users, distributed recovery capabilities, performance, and security. However, the requirements differ from most databases by the emphasis on saving a complete and recoverable record of historical data for some types of data. This requirement comes from the regulatory overseeing authority of the pharmaceutical industry by organizations such as the U.S. Government's Food and Drug Administration or Environmental Protection Agency, and often, the legal importance of the data (patent law). Some examples of historical data in a chemical laboratory include previous values of test results, designated reviewers and approvers of data, methods of analysis, and ingredients used to produce a product. It is necessary to be able to determine when this data changed, who changed it, and why a change was necessary.

Most laboratory database systems have tried to deal with historical data by adding complex logic to the application code to record and retrieve historical data in special tables that are added to traditional relational database schemas. While this technique works for simple schemas with a few objects that need to be monitored for change, its complexity overwhelms development, testing, and support efforts for more realistic databases. In short, it does not scale to the complex databases needed for the future.

Keeping track of historical data became a critical design factor when the HP ChemStudy product was being developed in the laboratory information management system program in HP's Chemical Analysis Solutions Division. HP ChemStudy controls all the information used in multiyear projects that determine the expiration dates on drugs. The database is complex with 128 types of application objects interconnected through numerous relationships. It is necessary to be able to reproduce the contents of objects and the state of their relationships at any time in the past to satisfy regulatory requirements.

Our solution to the historical data challenges of laboratory databases has been to develop a database management system (DBMS) that provides built-in support for historical data for any object and for groups of objects that are connected through relationships. The simplicity and extensibility of this system are possible because we have developed a pure object DBMS (ODBMS) in which relationships are themselves objects. Although the ODBMS provides many advantages for applications development, this article will concentrate on the issue of historical data.

The ODBMS is implemented in C++ on the HP-UX* operating system and Windows® NT.

System Overview

Before considering the details of how historical data is managed in the database, we need an overview of the distributed ODBMS to understand how an object is created and stored. While this modular system can be configured in many ways, Fig. 1 presents an example configuration that is used in the HP ChemStudy product.

In Fig. 1, a client is a process that incorporates C++ class code that defines application objects. While the object created by the application can contain any data needed in the application, the object is managed (locked, updated, saved) through the services of the generic object manager module. The object manager also controls logical transactions (commit and rollback) and provides save points and other DBMS functions. At the object manager level, all objects are treated alike and no changes are required to support any new application object types. The client may have a user interface (shown as a graphical user interface (GUI) in Fig. 1) or it could be an application server with code to support its own clients.

The object manager can connect to one or more object servers that control a database. The ability to connect to multiple object servers makes the system a distributed DBMS and necessitates a two-phase commit protocol to ensure that a transaction affecting multiple databases works correctly. The distributed capabilities of the ODBMS are employed for archiving operations (described below) and for integrating data from multiple active databases.

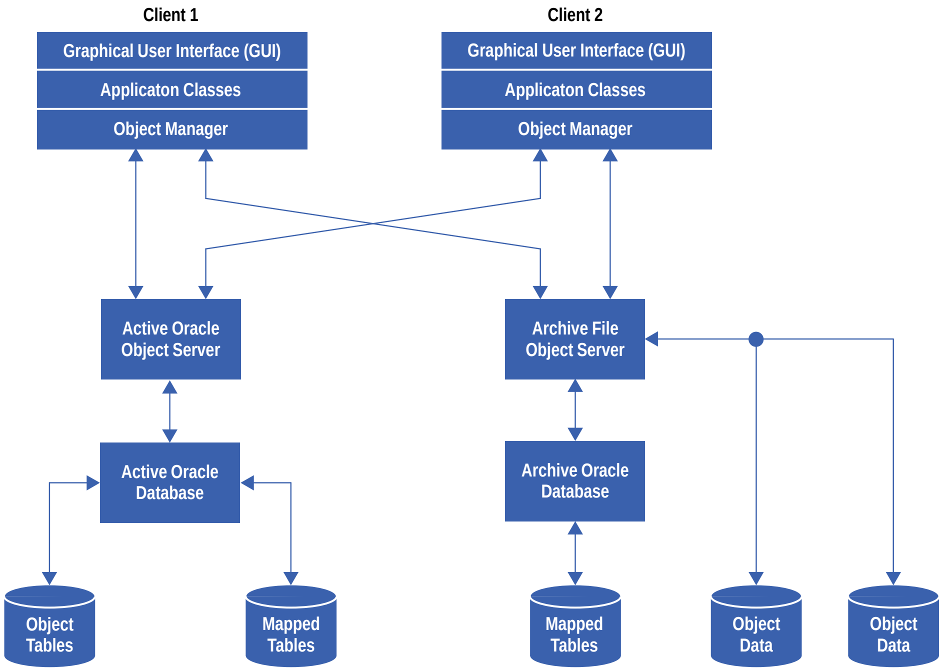


Fig. 1. *Example ODBMS configuration.*

Currently, we provide two types of object servers which differ only in the driver code module that stores object data. From the point of view of a client process, there is no difference in the way an object is treated. The Oracle object server stores an object in Oracle tables while the file object server stores the object in one or more redundant file structures as object data. While the file version is faster than Oracle for read and write by a factor of 30 to100, some customers prefer the Oracle version because it conforms to their corporate information systems requirements. The file version also stores data more compactly and is ideal for embedded databases that are not visible to users and for databases in which the speed of storing and retrieving data is critical. Because the object data stored by either type of server is binary, multimedia data or a binary file can be stored by breaking the data into objects. Objects are also useful for processing a large binary data file in clients that do not have enough memory to hold all the data at once.

Laboratory databases become so large that it is necessary to remove old data periodically from the *active* database and place it in some type of *archive* for long-term storage. Most systems have used a special storage medium for archived data and require that the data be *dearchived* back to the active database for review. Instead, we use the distributed capabilities of the ODBMS to transfer data from the active database to an archive database as a simple distributed database transaction. The archive database can then be taken offline without limiting current operations. Fig. 1 shows an Oracle server being used for the active database and a file version being used for an archive database.

The object database provides access for C++ object applications but lacks facilities for ad hoc queries and reports that can be customized by a customer. To accommodate ad hoc queries and report writers, a collection of mapped tables can be created that provide a more traditional relational database schema of the application data. Each type of C++ object can be mapped to its own table in the map schema when it is inserted or updated, but it is always read by the application from the object database. In practice, only some data in selected objects is mapped. This object-relational DBMS combination has proven to be very successful at providing the customer with reporting flexibility, while preserving the speed and simplicity of a pure object system for the application code.

An example of mapping is shown in Fig. 2. The example considers three objects of three different types: Dept, Emp and EmpList (relationship). A client connected to the object server transports binary objects to and from the server cache. Except for objects newly created by a client, all objects in the cache have persistent counterparts in the object database and are read into the cache from this database. All objects are inserted or updated in the object database during the commit operation. At the option of the application designer, selected data from an object can also be mapped to the map database as shown for the Dept and Emp objects. The EmpList relationship object is not mapped in this example. Relationships are usually defined using *foreign keys* in relational schemas.

We can see from this overview that an object is a bundle of data that can exist simultaneously as a C++ object in multiple clients, as an object in the cache in the object server, as object data in a database, and as mapped data in a relational table. Managing the relationships among these multiple representations of an object requires adherence to a rigorous transaction model. Many of the features necessary to deal with historical versions of an object are extensions to controls that already exist for object data.

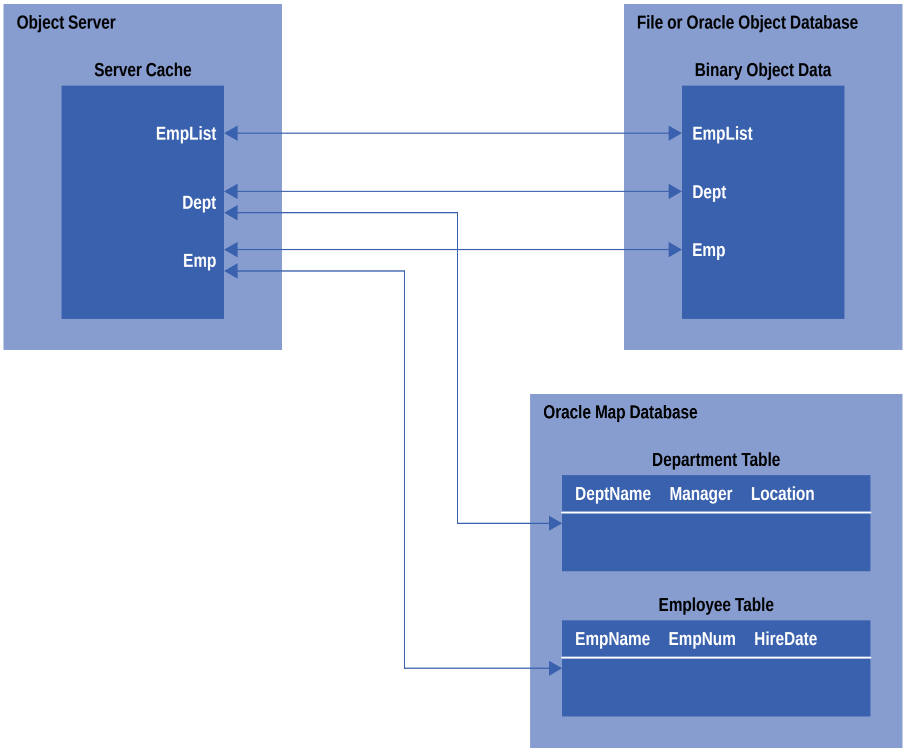


Fig. 2. *Object mapping example.*

Auditing Laboratory Data

There is more to a database data item than a value that can be retrieved. For example, that value was created by someone or some calculation, it may have been converted from a string representation with a specific precision, it was created at some date and time, it may have some application-specified limits that cannot be exceeded, and so on. Moreover, the current value may have replaced a previous value, requiring a justifying comment, and it may be necessary to retrieve all earlier values of this data item. It has long been a requirement for laboratory databases to maintain this type of information associated with a laboratory measurement and to record a history of changes to the measurement. We generally refer to the process of maintaining a record of a value and its associated information through time as *auditing* or maintaining an *audit trail*. In the context of an object database, auditing means keeping a record of the history of an object and objects associated with it through relationships.

Auditing database data has generally meant keeping a separate record or audit log of selected changes made to the database. For example, Oracle provides the capability to audit user, action, and date for access to selected object types but requires a user to write triggers to record changes to data values. While this straightforward mechanism does accomplish the task, its use for large and complex databases rapidly generates huge volumes of data that require sophisticated searching to identify particular changes of interest. A simple audit log of database changes is practical only if one hopes that it will never be needed! Audit logs are routinely needed in the pharmaceutical industry and will soon be a common requirement for other industries subject to regulatory oversight, such as software development processes subject to ISO validation. Searching through a huge audit log is not a reasonable way to answer an auditor's questions about the history of an object that may contain, or be associated with, hundreds of component objects.

The alternative to an external audit log is a DBMS that has an intrinsic method for auditing an object and its relationships. In the next section we discuss general methods developed to audit selected classes of composite objects stored in an ODBMS so that the audit data can be retrieved easily.

The subject of temporal databases has received considerable research attention directed mainly toward extending the relational model and providing time-based query methods.[1,2] The implementation presented here differs from these models principally by:

- Using an object model
- Using relationship objects together with lock-and-update propagation to synchronize the time history of related objects, rather than attempting to deal with the more general problem of "joining" any set of objects
- Being a working implementation for audit-trail applications that deals with load errors and numerous practical programming problems.

Commercial extended relational databases such as Illustra[3] are beginning to provide some time-based capabilities for specialized data.

Example Schema

Auditing an object is complicated by references to other objects. Consider Fig. 3, which shows an abbreviated class schema for a division of a company containing departments, department offices, and employees within departments. Relationship classes (objects) derived from the class list are shown explicitly in this diagram because they are important in auditing. (For clarity all lists are shown as separate classes rather than as inherited base classes). A reference to another object is shown explicitly as an arrow in this diagram because we will be concerned with the details of propagation of information between objects. A line terminated with a dot (as in IDEF1X or OMT modeling)[4] indicates that references to multiple objects can be stored. An A in the lower-right corner of a class indicates that objects in that class are audited.

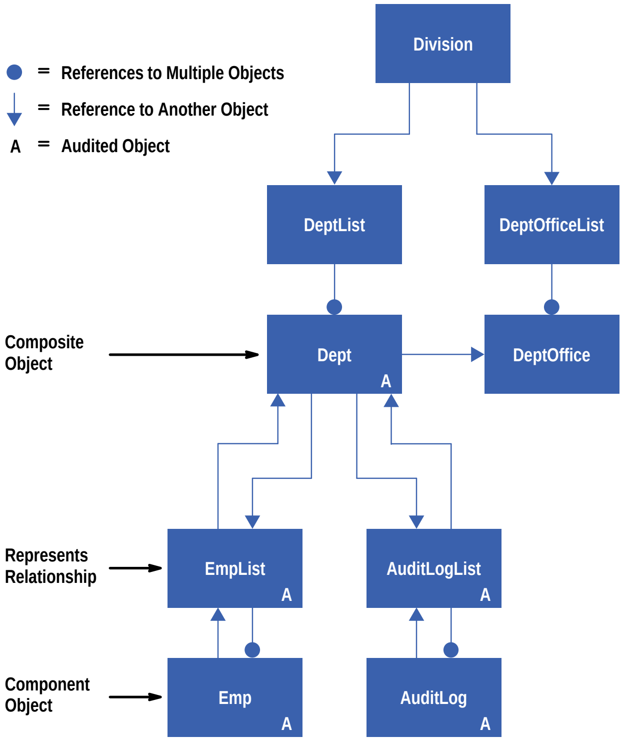


Fig 3. *Example class schema.*

Composite Objects. Audited relationships should be used to contain the components of a composite object. A composite object is one that can be considered to logically contain other component objects in the application. More precisely for our purposes, a composite object can be defined as one that should be marked as changed (updated) if component objects are added, deleted, or changed even if the data within the composite object itself remains unmodified.

In the example of Fig. 3, we will consider a Dept to be a composite object because it logically contains Emp component objects. An EmpList object is the relationship or container connecting the composite and its components. We consider Dept to be a composite object in this example because we implicitly include all the employees in the department as part of the department and want to consider the department to be modified if there are any changes to any of the employees. Alternatively, we could have considered Dept to exist independent of its employees. Clearly we can sink into the dark waters of a long philosophical discussion here (If you change the engine in the car is it the same car?), so the design is best approached physically. The basic question is whether examination of the history of a composite object should reflect changes to its component objects. For many complex objects in our products the answer is yes.

Two references are necessary for an audited relationship. References traversed from Dept to Emp are called *component references* and the reverse references are called *back references*.

Audited and Nonaudited Objects. As exemplified by the use of classes derived from the list class in audited and nonaudited relationships, auditing can be specified on a subclass or an individual object. Moreover, it is permissible to turn auditing on only after some event in the life of an object. For the moment, we consider only the case where an object in an audited class is audited from inception.

We see in Fig. 3 that objects of the composite Dept class should be audited from creation but that DeptOffice and Division are never audited. Semantically, this design means that the history of a Dept object, including the composition of all of its component Emps, can be retrieved at any stage of its history. In contrast, the DeptOffice for the Dept and the list of Depts in the Division can be retrieved only for their current values.

Audit Mechanism

Auditing Objects. Auditing an object means that all images of the object must be maintained in the database, starting with the image that existed when auditing was turned on. In contrast, only the latest (current) image of a nonaudited object is retained. Note that when an audited object is to be written to the database, the decision to replace the old image depends on whether the old one was audited. Successive object images generated through update will be referred to as *revisions* of the object, whether the object is audited or not. The revision number is used by the ODBMS to ensure that a client is working with the correct image of an object. There can be only one current revision of an object and only the current revision can be updated.

The term *version* is used for the concept of distinguishing variations of an object that can all be current. For example, different versions of a glossary can exist for different languages but each version may undergo revision to add terms or correct errors. An object is also marked with a *commit timestamp*, which is exactly the same for all objects in a (possibly distributed) transaction. These attributes of an object, along with its identifier and other data, are contained in a header that is prepended to the object in the database and maintained separately by the C++ object in the client object manager.

Auditing Relationships. Auditing relationships requires some mechanism for recording the history of the relationship. Rather than implement a database relationship mechanism and audit it separately from auditing objects, it makes sense to implement relationships as objects themselves. Auditing a relationship is then no different than auditing an object.

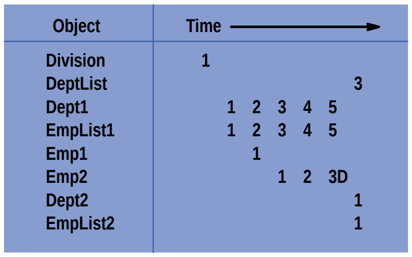
Deleting Audited Objects. Deleting an object becomes complicated when the object is audited because the object still exists in the database until the delete is committed. The delete action must be represented in the database somehow, so that the timestamp and revision number marking the end of its life are available. We use a pseudo-object for this purpose. Archiving audited objects, or portions of their history, may involve actually referencing and loading these pseudo-objects representing the delete operation.

Update Propagation. An important objective of the audit mechanism should be to update the minimum amount of information to document a change fully. For this reason we reject the simple "archive copy" approach to auditing whereby the entire composite object is copied each time a component changes. Thus, we should not simply make a copy of the entire Dept composite hierarchy just because an Emp changed because this produces a huge amount of redundant data.

Auditing a composite hierarchy is implemented in our system by propagating the update of a component through the relationship and composite parent objects using back references. For example, updating member data in an Emp object will trigger an update in the EmpList and Dept but will not necessitate an update or copy of other Emps or of other components of Dept. It is necessary to mark composite objects as updated even though their member data has not changed because the composite they represent has changed. Note that there is nothing to be gained by updating a nonaudited object that references an audited one because it does not have a history corresponding to the past history of the referenced object. Therefore, for example, Division is not updated when Dept changes.

It is impractical to expect programmers to follow these back references each time they update an object. It is also asking for bugs to expect them to qualify the propagation correctly according to audit state and update type. We have solved this problem by incorporating back references implicitly within relationship objects and component objects. The object manager code propagates updates automatically as appropriate.

The audit contents of a database can be illustrated using Fig. 4, an example history of a part of the example schema in Fig. 3. The number shown for each object at a particular time is its *revision number*, a simple count of the number of database transactions that have changed the object. We see that Division has not been changed since it was created. DeptList was created at the same time as revision 1 but has been modified twice since then (when Dept1 and then Dept2 were added). Since DeptList is not audited, only the last revision (revision 3) exists in the database.



Object	Time →						
Division	1						
DeptList							3
Dept1		1	2	3	4	5	
EmpList1		1	2	3	4	5	
Emp1			1				
Emp2				1	2	3D	
Dept2							1
EmpList2							1

Fig. 4. *Example object history.*

The behavior of audited objects is different. Dept1 and its EmpList1 were added to the DeptList as revisions 1. When Emp1 was added to EmpList1, the update was propagated to Dept1 as well as EmpList1 so that the revision of the composite object Dept1 reflects a change to one of its components. The same thing happens when Emp2 is added. Note that Emp1 is not updated in this operation, nor does the update propagate to the nonaudited DeptList. A subsequent update of Emp2 (revision 2) similarly causes propagated updates to EmpList1 and Dept1. To make the example interesting, Emp2 has been deleted, represented by the creation of the pseudo-object with revision number 3D. This object really exists in the database as a marker of the end of the life of Emp2 (figuratively, we hope). Just as for an update, this delete operation causes an update of EmpList1 and Dept1.

Lock Propagation. For pessimistic concurrency models it is necessary to acquire an explicit lock on all objects to be updated at commit. Consequently, the object manager should propagate exclusive locks in the same way that it propagates updates and be able to deal with restoring locks to their original type if the propagation should fail partway through the propagation.

Audit Log. Another objective is to summarize changes to the composite Dept object in one place. In this example, suppose there are several changes to each of three Emps and to some other components (not shown) in a single transaction. The update mechanism records the fact in the Dept object that something changed in at least one component object in this transaction, but we need the AuditLog text object to itemize the specific changes bundled in that transaction. Fig. 3 shows a list of AuditLog objects hanging from Dept. Each AuditLog object summarizes the changes for the composite Dept object during a transaction. From the user's point of view, a convenient implementation is to generate one-line entries in the log automatically for each change the application makes to a component object or the composite object, and then require the user to add only a summary comment before commit.

Object Access

Revision and Time Retrieval. An audited object can be retrieved from the database by specifying either a specific revision of the object or by specifying an absolute time and finding the object that was current at that time. A special time token represents current time (also known as NOW in the literature), corresponding to the most recent object revision. Accessing objects by absolute time requires that the commit timestamp of an object be determined so that it corresponds correctly to the actions of multiple clients in a distributed database environment. A consistent source of time must be available to all clients and time must be specified precisely enough to distinguish two transactions on a fast network.

An example is the best way to explain why both access methods are needed. A common way to query the database history in Fig. 4 would be to locate the current Dept1 and then ask to see each of its previous revisions. Retrieving revision 5 of Dept1, the system would use its commit timestamp to retrieve revision 1 of Emp1 and not find Emp2 because it was deleted at this time in EmpList1. Moving back in time to revision 4 of Dept1, its EmpList1 would recover revision 1 of Emp1 again and also find revision 2 of Emp2. Instead of starting with the current revision of Dept1, the initial query could have specified any absolute time, say one somewhere between revisions 2 and 3 of Dept1 to find revision 2 of Dept1, then the commit timestamp of revision 2 would be used to find component data.

Multiple Revision Management. A consequence of auditing objects is that multiple revisions of the same object can exist in the client cache at the same time. This presents a number of practical problems for application programmers who need a simple mechanism for specifying the correct object revision to access. We have found that extending the meaning of locking an object to include cache management of old and current revisions of an object as well as the traditional meaning of granting an explicit lock on the object is a practical solution to this problem.

Accessing Objects through References. Mixing audited and nonaudited objects in the same application exposes the implementer to numerous opportunities to generate run-time database load errors. Despite the problems of a schema with both audited and nonaudited objects, it is often necessary to mix the two to avoid creating impractical quantities of data in the database. A few referencing rules, if they can be enforced, solve the problems.

- Rule 1: Current access to nonaudited objects. A nonaudited object must always be accessed as a *current-time* object, meaning the latest one available from the database. For example, all revisions of Dept use current time when accessing DeptOffice because old revisions of DeptOffice do not exist. If an old time were specified in the access request and DeptOffice had not been changed, the access would succeed, but a few minutes later, after DeptOffice had been updated by another client and its timestamp had changed, the same request would fail!

  This rule is simple enough but does introduce some opportunities for apparently inconsistent behavior. For example, if a report generated for a Dept uses the reference to DeptOffice to include its room number, the same report repeated later on the same revision of the Dept could have another room number if DeptOffice had been changed. Worse, the DeptOffice could have been deleted from the Division causing a load error. These apparent problems are not the fault of the database system but rather intrinsic in the heterogeneous schema. They are solved either by auditing DeptOffice or by indicating that DeptOffice is deleted by status data within the object rather than deleting the object.

- Rule 2: Qualified access from nonaudited to audited objects. As explained above, an access time or specific revision number must be specified when accessing an audited object. For example, the Division can reference a Dept in three ways: by specific revision, by current time (meaning the latest revision), or by absolute time. In practice, a user does not generally know a specific revision of the Dept object or a specific commit timestamp. Therefore the most useful access times are current time or an absolute time the user specifies for some reason.

A continuing complication when accessing audited objects is that the object exists at some times but not others. For example, if we delete the Dept when it is transferred out of the Division, we can't simply delete it from the DeptList because we may need to access the old Dept information in the future. Thus, the reference to a Dept should be tested for accessibility before we try to load it for a specific time to avoid a load error. These problems are solved if we simply audit the DeptList and Division.

- Rule 3: Self-timestamp access between audited objects. The easy and foolproof way for an audited object to access another audited object is for it to use its own commit timestamp. Furthermore, it is permissible for an audited object to drop a reference when the object is deleted (or for any other reason) because its previous revisions will still have the reference. However, there are some complications.

  It may be necessary for an object to access the same object in different ways. Suppose the DeptOffice in Fig. 3 were audited. If we create a report on a revision of Dept and include DeptOffice information, the method in Dept creating the report should use its timestamp access to DeptOffice to get contemporaneous information. However, if a Dept method is programmed to update the DeptOffice, say with its identification information, it is important that the current DeptOffice be accessed, because only a current object can be updated. As long as the Dept is updated first, timestamp access can be used for both but it will not work if the update in Dept is marked after accessing DeptOffice. In general, it is safer to code current access explicitly when updating a referenced object.

Midlife Changes of an Object. It is permissible to change an object from nonaudited to audited at some time in its life. Probably the most common reason to do this is to avoid generating large amounts of data while an object is in some draft stage and being updated frequently. Keep in mind that the object can be a composite object hierarchy encompassing hundreds of large objects. Only after some approval stage does the application really want to track the life of this composite construct.

Making an object audited may change the rules it uses to access component objects and propagate updates. By implementing these mechanisms in object manager utilities, the change can be made transparent to most application developers.

Schema Constraints

The previous discussion leads to a simple rule for auditing classes in a schema: audit the components and relationships if the composite is audited. For a composite object to truly represent the state of a component hierarchy, all the components and component-composite relationships beneath the composite must be audited when the composite is audited. Only then will locks and updates be propagated correctly and can the composite use its timestamp to access its components reliably.

For example, Fig. 3 shows the AuditLog as audited even though we expect to create only a single AuditLog revision for each transaction. Marking it audited follows the rule to acquire the programming simplifications enumerated above. There is really no penalty in this case, because storing one revision of an audited object takes no more room than storing one revision of a nonaudited one.

There are reasons for breaking this rule. In large realistic systems (in contrast to small demonstration ones) we face realistic constraints on space and often somewhat ambiguous application requirements. As an example, consider DeptOffice which is marked as nonaudited in Fig. 3. If we assume that there are good application reasons for not auditing DeptOffice, we have to carefully access the references between Dept and DeptOffice according to the complications discussed above and accept the apparent inconsistencies that these relationships may produce.

Database Storage

Object storage implementations are beyond the scope of this article, but it is worthwhile to mention a couple of considerations. First, it is not necessary to have a specialized database to store audited objects. We have implemented an auditing database that can use either Oracle tables or our own file storage manager. The main complications are:

- Providing an efficient access method that will find an object current at a time that does not necessarily correspond to a timestamp
- Handling pseudo-objects representing delete.

Second, it is advisable to provide efficient access to current objects. Because audited objects are never deleted it is not unreasonable to expect hundreds of copies of an object in an old database. Most applications will primarily access the current revision of an object and have to stumble over all the old revisions unless the storage manager distinguishes current and old audited data. It may be worth introducing some overhead to move the old revision of an object when a new revision appears to maintain reasonable access efficiency.

Some object database systems map object data to relational tables. The relational system can represent the primary object depository or, alternatively, only selected data can be mapped to enable customers to use the ad hoc query and report-writing capabilities of the relational database system. Extending these systems to handle audited data simply requires adding a revision number, timestamp, and object status code to the mapped data. The ad hoc user should be able to formulate the same type of revision and time dependent queries of the relational database as a programming language does of the object database. The status is necessary to distinguish old audit data, current objects, and deleted pseudo-objects.

Archiving

A lot of database data is created very rapidly in auditing databases. At some point some of it must be moved to secondary storage as archived data. As usual, auditing database systems pose special challenges for thinning data without corrupting the remaining objects.

What Is an Archive? Several types of archives are possible. One common repository is a file containing object data in a special format and probably compressed. Data is moved to the archive using special archive utilities and must be *dearchived* back into the active database for access using the same special utilities. This method maximizes storage compactness but pays for it by a cumbersome process to retrieve the archived data when needed. Another possibility is to move data to a separate data partition (table space) that can be taken offline. Access to the archived data might require dearchiving or, if the complexity is tractable, unioning the archived data with the active data in queries.

At the other extreme is the use of a distributed database system to connect the active and (possibly multiple) archive databases. The archive medium, then, is just another database that should have read-only access (except during an archive operation by system utilities). A distributed database system connects the active and archive databases during the archive and dearchive processes, allowing the data to be moved between databases as a distributed transaction. This is the method we have chosen to use in our products. A distributed archive system allows continued growth of archived data while retaining reasonable access times when necessary. Another advantage is the reliability of the archive and dearchive processes because they are a distributed transaction subject to two-phase commit protocols and recovery mechanisms. Finally, it is possible to access archived data automatically without dearchiving if the archive database is on line. This indirect access feature is explained more fully below.

Archiving Entire Objects. The first mechanism for thinning data is to remove objects that will no longer be modified. Generally status within the object indicates when this state of life has been achieved or, perhaps, just the time since the object was last modified is sufficient. Can we just remove all revisions of the object from the active database and put them in an archive record?

The first problem is simply finding the old object because it might have been deleted. It might not even be in the list of current objects in a nonaudited list. For example, in Fig. 3 we had better not delete a Dept or delete it from the DeptList until the time comes to archive, or we will never be able to find the orphaned object. When archiving a Dept it would be an oversight to archive just the current Emps. What about the one that was deleted earlier in the life of the Dept and is referenced only in an old revision? Fig. 4 shows this to be the case for Emp2 in Dept1. Evidently, it will be necessary to search all the old revisions of all composite objects just to identify all candidates for archiving. A special key field to identify all components of a composite to be archived is a big help here.

The second, admittedly mechanistic, worry is how to remove an audited object, since deleting actually results in inserting a new pseudo-object, and we can't even access a deleted object at current time! Presumably some additional code design and implementation provides a mechanism for actually removing an audited object and all of its old revisions, as well as accessing deleted objects. This operation is called *transfer out* to distinguish it from deletion. Similarly, the database must allow *transfer in* of multiple object revisions, including pseudo-objects representing delete.

Now we can move on to the problem of other objects that access the archived object. Because archiving is not deleting, objects that reference an archived object need to retain these references in case the archived object must be accessed in the future. For example, we should retain an entry in the nonaudited DeptList for an archived Dept object even if it is not immediately accessible. One solution is to place a status object on each relationship in the DeptList. This status object can contain archive information. Another solution is to replace the archived Dept object (and its components) with a placeholder object that marks it as archived and could also contain archive information. Unless we want to start changing references in old objects, this new placeholder object will have the same OID (object identifier) as the old one. A variation on the second method is to record archive information within the ODBMS and trap references to archived objects.

These solutions work if the referencing object is not audited. But what if it is audited? Updating the current object or marking the status of its reference to the archived object may be satisfactory for current time access but will result in a load error if older revisions attempt to access the object using references that were valid back when the old revision was current. Unless we want to start updating old revisions (a scary idea if we want to trust the integrity of audited data), the archiving mechanism must handle these old references between audited objects without modification or qualification of the old references. The general solution to the archive-reference problem probably must be implemented at the database level. The database lock or load mechanism must be able to distinguish a reference to an object that never existed for the revision-time criteria specified from one that existed but is now archived. The user must be notified that the data is archived without disrupting normal processes.

Incremental (Time-Slice) Archiving. In some applications it may not be practical to archive entire objects. The life time of some archivable objects (actually composite objects with thousands of component objects) in some systems can be as long as five years, making archiving the object theoretically possible at some time but not very useful for reducing online data on a monthly or yearly basis. Clearly a mechanism for archiving just the aged revisions of objects is necessary in these applications.

The best way to specify incremental archiving is on a time basis, because time can be applied uniformly to all objects. In this scenario we could specify a list of candidate archive objects and a threshold archive time, such that all revisions of these objects found with a commit timestamp equal to or earlier than the archive threshold would be moved to the archive. Well, actually, not quite all of them! Since we must satisfy requests by the active database for the object at the threshold time, we must keep the one object revision with a commit timestamp before the threshold time because this revision is current at the threshold time (unless the object was deleted, of course).

To implement this incremental archive mechanism, as described so far, the system must keep track of the threshold time and archive information about the revisions of each object. Attempted access to revisions extant before the archive time should receive an archive error and perhaps supply the archive information so that the user knows where the data can be found.

In this scenario, archiving probably is not a one-time operation. What do we do with the remaining revisions of the object when the archive operation is repeated a month later, specifying a threshold archive time one month later than that in the previous operation? From a bookkeeping point of view, it would make sense to simply append the new archive revisions of an object to the old ones in the archive and update archive information in the active database. In practice most customers will not find this method any more acceptable than filing tax records by subject rather than date. Most archive time slices will be kept as an archive record labeled by the date range of the data it contains; it could be a tape collecting dust in a rack. If we needed to append to an archive whenever more revisions of a long-lived object were archived, the archive operation would eventually require mounting many archives. Thus, a practical archive mechanism must allow various revisions of an audited object to be scattered in multiple archive databases.

If a single object can be contained in multiple archives, we must know which archive might contain the requested data. Moreover, it would be nice to guarantee that the load request could be satisfied if the archive were made available. A customer will be upset if the archive supposedly containing the missing data is found and mounted and then the customer is told that the data still missing! Thus, it will be most convenient to retain in the active database complete information about the range of revisions and commit timestamps of an object in each archive. This archive record, called an *archive unit*, contains information about the continuous sequence of object revisions of an object that were transferred in the archive operation.

An example of time-slice archiving is presented in Fig. 5. An audited object identified by ObjNum 101 has created 10 revisions in the active database. At some time in the past, an archive database was created, designated as 1995 here. The first time-slice operation moved revision 1 to the archive database and left an archive record in the active database. The archived object acquired a new identifier, shown as 23, because an ObjNum is unique only within a single database. Subsequently, another archive operation moved revisions 2 and 3 to the same database, leaving another archive record. The following year, another archive database was created and revisions 4, 5, and 6 were archived here.

Dearchiving and Archive Access

Dearchive Operation. The process of dearchiving is just the reverse of archiving, whether the archive medium is a compressed file or a remote database. If incremental archiving is used and an archive record is maintained in the active database, it reduces bookkeeping to dearchive an archive unit (group of continuous object revisions) and remove the archive record from the active database. It is also necessary to dearchive archive units continuously from the youngest one to the target one to ensure the integrity of the time-retrieval mechanism. There must be a continuous revision sequence from the current timestamp to the timestamp preceding or equal to the target timestamp.

Indirect Access to Archive Data. Of greater interest is the possibility that dearchiving may not be necessary. If archived data resides on archive databases in a distributed database system, it is possible for a sophisticated object manager to access archived data in remote archive databases and integrate it with the active data. Important advantages of this mechanism are:

- Reduced resources for the active database because dearchiving is not necessary
- Transparent access to archived data by ordinary users
- Reduced administration, because the archive and dearchive processes become simply distributed transactions without introducing special mechanisms into the life of a system administrator.

This mechanism relies on maintenance of an archive record in the active database that records information about each archive unit placed in an archive database. The existence of an archive record in the active database allows the active database to return a *forwarding reference* instead of a load error when a requested revision or time of an object has been archived. The reference contains the address of the archive database, allowing the object manager to proceed to indirectly load the archived object as an alias for the requested one. Obviously, alias objects must be marked to prohibit update. The object manager can take the appropriate action to access archived objects (or revisions of objects) depending on the wishes of the user and system policy. In our system, the object manager recognizes several access modes to indicate how to treat archived data for each application operation.

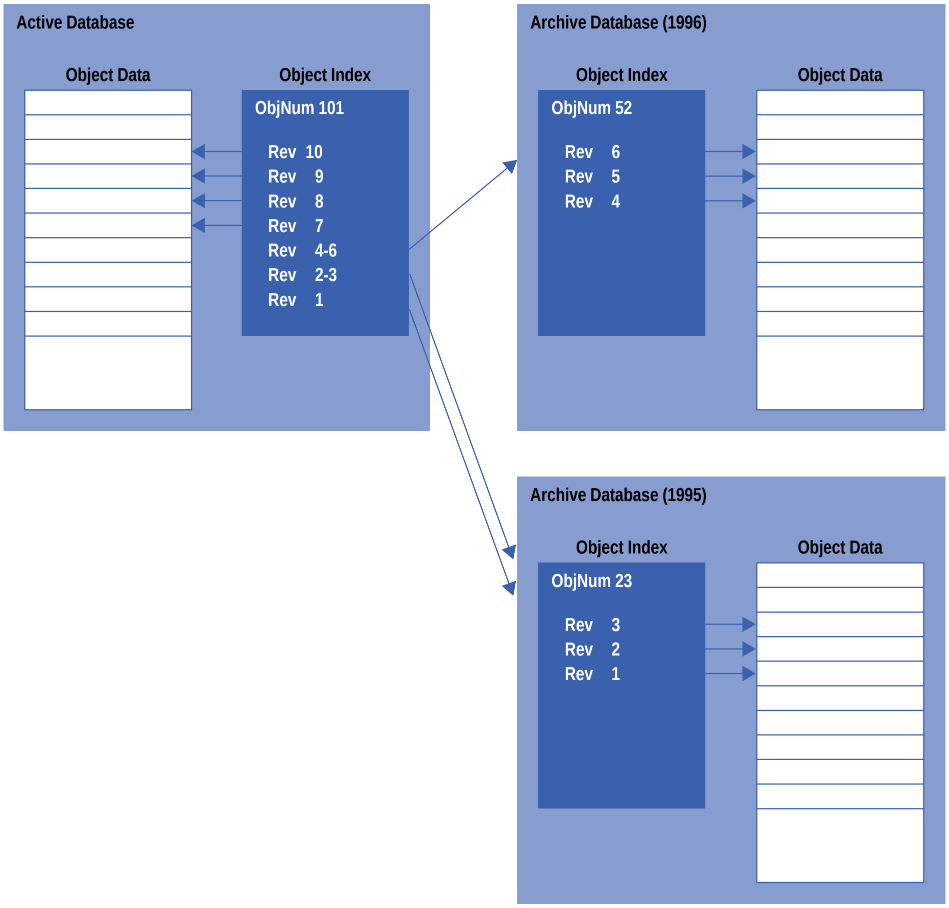


Fig. 5. *Time-slice archive example.*

Conclusion

The trend towards requiring audit trails of more and more processes is driving new database capabilities. Old models of audit logging and periodic archives do not provide routine access to audit data and are not scalable to large systems. We should not view auditing as a specialized, application-specific capability to be overlaid on a general-purpose database.

Object database systems are well-suited to implement this new technology because much of the technology can be incorporated efficiently within the DBMS, freeing the designer and programmer from many of the new complexities introduced in the discussion above. Ad hoc implementations using stored procedures, triggers, or other enhancements of relational databases will have difficulty matching the efficiency of systems in which auditing is an implicit capability.

Auditing objects in complex schemas and archiving the data in a distributed environment are complex processes that would appear to be difficult to implement in ordinary applications. On the contrary, we have found that these capabilities can be used reliably by application developers because most of the complexity can be concentrated in the object manager of an ODBMS and core class code. Similarly, access to archived data can be nearly transparent to most application code with judicious use of access modes and exception traps if the object manager implements automatic indirect access to archive databases.

The ambitious goals of rapid access to active data, convenient access to old data, practical database size, and reasonable application complexity can be achieved in an internally audited system by careful design of a distributed database system.

References

1. N. Kline, "An Update of the Temporal Database Bibliography," *SIGMOD Record*, Vol. 22, no. 4, 1993, pp. 66-80.
2. A.U. Tansel, et al, *Temporal Databases*, Benjamin/Cummings, 1993.
3. *Illustra TimeSeries DataBlade*, Illustra Information Technologies Data Sheet, January 1996.
4. J. Rumbaugh, et al, *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

Online

More information about the products associated with this article can be found at:

http://www.dmo.hp.com/apg/products/chemlms.html

Testing Policing in ATM Networks

Policing is one of the key mechanisms used in ATM (Asynchronous Transfer Mode) networks to avoid network congestion. The HP E4223A policing and traffic characterization test application has been developed to test policing implementations in ATM switches before the switches are deployed for commercial service.

by Mohammad Makarechian and Nicholas J. Malcolm

The Asynchronous Transfer Mode (ATM) is a network technology that can satisfy the quality-of-service requirements of many different types of traffic. The ability of ATM to handle many different types of traffic and its ability to operate at high bandwidths position it to be one of the core technologies behind future broadband wide area networks and the Internet. To provide quality-of-service guarantees, ATM relies crucially upon avoiding network congestion. Congestion can result in unacceptably large cell loss or delays. Cells that are lost may have to be retransmitted, which can result in increased congestion. Cells with excessive delays can cause higher-layer protocol timers (e.g., TCP/IP timers) to expire, which will result in even more cells being retransmitted. For video traffic, excessive delays or cell delay variation can result in underflow in the video decoder buffers. This can cause jagged movements or screen freezes when playing back the video.

Policing is one of the key mechanisms used by ATM to avoid network congestion. Policing is responsible for monitoring the amount of traffic sent by a connection. If a connection is sending more than the agreed-upon amount of traffic, then policing can discard traffic from the offending connection. By preventing too much traffic from entering the network, policing helps to avoid network congestion. This ensures that existing connections in the network will continue to receive their required quality of service.

Policing occurs at the *user-network interface* (UNI), where user traffic first enters a public network, and at the *broadband ISDN intercarrier interface* (B-ICI), where traffic crosses from one public network to another. Policing is known as *usage parameter control* (UPC) at the UNI and *network parameter control* (NPC) at the B-ICI.

Given the importance of policing to ATM, it is essential that policing be well-tested. Policing must be tested both by network equipment manufacturers when developing switches, and by network providers when commissioning switches. The HP E4223A policing and traffic characterization test application has been developed to test policing implementations in ATM switches. This product allows users to generate policing test traffic and to measure how the traffic is affected by policing. In this way the HP E4223A can thoroughly test policing in ATM switches before the switches are deployed for commercial service. The HP E4223A can also analyze the traffic originating from a traffic source to determine whether the source is sending too much traffic into the network.

How Policing Works

For ATM to meet its quality-of-service commitments, it is essential to reduce or eliminate network congestion. Congestion can result in unacceptably poor network performance. ATM attempts to avoid congestion by managing network resources (e.g., transmission links, buffer space inside switches) in such a way that congestion will not occur. A connection will only be established if there are enough network resources to provide an acceptable quality of service to the new connection without disrupting the service provided to existing connections. Once a connection has been established, usage parameter control (UPC) is responsible for policing the connection traffic when it enters the network and ensuring that the traffic does not exceed the agreed-upon traffic rate.

Depending on the requirements of the traffic source, ATM provides a variety of service categories, as shown in Fig. 1.[1] For example, an application such as digital voice may be suited for *constant bit rate* (CBR) service, while compressed video may be suited for *real-time variable bit rate* (rt-VBR) service. The service category used by a connection is chosen at connection setup time. For each service category, several traffic parameters are given to the network to describe the type of traffic that will be sent by the connection. For *switched virtual connections* (SVCs), the traffic parameters are given to the network during the call setup or renegotiation phase of signaling. For *permanent virtual connections* (PVCs) the traffic parameters can be specified manually at subscription time. The traffic parameters are used by the network to police the traffic on the connection and to determine how many network resources must be reserved to support the connection.

Cells in an ATM network can be given a high priority or a low priority. High-priority cells have the *cell loss priority* (CLP) bit in their headers set to 0, while low-priority cells have a CLP of 1. Low-priority cells are more likely to be discarded if the network becomes congested. At the minimum, the traffic parameters declared to the network at connection setup time include the *peak cell rate* (PCR) and the *cell delay variation tolerance* (CDVT) for CLP = 0+1 cells, that is, for all cells in the connection, regardless of priority.

Service Category*	Characteristics	Example Applications	Traffic Parameters
Constant Bit Rate (CBR)	• Tightly bounded cell delay variation • Static amount of bandwidth available throughout a connection's lifetime • Low cell loss ratio	Video/audio on demand, video conferencing, digital telephony	• PCR, CDVT on CLP= 0+ 1 cells
Real-Time Variable Bit Rate (rt-VBR)	• Supports bursty traffic • Tightly bounded cell delay variation • Low cell loss ratio	Compressed video, distributed classroom	• PCR, CDVT and SCR, MBS on CLP= 0+ 1 cells • PCR, CDVT on CLP= 0+ 1 cells, SCR, MBS on CLP= 0 cells • PCR, CDVT on CLP= 0+ 1 cells, SCR, MBS on CLP= 0 cells, tagging applicable
Non-Real-Time Variable Bit Rate (nrt-VBR)	• Supports bursty traffic • No cell delay variation bounds • Low cell loss ratio	Airline reservations, banking transactions	
Unspecified Bit Rate (UBR)	• No cell delay variation bounds • No cell loss ratio bounds • "Best effort" service	File transfer, e-mail	• PCR, CDVT on CLP= 0+ 1 cells • PCR, CDVT on CLP= 0+ 1 cells, tagging applicable

*The ATM Forum also defines a service category called available bit rate (ABR). Policing of ABR connections is not discussed in this article.

Fig. 1. *ATM service categories.*

The PCR is the maximum rate at which the source may generate traffic. The CDVT indicates how many back-to-back cells there may be at the user-network interface. Together, the PCR and the CDVT give the network an idea of when to expect the next arrival of a cell given that one has just arrived. In addition, for variable bit rate service categories, a traffic source can also specify a *sustainable cell rate* (SCR) and a *maximum burst size* (MBS). The SCR gives an upper bound on the conforming cell rate of a VBR connection. The MBS gives the maximum burst size for a VBR connection, assuming that the cells in the burst arrive at the PCR.Specifying the SCR and MBS allows the network to allocate resources such as buffer space more efficiently because the network has more knowledge about the type of traffic that will be generated.

The UPC function is responsible for ensuring that traffic on a connection does not exceed the agreed-upon rate. The policing function in a switch first validates the VPI/VCI (*virtual path identifier/virtual channel identifier*) of arriving cells, and then determines whether or not the cells are conforming to the agreed-upon PCR or SCR. Whether or not a cell is conforming is determined by an algorithm called the *generic cell rate algorithm* (GCRA),* popularly known as the "leaky bucket" algorithm (Fig. 2). The GCRA has two parameters, denoted T and τ. The first parameter, T, is the emission interval and can be regarded as the expected interarrival time of conforming cells. The second parameter, τ, is the cell delay variation tolerance (CDVT) and determines how many back-to-back cells are allowed. The GCRA maintains a variable called the *theoretical arrival time* (TAT), which gives the expected arrival time of the next cell. Cells arriving more than τ units of time before the TAT are considered to be nonconforming. Nonconforming cells can be tagged (given a lower priority) or

* The GCRA is a reference algorithm used to define conformance. An actual UPC implementation may use the GCRA or another algorithm provided that the quality-of-service objectives for connections are met.

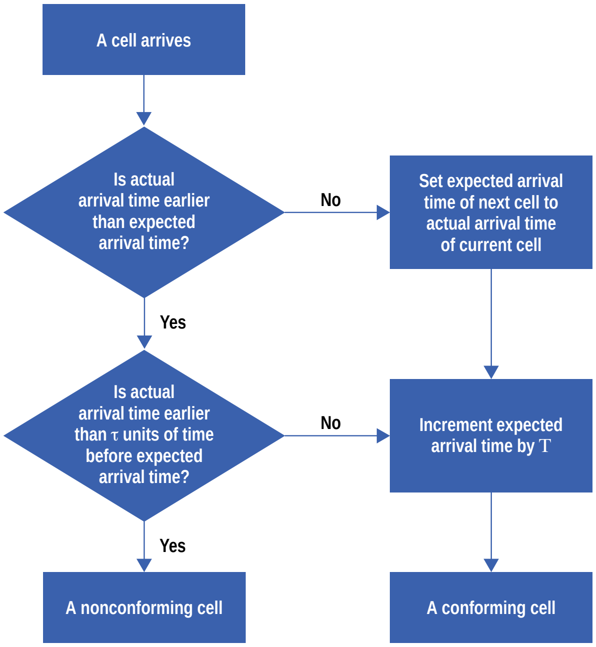

A cell arrives
Is actual arrival time earlier than expected arrival time?
No
Set expected arrival time of next cell to actual arrival time of current cell
Yes
Is actual arrival time earlier than τ units of time before expected arrival time?
No
Increment expected arrival time by T
Yes
A nonconforming cell
A conforming cell

Fig. 2. *Generic cell rate (leaky bucket) algorithm.*

discarded by the switch. Cells arriving τ units of time before the TAT or later are considered to be conforming. For each conforming cell, the TAT is updated to give the expected arrival time of the next cell in the connection.

For an example of how the GCRA can be used to police a CBR connection, suppose that a connection has a peak cell rate of PCR = 8000 cells/s and a CDVT of 0 μs. The GCRA emission interval is calculated as T = 1/PCR = 125 μs. The GCRA CDVT is τ = 0 μs. Table I shows the resultant conforming and nonconforming cells assuming that a cell on the connection arrives every 120 μs.

Table I
Conforming and Nonconforming Cells
for PCR = 8000 cells/s, $\tau = 0$ μs

Cell #	$t_{arrival}$ (μs)	TAT (μs)	Conforming?
1	0	0	yes
2	120	125	no
3	240	125	yes
4	360	365	no
5	480	365	yes
6	600	605	no
7	720	605	yes
8	840	845	no
9	960	845	yes
10	1080	1085	no

To see how the CDVT can be increased to allow more back-to-back cells, suppose that the CDVT used by the GCRA is increased to τ = CDVT = 11 μs. The resultant pattern of conforming and nonconforming cells is shown in Table II.

Table II
Conforming and Nonconforming Cells
for PCR = 8000 cells/s, $\tau = 11$ μs

Cell #	$t_{arrival}$ (μs)	TAT (μs)	Conforming?
1	0	0	yes
2	120	125	yes
3	240	250	yes
4	360	375	no
5	480	375	yes
6	600	605	yes
7	720	730	yes
8	840	855	no
9	960	855	yes
10	1080	1085	yes

As mentioned earlier, some service categories also have sustainable cell rate (SCR) and maximum burst size (MBS) parameters. In this case, one GCRA is used to police the peak cell rate and another GCRA is used to police the sustainable cell rate. The GCRA tolerance used with the SCR GCRA (τ_{scr}) is derived from the PCR, SCR, MBS, and CDVT parameters,[1] and permits a burst of MBS cells at the peak cell rate. The PCR and SCR GCRAs form a dual leaky bucket algorithm and operate in lockstep fashion. A cell is only considered to be conforming if it conforms to both GCRAs. If tagging is allowed, then high-priority CLP = 0 cells can be tagged (given a lower priority) if they do not conform to the SCR GCRA.

In addition to its role in the UPC function, the GCRA can also be used inside a traffic source to ensure that the outgoing cell flow conforms to a particular cell rate. This is referred to as traffic shaping. When nonconformance is detected during shaping, the offending cells are delayed until their transmission will be conforming. In this way, traffic can be guaranteed conforming before it enters the network.

The HP BSTS Policing Application

The HP E4223A policing and traffic characterization test application is designed to test UPC implementations in network equipment and to analyze the characteristics of traffic on a connection. The HP E4223A is part of the HP Broadband Series Test System (BSTS).[2] The HP Broadband Series Test System contains a number of VXIbus modules that allow testing of broadband networks over a variety of physical interfaces. For brevity, the HP E4223A will be denoted the HP BSTS policing application during the remainder of this article.

The HP BSTS policing application works with the HP E4209 cell protocol processor, a VXIbus module forming part of the HP BSTS. The cell protocol processor in conjunction with a line interface module can transmit and receive ATM cells for testing purposes. Received ATM cells can be stored in a capture RAM for later analysis. The HP BSTS policing application consists of embedded software running on the cell protocol processor module and a software component running on HP 9000 Series 400 or 700 workstations.

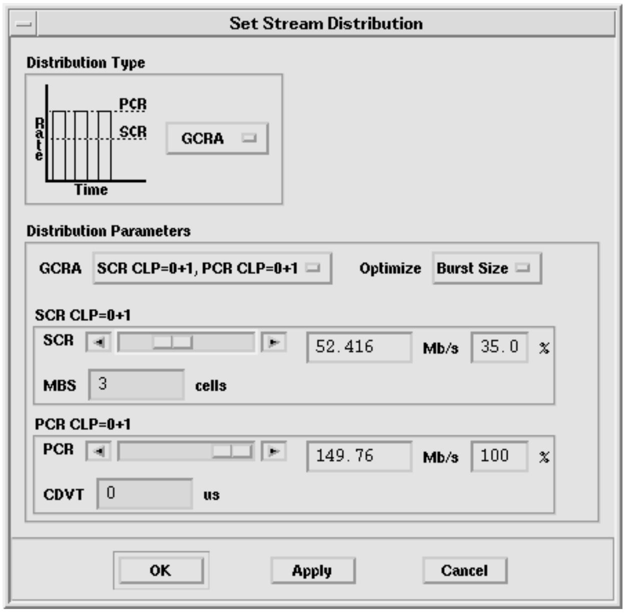
The HP BSTS policing application provides the following functions:

- Generates traffic conforming to a single or dual leaky bucket algorithm (GCRA).
- Generates UPC test cells, which are test cells designed for testing policing.
- Makes a number of policing-related measurements on captured ATM cells, such as the number of nonconforming cells and the number of cells that were lost or tagged.
- Makes general performance measurements on captured ATM cells, such as cell delay, interarrival time, and one-point cell delay variation.

Traffic Generation

When generating traffic to test policing, it is important to test the limits of the GCRA being used for policing. This means that it is important to generate traffic that has the maximum cell rate and burst size but is still conforming. Because policing is configured in a switch using the parameters of a GCRA, it is convenient to use the parameters of a GCRA when specifying traffic to test policing. Consequently, the HP BSTS policing application provides a GCRA distribution, which allows traffic to be generated using the parameters of a GCRA (Fig. 3). The GCRA combinations supported are:

- PCR CLP = 0+1 (single leaky bucket)
- SCR CLP = 0+1 and PCR CLP = 0+1 (dual leaky bucket).



Set Stream Distribution

Distribution Type: GCRA

Distribution Parameters

GCRA: SCR CLP=0+1, PCR CLP=0+1 — Optimize: Burst Size

SCR CLP=0+1

SCR: 52.416 Mb/s, 35.0 %

MBS: 3 cells

PCR CLP=0+1

PCR: 149.76 Mb/s, 100 %

CDVT: 0 us

Fig. 3. *Specifying the generic cell rate algorithm distribution with the HP E4223A policing and traffic characterization test application.*

The GCRA distribution can be optimized to generate traffic based on either the cell rate or the burst size. This allows independent testing of how policing implementations handle cell rates and burst sizes.

Traffic that optimizes the burst size consists of repeated bursts of the maximum possible conforming burst size. The burst is at the line rate for the single leaky bucket. For the dual leaky bucket, the burst consists of MBS cells at the peak cell rate. The spacing between the bursts is the minimum necessary to maintain conforming traffic. For example, suppose a dual leaky bucket is chosen with SCR = 35%, MBS = 3 cells, PCR = 100%, and CDVT = 0 ms. The generated traffic will consist of repeated bursts of three cells at 100% of the line rate, with a gap of six cells between any two bursts. Because cell transmission is quantized, the generated load is 33.3%, which is less than the SCR. Traffic that optimizes the burst size has very precisely controlled burst sizes, but the rate of the generated traffic may be less than requested.

Traffic that optimizes the cell rate consists of traffic generated at the maximum conforming rate. The traffic will consist of an initial burst of the maximum conforming burst size, followed by cells at the PCR (for a single leaky bucket) or at the SCR (for a dual leaky bucket). For example, with a dual leaky bucket with SCR = 35%, MBS = 3 cells, PCR = 100%, and CDVT = 0 ms, the generated traffic consists of an initial burst of three cells, followed by conforming traffic at the SCR. When optimizing the cell rate, the generated traffic rate can be much closer to the requested rate than traffic that optimizes the burst size.

Policing Measurements

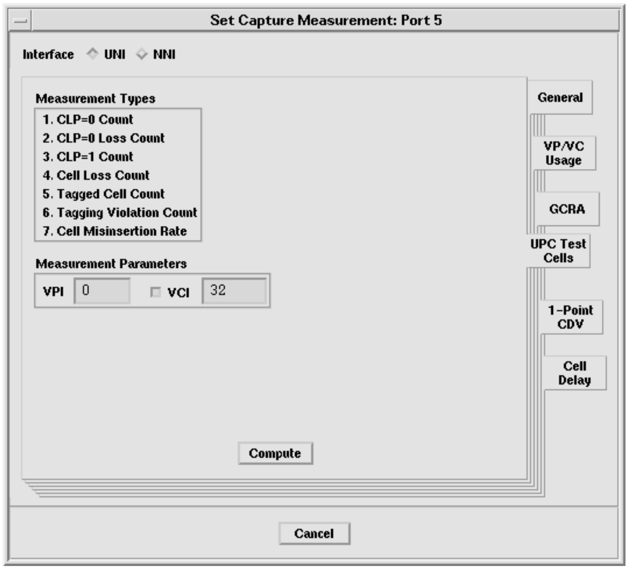
When generating traffic to test policing, the number of tagged or discarded cells must be measured. It is difficult for test equipment to make these measurements with regular user traffic. This is because the test equipment usually does not know how many user cells were transmitted or the original priority of the user cells. For this reason, test cells are often used to measure policing performance. The HP BSTS policing application can transmit sequences of UPC test cells, each cell having the format shown in Fig. 4. The UPC test cells are specifically designed for testing policing with the HP BSTS policing application. The payload of each UPC test cell within a sequence contains the following information:

- SN_{0+1}. The number of previous test cells in the sequence.
- RES. Reserved bytes, set to zero.
- SL_{0+1}. The total number of test cells in the sequence. The HP BSTS policing application can repeatedly transmit sequences of 512 or 1024 test cells.
- SL_0. The total number of high-priority test cells in the sequence.
- OCLP. The priority of the test cell when it is first transmitted. The low-order bit is set to 0 for high-priority cells and 1 for low-priority cells. The remaining bits in this field are set to zero.
- SN_0. The number of previous high-priority test cells in the sequence.
- VN. The version number of the test cell format. The current version number is 0.
- CRC-16. A cyclic redundancy check error code to provide protection and validation of the encoded payload information. The CRC-16 code is computed using the polynomial $x^{16} + x^{12} + x^5 + 1$.

SN_{0+1}	RES	SL_{0+1}	SL_0	OCLP	SN_0	VN	CRC-16
3 Bytes	32 Bytes	3 Bytes	3 Bytes	1 Byte	3 Bytes	1 Byte	2 Bytes

Fig. 4. *Payload of HP BSTS policing application UPC (usage parameter control) test cell.*

The information contained in the payload of UPC test cells allows a number of measurements to be made on captured ATM cells (Fig. 5). These measurements include the number of lost or tagged cells, which are measurements directly relevant to testing policing.



Set Capture Measurement: Port 5

Interface ◇ UNI ◇ NNI

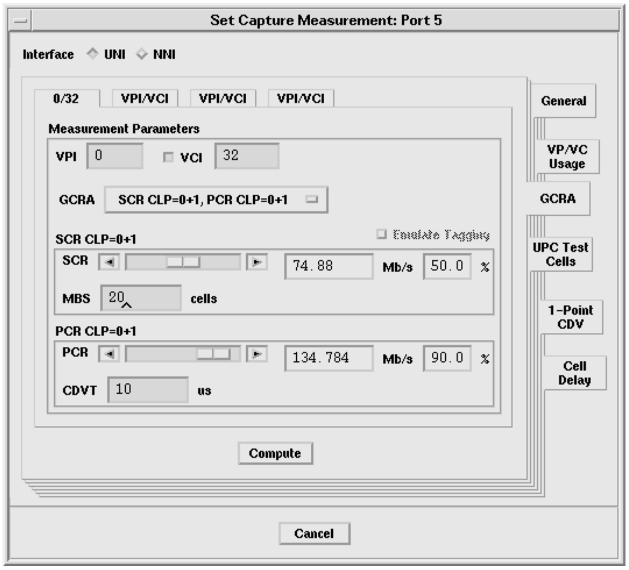
Measurement Types
1. CLP=0 Count
2. CLP=0 Loss Count
3. CLP=1 Count
4. Cell Loss Count
5. Tagged Cell Count
6. Tagging Violation Count
7. Cell Misinsertion Rate

Measurement Parameters

VPI 0 □ VCI 32

Fig. 5. *UPC test cell measurements.*

In addition to making measurements with UPC test cells, the HP BSTS policing application can measure the conformance of traffic on a connection (Fig. 6). Conformance is measured by saving ATM cells in the capture RAM and then measuring the number of captured nonconforming cells. These measurements can be used to test the number of nonconforming cells detected by a switch or to test whether the traffic on a connection is conforming.



Interface ◆ UNI ◇ NNI

Measurement Parameters

VPI 0 □ VCI 32

GCRA SCR CLP=0+1, PCR CLP=0+1

□ Emulate Tagging

SCR CLP=0+1

SCR 74.88 Mb/s 50.0 %

MBS 20 cells

PCR CLP=0+1

PCR 134.784 Mb/s 90.0 %

CDVT 10 us

Fig. 6. *GCRA (generic cell rate algorithm) conformance measurements.*

Testing Policing in a Switch

Policing in ATM switches must work correctly if ATM is to realize its potential for providing guaranteed quality of service for many different types of traffic. This requires that policing be thoroughly tested, both during switch development and during switch deployment. There are basically two aspects of policing to be tested: conforming cells should not be tagged or discarded, and nonconforming cells should be tagged or discarded to protect the quality of service provided to other connections. To test the above aspects of policing, the number of lost cells, the number of tagged cells, and the number of lost high-priority cells must all be measured (Table III).

Table III
Parameters to Measure when Testing Policing

Parameter	Description
Lost cells	Number of cells discarded or lost in the switch. This parameter is used to check that policing is not discarding too many cells.
Tagged cells	Number of cells tagged (changed from high to low priority) by the switch. This parameter is used to check that policing only changes the priority of a cell when necessary.
Lost high-priority cells	Number of high-priority cells discarded or lost in the switch. This parameter is used to check that policing is not discarding too many high-priority cells.

When testing policing in a switch with the HP BSTS policing application, the approach is to transmit a well-understood stream of test cells into the switch, capture the cells after they have traversed the switch, and then calculate how many cells were tagged or discarded. This approach is well-suited for stimulus-response type testing to test the capabilities of the switch systematically.

To simplify testing, the overall philosophy when testing policing in a switch is to test one GCRA parameter at a time. This means keeping the cell rate constant while varying the burst size, or keeping the burst size constant while varying the cell rate. Fig. 7 shows how to test the cell rate of a single leaky bucket. The switch is first configured with the leaky bucket parameters to be tested—in this case, the PCR and CDVT for a single leaky bucket. The PCR and CDVT used to generate test traffic are then entered, with the PCR used to generate the traffic being lower than the PCR in the switch. The testing then iterates between measuring the number of tagged or lost cells and increasing the PCR. If the number of tagged or lost cells differs from what is expected, a potential defect is logged.

Example: Testing a single leaky bucket in a switch. The approach in Fig. 7 was followed to test a single leaky bucket (GCRA) in a switch with PCR = 8 Mbits/s and CDVT = 60 ms on a 155-Mbit/s SONET port. The HP BSTS policing application was used to generate traffic consisting of a repeating sequence of 1024 UPC test cells conforming to a GCRA with PCR = 4% (5.9 Mbits/s) and CDVT = 60 ms. The traffic was sent through the switch and back into the HP BSTS, where it was placed in the cell protocol processor capture RAM. The ratio of lost cells was calculated based on the captured UPC test cells. The PCR was then incremented by 0.5% for the next iteration of the test. The expected cell loss ratio was 0 if the generated cell rate was

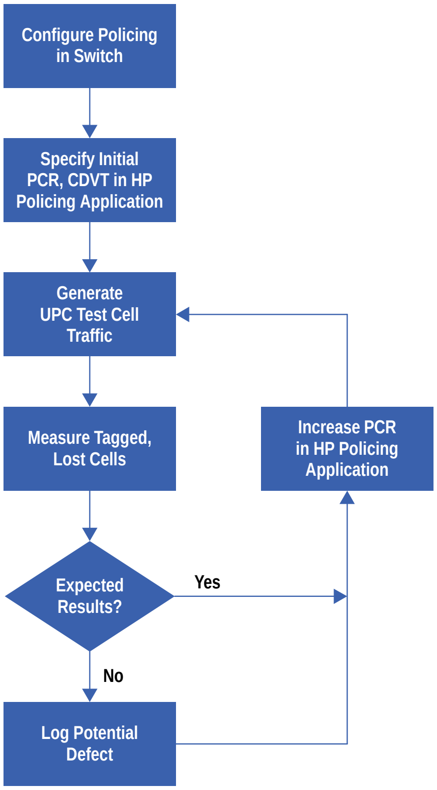

Configure Policing in Switch
Specify Initial PCR, CDVT in HP Policing Application
Generate UPC Test Cell Traffic
Measure Tagged, Lost Cells
Expected Results?
Yes
No
Log Potential Defect
Increase PCR in HP Policing Application

Fig. 7. *Testing the PCR (peak cell rate) of a single leaky bucket.*

less than the policing cell rate, otherwise the expected cell loss ratio was the proportion of generated cell rate greater than the policing cell rate, that is $(PCR_{traffic} - PCR_{police})/PCR_{traffic}$. Table IV shows the test results.

Table IV
Test Results for a Single GCRA in a Switch

PCR for Generating Traffic	Cell Loss Ratio	Expected Result?
4.0%	0.00	yes
4.5%	0.00	yes
5.0%	0.00	yes
5.5%	0.02	yes
6.0%	0.10	yes
6.5%	0.17	yes
7.0%	0.23	yes

Testing Traffic Conformance

Although policing in network switches will protect the network from traffic sources that send too much traffic, it is also important for traffic sources themselves to generate conforming traffic if possible. If a source generates nonconforming traffic, then the nonconforming cells will be discarded by the network and may have to be retransmitted by the source. This can significantly degrade the network performance experienced by the traffic source. The HP BSTS policing application can be used to check whether a source is generating conforming traffic.

Example: Testing the conformance of MPEG-2 video traffic. This example demonstrates how to use the HP BSTS policing application to measure the conformance of a traffic source. As shown in Fig. 8, a laser disk player was connected to a commercial MPEG-2 encoder with a 45-Mbit/s DS3 ATM output. The encoder was set up to generate MPEG-2 video over ATM at 4 Mbits/s. The user's guide for the encoder states that a CDVT of 100 ms should be sufficient to compensate for the effects of adapting MPEG-2 packets to ATM cells.

The ATM output of the MPEG-2 encoder was first sent directly to the HP BSTS, where the MPEG-2 traffic was placed in the cell protocol processor capture RAM. To verify that the MPEG-2 traffic was being captured correctly, the HP E4226B MPEG-2 protocol viewer test software was used to play back the captured video segment. The HP BSTS policing application was then used to measure the number of nonconforming cells with a PCR CLP = 0+1 GCRA. The GCRA parameters were

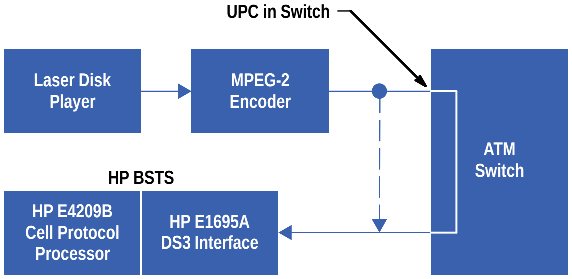


Fig. 8. *Testing the conformance of video traffic.*

chosen conservatively to be PCR = 4.07 Mbits/s and CDVT = 200 ms. The HP BSTS policing application measurements showed that the MPEG-2 encoder was not well-behaved, with approximately 25% of the cells being nonconforming.

To see the effect of the nonconforming cells on the video traffic, the output of the MPEG-2 encoder was then directed to an ATM switch before being routed to the HP BSTS. The switch was configured to police the MPEG-2 traffic with PCR = 4.07 Mbits/s and CDVT = 200 ms. Like the HP BSTS, the switch detected approximately 25% of the cells as being nonconforming. These nonconforming cells were discarded by the switch. The remaining cells passed through the switch and were captured in the cell protocol processor capture RAM. However, because of the large number of ATM cells that were discarded by the switch, it was not possible to play back even one video frame. This example clearly demonstrates the importance of generating conforming ATM traffic and shows how the HP BSTS policing application can be used to test the conformance of a traffic source.

Conclusion

Policing network traffic at the UNI or B-ICI is crucial to maintaining quality-of-service guarantees in ATM-based networks. The ability to support the quality-of-service requirements of many different types of traffic is one of the distinguishing features of ATM. This feature means that ATM is well-suited to providing the backbone network for future broadband wide area networks and the Internet. The HP BSTS policing application enables switch vendors and service providers to test policing and helps ensure the successful deployment of ATM.

Acknowledgments

The authors would like to acknowledge the contributions of many individuals who participated in the development and deployment of the HP BSTS policing application, including Brian Smith (project manager), Lawrence Croft (product design), Judith Watson (learning products), Mark Leonard (usability engineer), Drew Paterson, Jack Lam, and Scott Reynolds (QA testing), and Reto Brader (product marketing).

References

1. *ATM Forum Traffic Management Specification Version 4.0,* The ATM Forum Technical Committee, March 1996.
2. The URL for the HP BSTS is http://www.hp.com/go/bsts.

List of Acronyms

ATM	Asynchronous Transfer Mode
B-ICI	Broadband ISDN intercarrier interface
BSTS	Broadband Series Test System
CBR	Constant bit rate
CDVT	Cell delay variation tolerance
CLP	Cell loss priority
GCRA	Generic cell rate algorithm
MBS	Maximum burst size
NPC	Network parameter control
PCR	Peak cell rate
PVC	Permanent virtual connection
rt-VBR	Real-time variable bit rate
SCR	Sustainable cell rate
SVC	Switched virtual connection
TAT	Theoretical arrival time
TCP/IP	Transmission Control Protocol/Internet Protocol
UNI	User-network interface
UPC	Usage parameter control
VBR	Variable bit rate
VPI/VCI	Virtual path identifier/virtual channel identifier

MOSFET Scaling into the Future

2D process and device simulators have been used to predict the performance of scaled MOSFETs spanning the 0.35-μm to 0.07-μm generations. Requirements for junction depth and channel doping are discussed. Constant-field scaling is assumed. MOSFET drive current remains nearly constant from one generation to the next and most of the performance improvement comes from the decreasing supply voltage. Gate delay decreases by 30% per generation, nearly the same trend as previous generations. However, this performance gain comes at the price of much higher off-state leakage because of the reduction of the threshold voltage. Various solutions to this high leakage are discussed.

by Paul Vande Voorde

Hewlett Packard adopted CMOS technology in the mid- 1970s. At that time the gate length L_g was 4 μm and the gate oxide thickness T_{ox} was 50 nm. Since then, each new generation of technology has shrunk L_g by about 30% and T_{ox} by about 25%. The decrease in L_g has been tied to the evolution of lithography equipment. Following these scaling trends, intrinsic gate delay has decreased about 30% per generation. New generations of technology are released about every three years. The important principle in MOSFET scaling is that L_g and T_{ox} must decrease together. Scaling one without the other does not yield adequate performance improvement.

The performance metric for gate delay is CV/I, where C is the load capacitance, V is the supply voltage (V_{dd}), and I is the drive current of the MOSFETs (average of NMOS and PMOS). C is composed of both gate and junction capacitance. MOSFET scaling, which decreases L_g, T_{ox}, and junction area while increasing substrate doping, tends to keep C fairly constant from generation to generation. For several generations of technology, the supply voltage was held constant at 5V (constant-voltage scaling). In that era, gate delay was reduced by ever-increasing MOSFET drive currents. Since the voltage was held constant while the dimensions decreased, the electric fields continuously increased. High fields and high currents tend to damage the gate oxide and lead to device deterioration. Thus, one of the main technology challenges has been to design MOSFETs with adequate reliability.

Constant-voltage scaling ended as L_g approached 0.5 μm and T_{ox} neared 10 nm. The demands of gate oxide reliability required that the supply voltage be reduced. This occurred as the peak oxide field reached roughly 4 MV/cm. We are now in an era where supply voltage is scaled along with T_{ox} so that the peak oxide electric field remains roughly constant (constant-field scaling). This study examines some of the implications for this of type scaling in future technology generations.

Process and Device Simulations

The 2D process simulator TSUPREM-4 from Technology Modeling Associates Inc. of Sunnyvale, California was used to simulate scaled MOSFET device structures. The inputs to TSUPREM-4 are the implant and oxidation steps that would be used in the actual process. The process architecture assumed is similar to current CMOS processes, employing shallow source/drain extensions and deeper main source/drain regions followed by silicidation.

The 2D device simulator MEDICI, also from Technology Modeling Associates Inc., was used to predict the electrical characteristics of the device structures from TSUPREM-4. Here we use field dependent mobility models that have been benchmarked to the HP CMOS10 process. Iterative simulations with TSUPREM-4 and MEDICI were performed to determine the requirements on junction depth and channel doping profile to ensure proper threshold and subthreshold behavior. Fig. 1 shows the device structures resulting from these simulations for each generation from 0.35 μm down to 0.07 μm. For L_g less than 0.15 μm, retrograde channel doping profiles are needed to control the subthreshold characteristics.

Figs. 2 through 5 summarize the results of this scaling study. Fig. 2 shows the scaling of T_{ox} with L_g. These two must scale together to get adequate performance improvement. Constant field scaling dictates that V_{dd} must decrease proportionally to T_{ox}, maintaining a peak oxide field of 4 MV/cm. For example, this results in T_{ox} = 2.5 nm and V_{dd} = 1V for the L_g = 0.1 μm generation.

Fig. 3 shows the scaling of effective channel length (L_{eff}) and the source/drain extension junction depth (X_j). For the 0.1-μm generation, L_{eff} is about 0.07 μm and X_j must be nearly 50 nm. The series resistance of the source/drain extension must

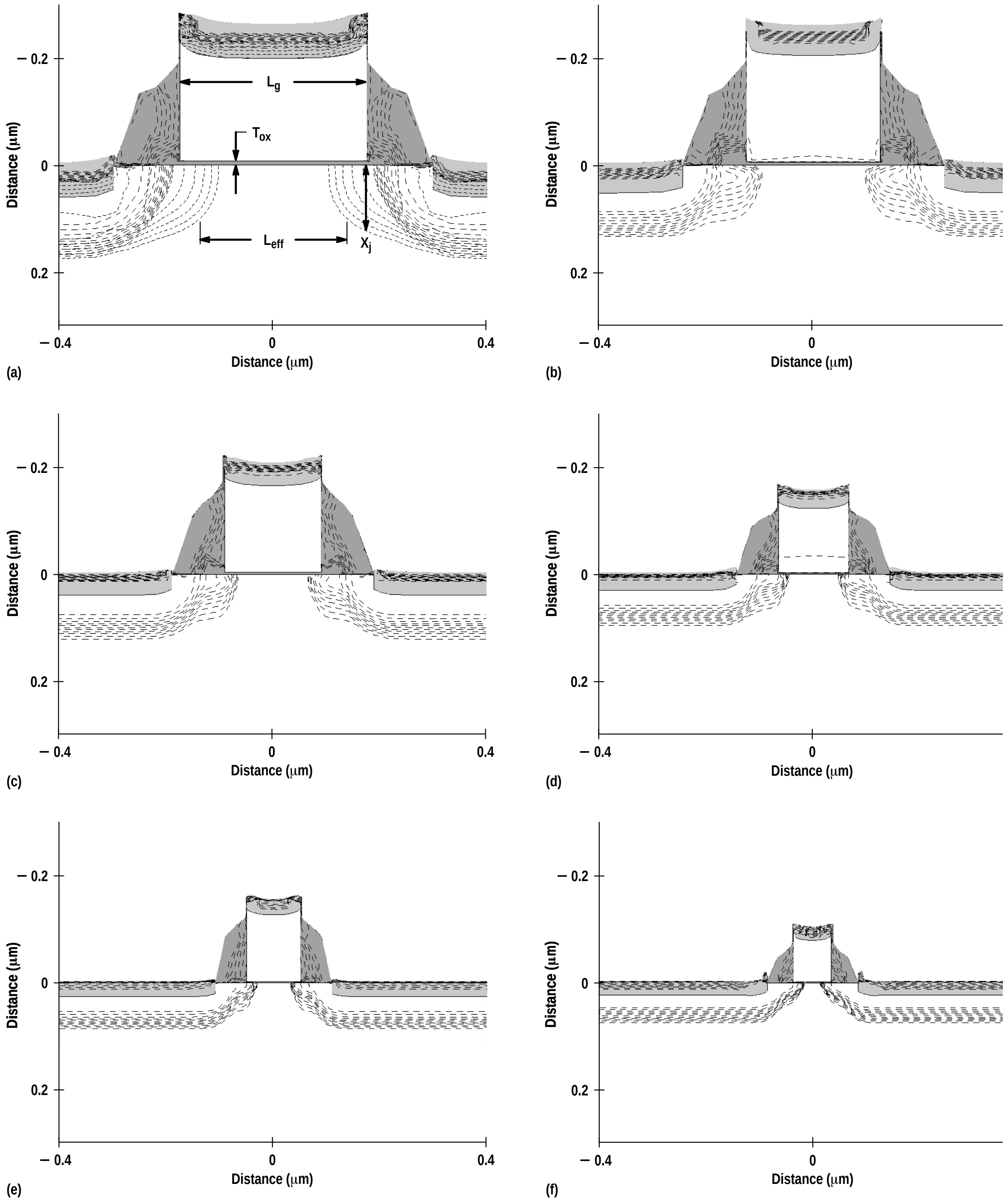


Fig. 1. *Simulated device structures. Dark shading is oxide. Lighter shading is silicide. Dashed lines are doping contours. (a) L_g = 0.35 µm, T_{ox} = 8.0 nm. (b) L_g = 0.25 µm, T_{ox} = 6.0 nm. (c) L_g = 0.18 µm, T_{ox} = 4.5 nm. (d) L_g = 0.13 µm, T_{ox} = 3.4 nm. (e) L_g = 0.10 µm, T_{ox} = 2.5 nm. (f) L_g = 0.07 µm, T_{ox} = 1.9 nm.*

decrease even as the junction depth also decreases. This requires higher doping levels in the extension region and carefully minimized spacer widths.

Fig. 4 shows the scaling of threshold voltage (V_t). Here V_t is kept at 20% of V_{dd} to maintain adequate current drive. This yields V_t = 0.2V for the 0.1-µm generation. Unfortunately, since off-state current varies exponentially with V_t, reducing V_t leads to much higher off-state leakage current (100 nA/µm for the 0.1-µm generation) than in current CMOS technologies. Here the simulations are tailored to predict the nominal leakage. Worst-case leakage would be approximately one order of magnitude higher for the 0.1-µm case.

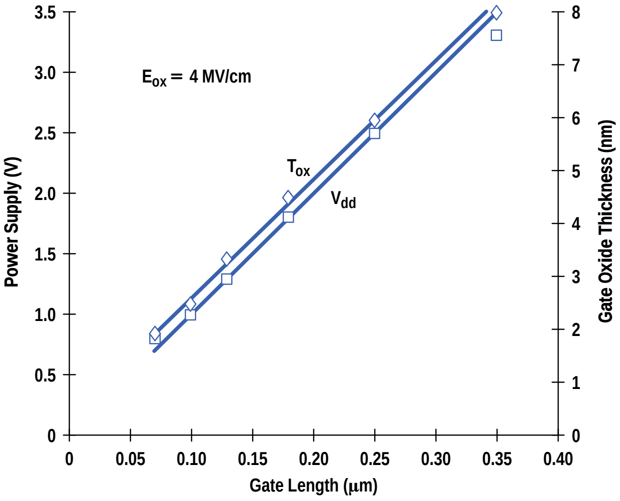


Fig. 2. *Scaling of power supply voltage V_{dd} and oxide thickness T_{ox}.*

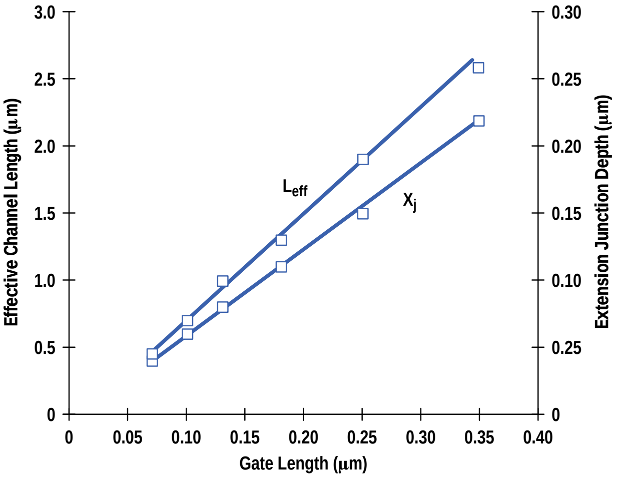


Fig. 3. *Scaling of effective channel length L_{eff} and extension junction depth X_j.*

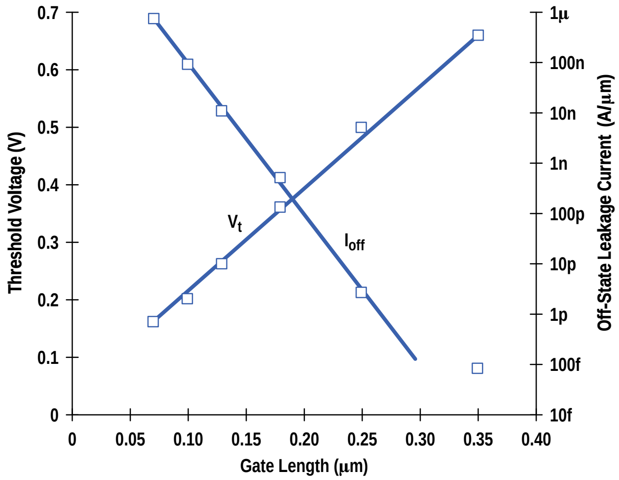


Fig. 4. *Scaling of threshold voltage V_t and off-state leakage current I_{off}.*

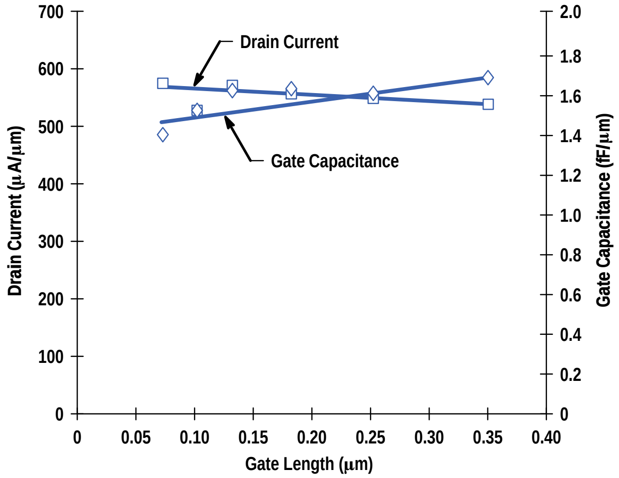


Fig. 5. *Scaling of maximum drain current and total gate capacitance.*

Fig. 5 shows the scaling of drive current and total gate capacitance. Because of the simultaneous scaling of L_g, T_{ox}, V_{dd}, and V_t, the current and capacitance do not change much from one generation to the next. Therefore, the gate delay metric CV/I decreases primarily because of the decreasing supply voltage.

Device simulators allow one to examine the internal distributions within the device. Fig. 6 shows the lateral electric field along the channel for each of the device structures in Fig. 1. Even though V_{dd} decreases as shown in Fig. 2, the peak electric field near the drain continues to increase as L_g decreases. However, the width of the high-field region decreases, giving the electrons less and less distance to reach equilibrium with the electric field. When this “nonlocal” effect is included in MEDICI, the electron temperature can be calculated as shown in Fig. 7. Here, even though the peak field increases, the electron temperature decreases as L_g decreases. Thus, we expect that the reliability issues related to high-energy charge carriers will become less important in future generations of technology.

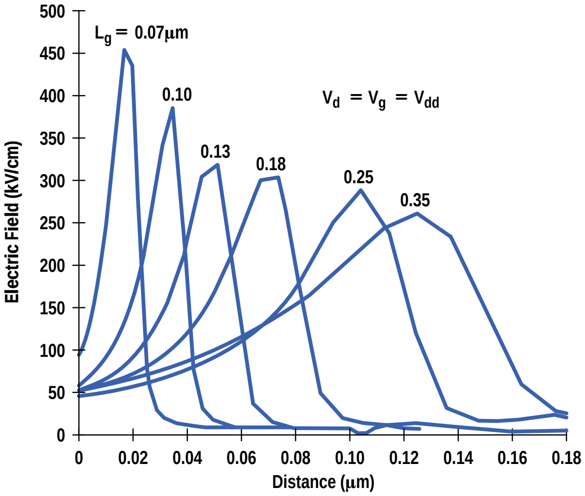


Fig. 6. *Lateral electric field along the channel beginning at the middle of the gate.*

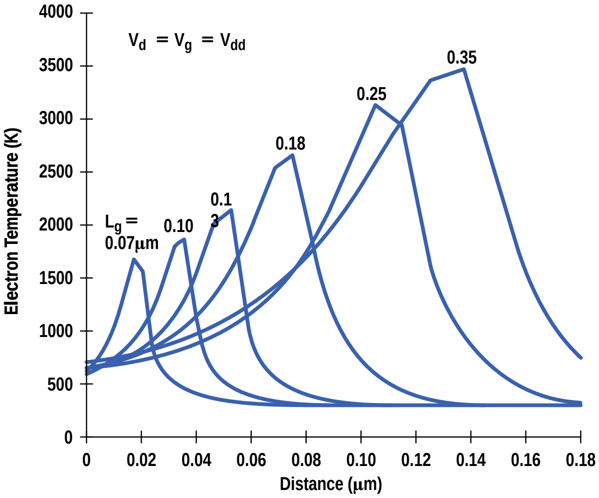


Fig. 7. *Electron temperature along the channel beginning at the middle of the gate.*

Gate Delay Simulations

MEDICI was used to generate a full set of IV curves for each of the devices in Fig. 1. IC-CAP, an HP software product for modeling semiconductor devices, was then used to extract a SPICE model for each device. Only the NMOS devices were actually simulated. The PMOS models were created from the NMOS models with appropriate modifications in mobility and series resistance to yield half the current drive of the corresponding NMOS. These device models were then used to simulate inverter chains as shown in Figs. 8 and 9. The load capacitance was varied to approximate fanouts of 3 and 7. Interconnect loading was ignored. The results are shown in Figs. 10 and 11. Fig. 10 shows that the gate delay improves about 30% per generation with the scaling described in the previous section. This is nearly the same as the historical trend of previous generations. Note that for L_g = 0.1 μm the gate delay (fanout = 1) is less than 15 ps. This is faster than the best that can currently be obtained with bipolar ECL. Fig. 11 shows the dependence of gate delay on the power supply. The stars denote the operating point from constant-field scaling. Note that these highly scaled devices offer high-speed operation even at low supply voltages. For example, the 0.1-μm generation should yield 23-ps gate delay (fanout = 1) even with V_{dd} = 0.5V. This would be excellent for low-power applications assuming that the high off-state leakage could be dealt with.

Off-State Leakage

The previous sections show that constant-field scaling of MOSFETs leads to a continuation of the historical trends of gate-level performance improvement. However, this comes at the price of exponentially increasing off-state leakage currents. For example, if an advanced circuit had 50 million micrometers of device width producing leakage current at 1 μA/μm, the quiescent supply current would be 50A. Clearly this is unacceptable. There are several proposals for dealing with this problem and I will briefly discuss some of them in this section. At this time we do not know the best way to deal with this problem.

One obvious solution to control quiescent power consumption is to put almost all the circuit in power-down mode at any instant and activate only those blocks that are being accessed. This system-level type of solution is beyond the scope of this paper and needs to be evaluated by the design community.

Another possible solution that has been proposed involves multiple threshold devices in the same technology. For example, the 0.1-μm generation could offer FETs with V_t = 0.2V and V_t = 0.4V. The low-V_t FETs could be used for speed-critical paths and the higher-V_t FETs could be used for tasks for which speed is not as important.

After modifying the doping profiles in TSUPREM-4 to get higher thresholds, the MEDICI simulations were repeated and new SPICE models extracted. Fig. 12 shows the resulting drive current and off-state current for various values of V_t in the 0.1-μm generation. Fig. 13 shows the gate delay as a function of V_t. From these graphs, FETs with V_t = 0.4V would yield gate delays about 80% longer than V_t = 0.2V but with off-state currents reduced by nearly three orders of magnitude. Again, the off-state currents shown are for nominal devices and worst-case would be higher. This approach is conceptually easy to implement in any technology. However, it increases the complexity of both the process and the circuit design.

Fully depleted (FD) silicon-on-insulator (SOI) devices have been proposed to reduce off-state current for a given V_t. These devices have a steeper subthreshold slope than conventional bulk devices, thus reducing off-state current without increasing V_t. However, single-gate FD SOI devices are difficult to scale into the deep submicrometer regime. Dual-gate FD SOI devices scale much better but are very complicated to make. These difficulties, coupled with the material quality and availability issues, make the FD SOI device an unlikely candidate for future generations of high-speed digital technology.

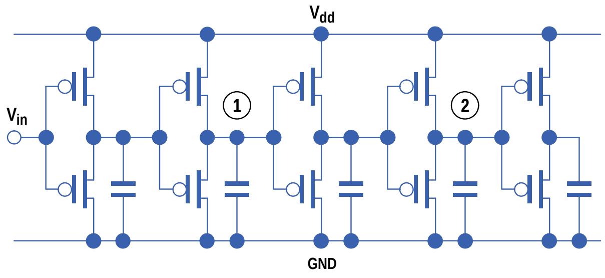


Fig. 8. *Inverter chain.*

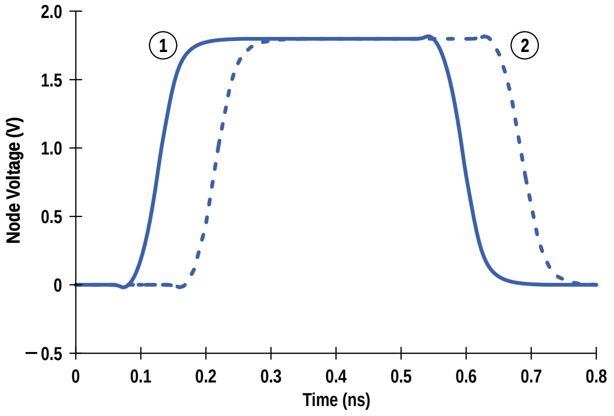


Fig. 9. *Inverter switching waveforms at nodes 1 and 2 of Fig. 8.*

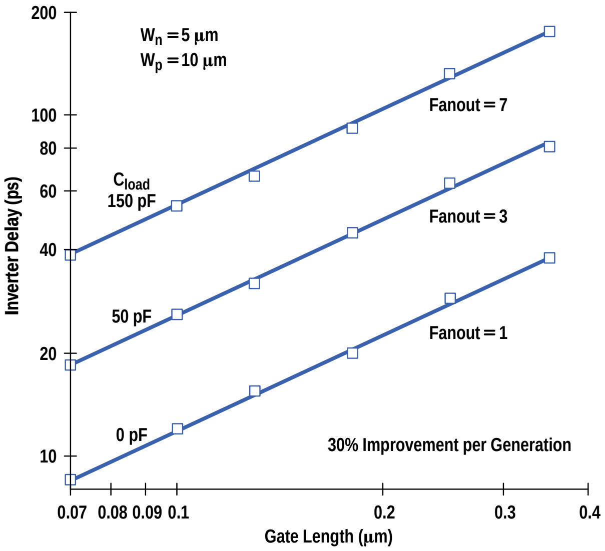


Fig. 10. *Inverter delay versus gate length.*

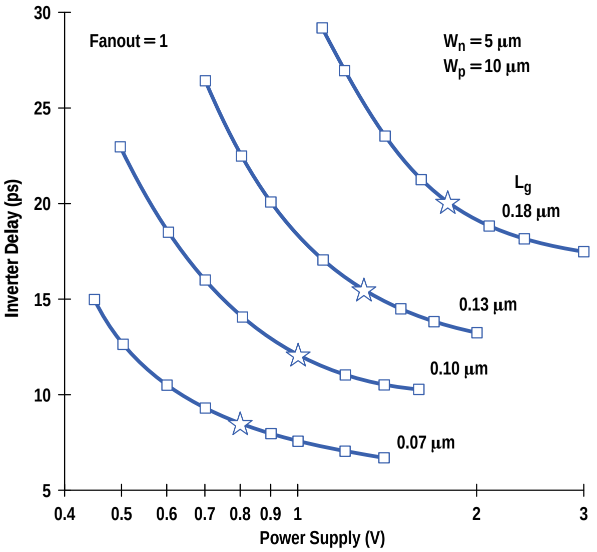


Fig. 11. *Inverter delay versus power supply voltage. The stars show the expected operating points.* W_n *and* W_p *are the widths of the n-channel and p-channel transistors.*

If no other solution for high I_{off} can be found, then V_t cannot be scaled lower than a certain point. For example, if one needed to keep I_{off} (nominal) at 1 nA/µm, then V_t (nominal) could not go below about 0.35V. We can apply this to the 0.1-µm generation (T_{ox} = 2.5 nm) and resimulate the device with V_t = 0.35V. After compact model extraction and inverter simulations, we find that V_{dd} must be increased to 1.8V to get the same performance as shown in Fig. 10 for the 0.1-µm generation. At V_{dd} = 1.8V and V_t = 0.35V, the device simulations predict a drive current of slightly over 1 mA/µm (NMOS). The peak oxide field would be over 7 MV/cm and the peak electron temperature would be about 3300K at $V_d = V_g$ = 1.8V (compare to Fig. 7). Even if we could obtain this very high drive current, it is questionable whether such a device could be created with adequate reliability. In any case, it is clear from this discussion that ceasing threshold voltage scaling would have a crucial impact on future device technologies.

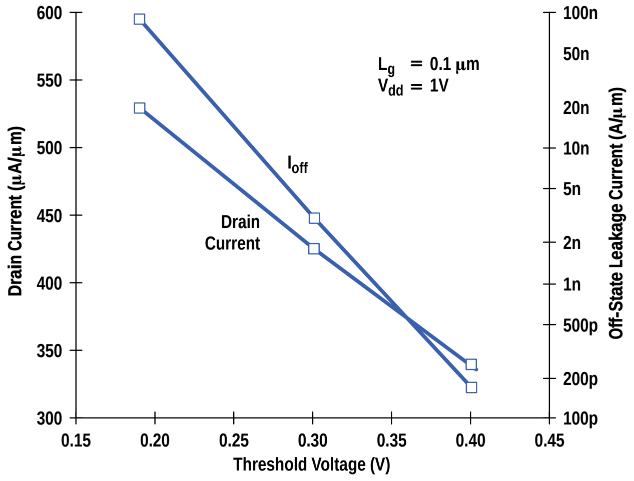


Fig. 12. *Drain current and off-state leakage current I_{off} versus threshold voltage V_t for the 0.1-μm generation.*

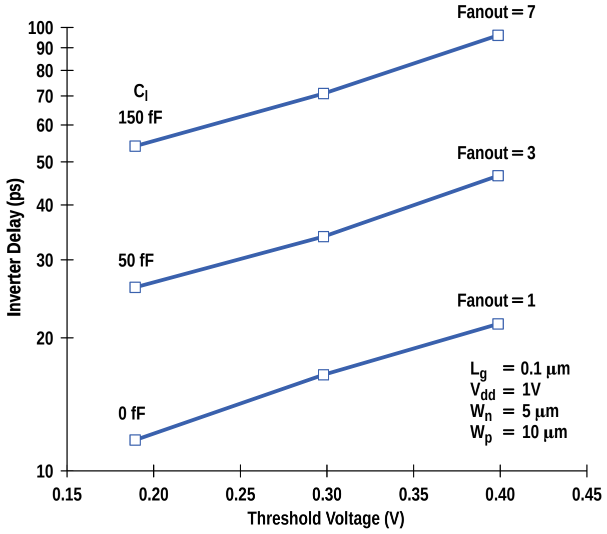


Fig. 13. *Inverter delay versus threshold voltage V_t for the 0.1-μm generation.*

Conclusions

We have explored MOSFET scaling into the future, extrapolating past scaling trends in channel length and gate oxide thickness. This scaling requires ever-shallower junction profiles and, below L_g = 0.15 μm, retrograde channel profiles. Constant-field scaling applied to V_{dd} and V_t continues the historical trend of about 30% improvement in gate delay per generation. In this era, MOSFET drive current remains nearly constant from one generation to the next and most of the performance improvement comes from the decreasing supply voltage. However, this performance comes at the price of exponentially increasing off-state leakage. Possible alternatives to this problem were discussed briefly but no clear resolution is available at this time. Clearly, this is an area where the design and technology communities must work together to develop an optimal roadmap for future device scaling.

Frequency Modulation of System Clocks for EMI Reduction

This paper focuses on clock dithering as an on-chip technique for EMI reduction. It is a survey paper based on information gathered from inside and outside HP's Integrated Circuit Business Division (ICBD). It reviews the basic concept, the work that has been done at ICBD and elsewhere, ICBD customer experiences, and lessons drawn from these experiences about design, effectiveness, and customer implementation with ICBD.

by Cornelis D. Hoekstra

The proliferation of electronic products in the home and office is putting increasing pressure on every product to reduce its electromagnetic interference (EMI). At HP's Integrated Circuit Business Division (ICBD), several different methods have been used to help deal with EMI directly on-chip, among them frequency modulation of the system clock, also called clock dithering, and control of pad output rise and fall times over process, voltage, and temperature (PVT) variations. This latter method is also called adjustable output pad (AOP) control, and sometimes includes programmable adjustment for capacitive loading.

This paper focuses on clock dithering as an on-chip technique for EMI reduction. It reviews the basic concept, the work that has been done at several different ICBD design centers and elsewhere, ICBD customer experiences with that work, and lessons drawn from these experiences about design, effectiveness, and customer implementation with ICBD. The paper closes with a brief review of the costs and benefits of implementing dithering and a summary of what customers can expect when working with ICBD. This paper does not aim to be a comprehensive description of dithering circuitry and mathematics, but rather a more narrative description of experiences and rules of thumb. See **reference 1** for a more detailed discussion of circuit implementation.

Typical Clock Dithering Circuit

The key idea, illustrated by Fig. 1, is the control of the frequency of the voltage-controlled oscillator (VCO) of a phase-locked loop by appropriate division of the reference clock (RefClk) by the input divider (Q) and of the VCO clock (fvco) by the feedback divider (P). The dividers all consist of digital counters. The divided digital waveforms are compared by the phase-frequency detector, which puts out an up or down signal pulse depending on whether the P waveform lags or leads the Q waveform. The width of the up or down pulse is proportional to the amount of lag or lead. The up or down pulse is fed to the charge pump and low-pass filter block, which translates it to a change in the VCO control voltage (vcntl). The VCO control voltage is repeatedly adjusted by up or down pulses until the VCO frequency fvco is such that the P and Q waveforms align (i.e., the up and down pulses are of nearly zero width). At this point RefClk/Q = fvco/P. Since the VCO frequency is divided by the output divider D, the actual output signal PllClk = fvco/D. Thus, the output frequency at stable operation is dictated by the values of the Q, P, and D dividers, and by appropriate substitution can be written as PllClk = P(RefClk)/(QD). Thus, for example, if RefClk = 16 MHz, P = 50, Q = 10, and D = 5, the output frequency PllClk is the same as the input frequency, 16 MHz.

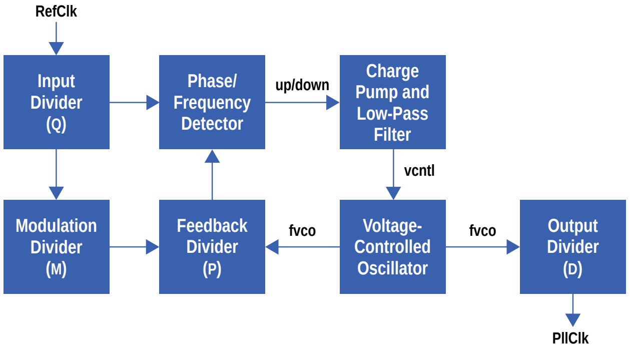
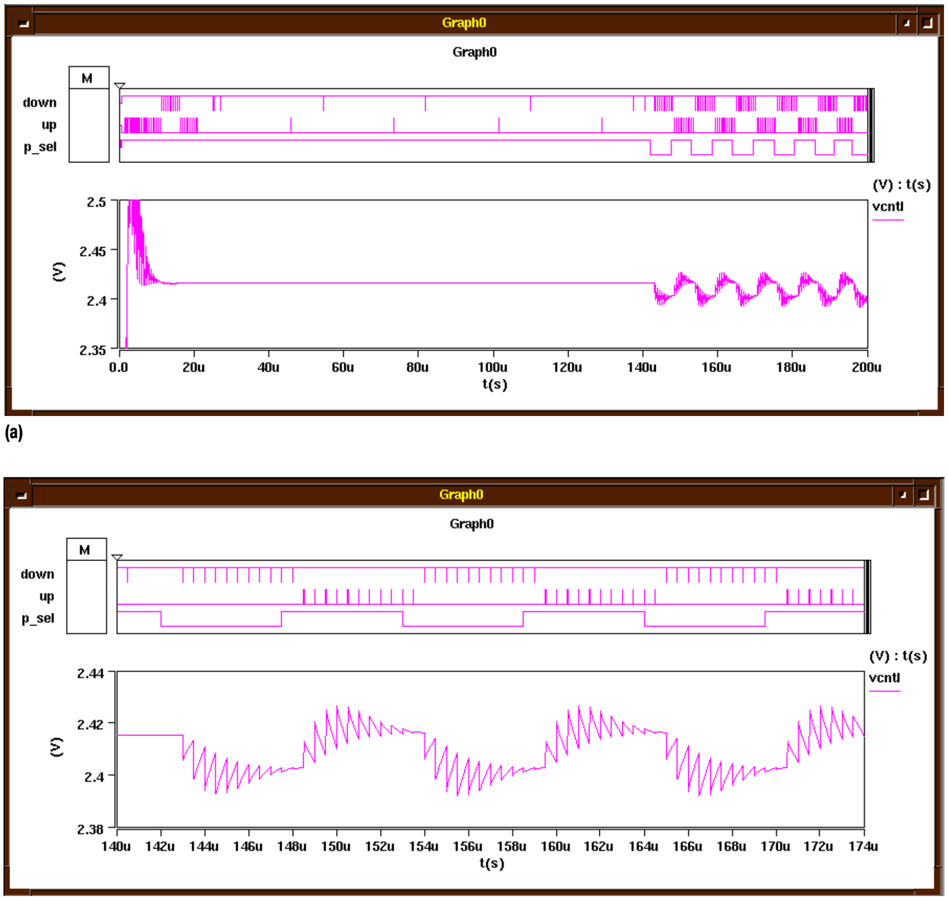


Fig. 1. *Block diagram of a typical dithering phase-locked loop.*

If the P counter endpoint is 49, the output frequency is 15.68MHz, 2% less than 16 MHz. Therefore, a simple way to achieve dithering is to change the P counter endpoint back and forth between 50 and 49 at some reasonable rate. Controlling this rate is the job of the M counter, that is, the P counter endpoint is changed each time the modulation counter (M) reaches its endpoint. A typical value for M might be 10, so that the modulation frequency is then 16 MHz/(QM) = 160 kHz. In practice, either the Q counter, the P counter, or both can be changed to achieve different target frequencies. Furthermore, by using both the rising and falling edges of the VCO clock, P can effectively have values such as 50.5 and 49.5, thus allowing a symmetric deviation of ±1% about a center value of 50.

The scheme described above can be thought of as square wave modulation because the phase-locked loop is asked to jump instantaneously from one frequency to another. Because real systems don't respond that way, and because of deliberate filtering to moderate this sudden transition, the actual frequency modulation waveform looks more like a ringing square wave. Typical simulation results for startup, lock, and modulation for this design, using the SABER analog/digital mixed-signal simulation tool, are shown in Figs. 2a and 2b. The VCO control voltage vcntl represents frequency, up and down are as described above, and p_sel is the output of the modulation divider (M). Frequency deviation of the dithered clock is typically ±1% to ±2%, and modulation frequency is typically 50 kHz to 250 kHz. Cycle-to-cycle jitter has ranged from well under 0.5% to as much as 2% for designs to date.



(a)

(b)

Fig. 2. *(a) SABER simulation of the startup, lock, and dithering regions of phase-locked loop operation using square wave modulation. The VCO control voltage vcntl is proportional to the output frequency. (b) Closeup of the square wave modulation waveform. Note the ringing square wave appearance.*

Square wave modulation as just described has been used successfully in a number of products to reduce EMI emission sufficiently to allow products to pass FCC testing when they otherwise could not. However, in some applications, the cycle-to-cycle jitter associated with this modulation method cannot be tolerated by the system (this is discussed further below). Over the last year this drawback has been addressed at ICBD by the development of triangle wave modulation. This method uses delta-sigma methods to step the phase-locked loop frequency more gradually from a low-frequency target to a high-frequency target and back again, resulting in what is usually called triangle wave frequency modulation. The technique greatly reduces cycle-to-cycle jitter of the phase-locked loop output clock compared to square wave frequency modulation. It also provides some improvement in EMI reduction because of flatter spectral response between the upper and lower frequency targets. This new phase-locked loop design has been successfully implemented in ICBD's CMOS14TB process on two ASICs for HP products. The new modulation method is less sensitive to process variation than previous methods, and should therefore be easy to port to future process generations and second-source fabrication facilities. Fig. 3 shows a closeup of the triangle-wave modulation waveform of this new design (startup is similar to square wave modulation).

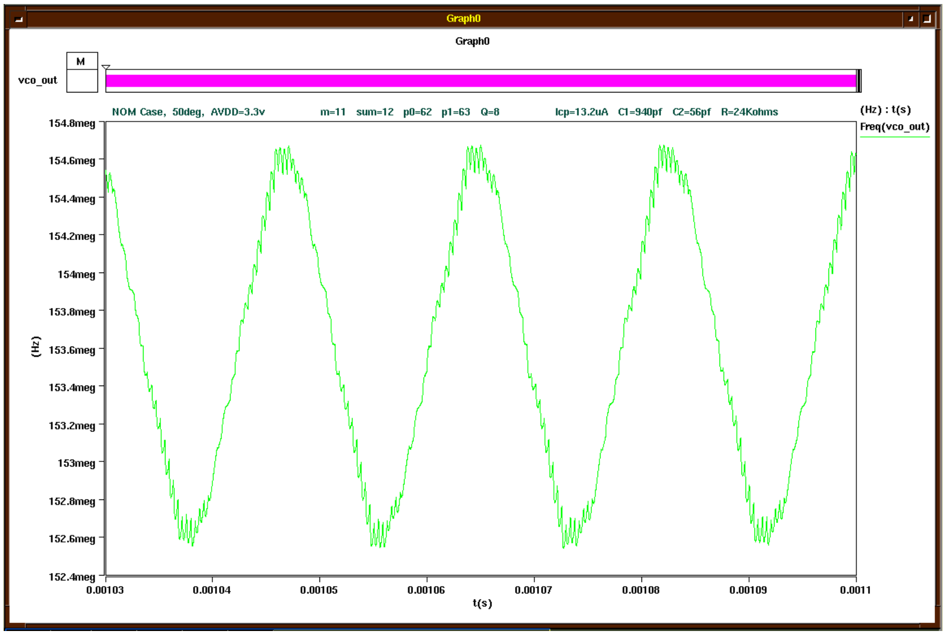


Fig. 3. *Closeup of the triangle wave modulation waveform. Startup and lock are similar to square wave startup and lock of Fig. 2a.*

HP Experiences with Dithering

ICBD Customer Divisions. A number of HP products have used clock dithering to date. For one product, two different modulation schemes were designed and manufactured by two independent organizations using different processes. For one design and process the modulation waveform looked like a ringing square wave that substantially overshot the target frequencies, while for the other design and process the modulation waveform was more triangular because of its use of a smaller-bandwidth filter. The advantage of the triangular version was that changes in period from cycle to cycle were gradual (less cycle-to-cycle period variation or jitter), and the spectrum was smoothly spread across many frequencies between the minimum and maximum. However, because the narrow-bandwidth filter loop response was so slow, for worst-case slow conditions the VCO frequency never reached its target value, limiting total frequency deviation and thus EMI reduction. In the ringing square wave version, the loop response was very fast, so that target frequencies were reached and even exceeded because of overshoot, over all process conditions. For this square wave scheme, the frequency was distributed over a wider range, although less evenly. Fig. 4 shows simulated VCO control voltage waveforms for these two different designs for qualitative comparison.

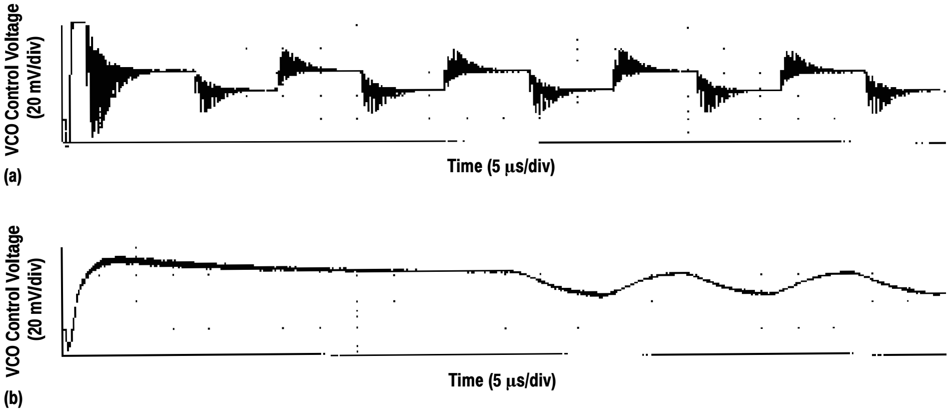


Fig. 4. *Simulated VCO control voltage plots for (a) a square wave modulation design and (b) a triangle wave modulation design. Note the longer startup and lock time for (b), reflecting its slower response.*

Conducted EMI measurements of the frequency spectrum of the system clock pin showed lower peak values for the square-wave-modulated part than for the triangular-wave-modulated part, which appeared to be a result of greater spectrum spreading because of square wave overshoot. However, radiated emissions from the boards using parts designed with triangular wave modulation exhibited less noise overall. Although the reason for this was not proved, it appeared to be the result of slower overall switching speeds of the process used to manufacture this version of the design. Nevertheless, for parts from both processes, dithering had a beneficial effect on EMI. Fig. 5 shows conducted frequency spectra for one of the harmonics of the dithered clock measured on the clock pin of each part.

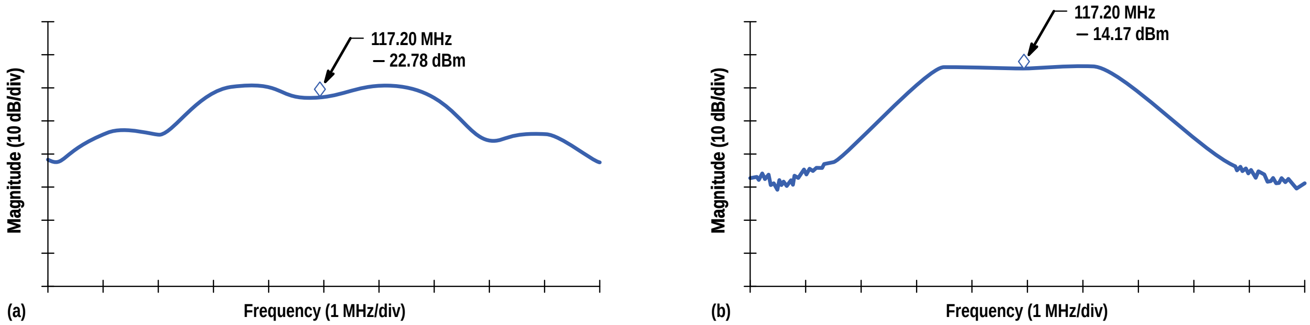


Fig. 5. *Conducted spectrum of a dithered clock harmonic for (a) a square wave modulation design and (b) a triangle wave modulation design. Spectra were filtered through a CISPR16-compliant quasipeak detector (see “**EMI Measurement Standards**” on page 6).*

For the product described above, the EMI reduction observed for square wave modulation did not seem to match the reduction predicted mathematically by standard FM theory based on the deviation and the modulation rate. Therefore, for another product, a dithering phase-locked loop using square wave modulation was made programmable to a number of different deviation and modulation values to make it possible to explore EMI reduction based on these two parameters. This gave the interesting result that EMI reduction was optimum somewhere between very slow and very fast modulation, contrary to standard FM theory. The reason for this is not really very surprising and is discussed further below (see “***Design Considerations***”).

The complex relations described above between overall EMI reduction and modulation waveshape, process characteristics (e.g., intrinsic switching speeds), measurement method (e.g., conducted versus radiated spectra), and modulation rate caused substantial confusion and disagreement about which modulation method was better for EMI reduction, and was a primary stimulus for writing this paper.

Other HP Divisions. Another HP division has taken a different approach. This division developed an all-digital gate-array part that receives a 40-MHz input reference signal and outputs a clock that varies pseudorandomly near 14 MHz. This pseudorandom 14-MHz signal is then fed to an IC containing a phase-locked loop, which smooths the pseudorandom 14-MHz signal and thus effectively generates a dithered system clock. The pseudorandom 14-MHz signal is generated in a deterministic way that makes it exactly 14.31818 MHz on average, so that it can be used as a real-time clock. Modulation is synchronized to the horizontal sync signal of the video display so that no random jitter is observed in the video picture. Fig. 6 illustrates the appearance of the pseudorandom 14-MHz clock compared to the 40-MHz input clock and the ideal 14.31818-MHz clock. Two modulation cycles are shown.

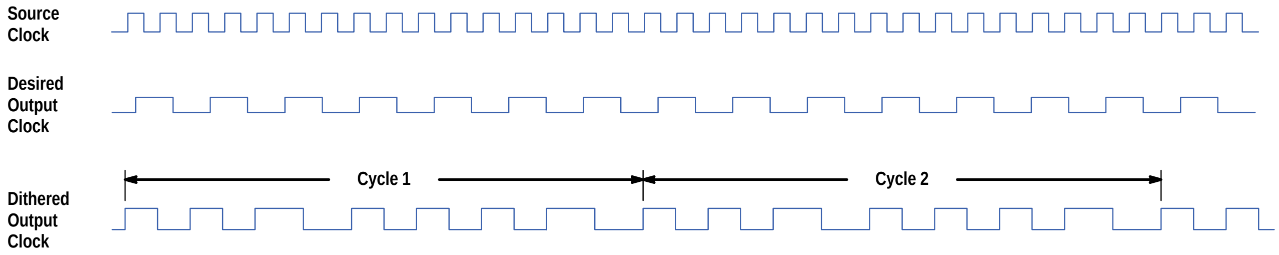


Fig. 6. *Pseudorandom clock with average frequency of 14.31818 MHz, digitally generated from 40 MHz. Two modulation cycles are shown.*

Non-HP Clock Dithering Products. A standalone product that provides a clock whose frequency varies very smoothly over various ranges of center frequency and deviation is available in the industry. However, the product is expensive, takes up board space, and requires additional surface mount parts for operation, further adding to board costs. In addition, the modulation frequency is fixed and cannot be synchronized with product operation, such as the horizontal sync signal of a video display to prevent visual distortion due to jitter. The recently developed ICBD dithering phase-locked loops described above offer smoothly varying frequency modulation without these disadvantages, and at very low cost.

Design Considerations

EMI Reduction versus Modulation Waveform. A square wave can be described as a linear superposition of the odd harmonics of a fundamental sinusoid whose frequency is equal to the frequency of the square wave. Thus, a lot can be understood about square wave modulation of a square wave by considering square wave modulation of a sine wave. The discussion below is given with this in mind.

FM theory predicts that the power of a sine wave whose frequency is modulated by another sine wave is distributed across individual small peaks between the minimum and maximum frequency endpoints, separated by a frequency difference equal to the modulation frequency. Thus, as modulation frequency is decreased, there are more power peaks, but with lower peak values and spaced closer together. On the other hand, the spectrum of a sine wave whose frequency is modulated by a square wave contains just two peaks, regardless of how slowly the sine wave is modulated. These peaks are at the minimum and maximum frequency deviation points, and each contains half the total power of the unmodulated sine wave. This can be intuitively understood by realizing that the modulated signal spends virtually all of its time stabilized at one or the other of the two frequency endpoints.

As the modulation rate is increased, a real system designed to do square wave modulation cannot actually respond instantaneously in true square wave fashion and spends relatively more time in transition between frequencies and less time stabilized at its frequency extremes. Thus, the system starts to look more like a sinusoidally modulated system, and the two square wave power peaks tend to distribute into multiple smaller peaks. Finally, as the modulation rate is increased even further, the number of peaks will tend to decline again and their individual peak values will increase, in accordance with FM theory as discussed above.

In other words, in a real system designed to do square wave modulation there is a point of maximum EMI reduction between very fast and very slow modulation rates. The exact location of this point varies depending on phase-locked loop and product characteristics. This phenomenon was verified for the product with programmable parameters described above. For this product's particular phase-locked loop and product characteristics, measurements showed that the greatest reduction occurred at a point where the ratio of frequency deviation to modulation frequency was about 1.4.

As discussed near the beginning of this article, aside from reducing cycle-to-cycle jitter, the recent development of triangle wave modulation at ICBD also improves on the EMI reduction limitations of square wave modulation just described by more smoothly spreading spectral energy across the entire range of frequencies between the maximum and minimum endpoints at low modulation rates.

Programmability. It is valuable to provide modulation, deviation, and dithering on/off programmability in the phase-locked loop. These features should be easy to control during both phase-locked loop test mode and normal operation to allow rapid and effective evaluation of silicon and to optimize the product's EMI characteristics. This kind of characterization adds knowledge to the phase-locked loop database, and with appropriate programmability can also be used to assess product margin over a range of fixed operating frequencies.

Frequency Synthesis. Frequency synthesis is a built-in option for dithering phase-locked loops. The same basic method used to create frequencies slightly smaller or larger than the reference can be used to synthesize nearly any arbitrary frequency within phase-locked loop performance limitations. This allows the use of lower-frequency crystals (<20 MHz or so) operating in fundamental mode to generate the frequency reference. These crystals are typically less expensive, require fewer extra components, and cause fewer startup problems than higher-frequency crystals, which need to operate in overtone modes.

Spectrum Overlap. When deciding on deviation values, the designer should keep in mind the potential for spectrum overlap at higher harmonics. This can occur when the output frequency is relatively low and the frequency deviation is relatively high, and can tend to cancel out the expected EMI reduction for high-frequency components.

Mixing Dithered and Nondithered Clocks. Dithered and nondithered clock domains on the same chip usually must be treated as unrelated clock domains, and therefore should be avoided if possible. Consider the case of two clock domains, one dithered and one undithered. Given a point where the rising edges of the two clocks go up simultaneously, the edges of the dithered clock that follow will alternately lead or lag the corresponding edges of the reference clock over time. This is sometimes referred to as clock slip. Clock slip is both difficult to control accurately and difficult to measure accurately, particularly in a production environment. Designing a system with asynchronous domains is typically messy, complicated, and hard to simulate, so it should be avoided if possible. For example, in the case of video clocks, rather than have a nondithered clock to prevent visual jitter in the video, modulation can be synchronized to the horizontal sync signal. On the other hand, if separate clock domains are necessary, they can be used—they just require more careful engineering.

System Simulation. We recommend that our customers simulate a behavioral/structural Verilog model in their chip designs to catch unexpected problems. Examples of problem areas exposed when simulating with such a model are improper multiplexing and I/O control, inadequate pad drive strength for fast (system clock speed) output test signals, asynchronous interfaces, and marginal performance with respect to operating frequency.

A mixed-signal (i.e., analog and digital) simulation tool is indispensable for phase-locked loop design and simulation. The tool should have links to Verilog and C, and ideally should offer built-in FFT analysis capability, which can be useful for evaluation of the spectral characteristics of various dithering alternatives.

Multiple Phase-Locked Loops. Careful attention must be paid to systems with multiple phase-locked loops. Some major system IC components (e.g., microprocessors) contain phase-locked loops of their own, used for things such as frequency synthesis and generation of nonoverlapping clocks. These phase-locked loops must be able to track the dithering of the reference phase-locked loop's output adequately so that clock skew does not become excessive across the system. Unfortunately, assessing this phase-locked loop tracking ability in simulation can be difficult. A dithering phase-locked loop with very low cycle-to-cycle jitter (i.e., very slowly changing frequency) can help avoid the need for this simulation, and is a major reason for ICBD's development of triangular modulation.

Product Evaluation

Silicon Process Variation. Process speed can be an important factor affecting the apparent effectiveness of dithering. For current designs, process speed has a fairly strong effect on VCO gain as well as clock tree and output driver switching speed. Thus, we recommend that our customers make a point of looking at both fast and slow parts when evaluating the effectiveness of dithering.

EMI Measurement Standards. The CISPR16 EMI measurement standard is not absolute, and different measurement tools may all meet the standard yet give different results. The method by which EMI emissions are to be measured is defined by a standard called CISPR16-1.[2] This standard is intended to approximate the characteristics of typical radio-frequency receivers. For frequencies from 30 MHz to 1 GHz, power is averaged for a passband whose nominal width is 120 kHz at 6 dB. However, the standard allows passbands ranging from 100 kHz to 140 kHz at 6 dB, so results vary depending on the measurement filter chosen. Peak values that change at a rate faster than 10 to 20 kHz are ignored by using a quasipeak detector defined by the standard. The time-constant characteristics of the quasipeak detector are again given as a range of allowable values. This ambiguity in both filter and peak detector characteristics means one should be careful when comparing EMI measurements from different tests.

Conducted versus Radiated Spectra. It is important to differentiate between conducted and radiated spectra. When a probe is directly touched to the clock pin of a part, the conducted spectrum observed is a fairly direct representation of the spectral composition of the clock signal. However, when electromagnetic emissions are monitored at a distance from a finished product, the clock signal has been significantly "filtered" by the antenna characteristics of the product and the measurement environment. In other words, products act as frequency selective antennas, so conducted and radiated spectra can be and usually are quite different. This increases the desirability of phase-locked loop programmability to find optimum performance. Consequently, this also makes ease of controllability and access for measurement important.

Effectiveness as a Function of Frequency. Dithering is intrinsically more effective at higher harmonics and less effective at lower harmonics. This is simply because the absolute value of frequency deviation increases linearly with harmonic number, so that spectral energy is spread over a larger range at higher harmonics, while the width of the filter over which spectral energy is measured is fixed. Fortunately, high frequency is exactly where certain customers have their most severe problems. For example, one HP printer division tends to have many components on the printer board, and relies heavily on shielding to limit emissions. Low-frequency noise seems to be effectively contained, but high-frequency (short wavelength) noise tends to leak out through openings in the shielded box. For another HP printer division, on the other hand, low-frequency radiation turns out to be the primary noise source, at least in part because of unique resonant conditions created by printer cabling, with the result that dithering is less effective.

Dithering versus PVT/AOP. The PVT or AOP technique (defined at the beginning of this article) consists of controlling the turn-on times or the rise and fall times of IC output drivers (pads), ideally keeping these times constant over PVT variations, and sometimes adjusting for capacitive load as well. This method is also intended to keep ground bounce and signal reflections constant over PVT variations. It requires circuitry to monitor the PVT operating point of the IC (for example, by counting cycles of a free-running ring oscillator with respect to the reference clock), and then adjusting the driver or predriver current to control the drive strength or turn-on time of the driver, respectively. This technique requires considerable effort for pad design, customer simulation, and characterization of first silicon to accurately correlate the PVT reference to pad programming. With the practice of second-sourcing becoming more common, much of this work has to be repeated for each foundry. Unfortunately, EMI reduction has been negligible, at least as observed for HP DeskJet printers, a major user of this method. For these products, system clock noise has turned out to be a much greater source of EMI than pad switching noise, and system clock noise is not helped by this method. In fact, the low output resistance of typical PVT pads may encourage the transmission of system clock noise from the power net out through the output drivers and onto the product board. In summary, PVT or AOP is still believed to have potential benefit, especially for very fast-switching outputs, but is not likely to realize its potential until drive control is fairly automatic inside each pad, without requiring chip-level programming intervention.

Dithering addresses most of the limitations of PVT. The system clock tends to be the top noise source because by design everything happens at rising and falling edges of the clock. This means that clock dithering will tend to spread out all sources of noise throughout the system, since all of them are related to the clock. Thus, the advantage of dithering circuitry is that it is essentially a single independent block that can be inserted into an IC design to help reduce EMI globally, at both chip and board levels.

Verifying Testability and Compatibility. Customers need to set aside engineering time to design their IC to make the phase-locked loop accessible for testing and to ensure that the IC will work with a dithered clock. In production testing the phase-locked loop internals are typically tested while the IC is in a special phase-locked loop test mode. During this mode certain pins of the IC are multiplexed to the phase-locked loop block's input and output ports for direct access on a production tester. Special test decks written by ICBD are then applied to the part. The customer needs to design the IC to accommodate this phase-locked loop test mode configuration. The customer also is advised to simulate the IC design at the extremes of frequency expected from the dithered clock, and to include uncertainty in the clock edge to account for cycle-to-cycle jitter. Finally, recall the earlier recommendation that if dithering is used it should be applied to the entire clock domain. If that is not possible, customer effort will be required to design asynchronous interfaces that do not rely on controlled phase relations between the dithered and nondithered clock domains.

Customer Evaluation. At this point in the evolution of clock dithering, customers should plan to spend some extra time beyond the usual EMI characterization of their product to characterize their systems with dithering. As experience is gained both by customers and ICBD, this need should decline.

Acknowledgments

This is a survey paper based on information gathered from many different people inside and outside ICBD. People within ICBD who provided information were Mike Oshima, Tom Thatcher, and Doug Sojourner from California Design Center, Charles Moore and Derek Knee from Fort Collins Design Center, and Linda Engelbrecht from Corvallis Design Center. The development of triangle wave modulation was by a team headed by Linda Engelbrecht, using a concept originated by Charles Moore. I have also obtained information from Bob Dockey and Ron Juve of Vancouver Division, Tom Wheless of Business LaserJet Division, Steve Smith of San Diego Printer Division, and Dave Arnett and Bob Puckette of Mobile Computing Division. I would like to take this opportunity to sincerely acknowledge and thank each of these people here for their contribution of information and comments for this paper.

Reference

1. T.J. Thatcher, M.M. Oshima, and C. Botelho, "Designing, Simulating, and Testing an Analog Phase-Locked Loop in a Digital Environment," *Hewlett-Packard Journal*, Vol. 48, no. 2, April 1997, pp. 84-88.
2. *International Special Committee on Radio Interference, CISPR16-1: Specification for radio disturbance and immunity measuring apparatus and methods, Part 1*, International Electrotechnical Commission, Geneva, Switzerland. First edition 1993.

Fully Synthesizable Microprocessor Core via HDL Porting

Microprocessors integrated in superchips have traditionally been ported from third-party processor vendors via artwork. A new methodology uses hardware description language (HDL) instead of artwork. Having the HDL source allows the processor design to be optimized for HP's process in much the same way as other top-down designs.

by Jim J. Lin

The level of integration has been rapidly increasing with advances in semiconductor technology. Many HP design groups have capitalized on this capability to create highly integrated ASICs, or superchips. Superchips that integrate conventional ASIC logic, microprocessors, embedded RAM and ROM, and other megacell functions save cost, power, board space, and inventory overhead and increase I/O performance at the same time. This industry-wide trend has put an increased burden on ASIC suppliers to come up with megacells that are off-the-shelf, proven, and testable. As an ASIC supplier to other HP divisions, we at HP's Integrated Circuit Business Division (ICBD) licensed several early microprocessors that customers demanded and artwork-ported them into our process. All superchips built at ICBD today contain artwork-ported microprocessors.

However, artwork porting has its limitations. It often does not yield the best area possible in a given technology. In addition, the process technologies of processor vendors tend to diverge from ICBD's technology for the next generation. The testability of these processors is also a problem in a superchip because they require access to their functional pins to run parallel vectors. Multiplexing the processor pins with ASIC pins is a level of complication preferably avoided. Customers also demand some controllability of the exact microprocessor configuration that goes into the superchip. For example, customers may choose to increase or decrease the size of the cache that's included with the microprocessor after profiling target code under different cache configurations. Making such changes at the artwork level is a major undertaking and often requires a lot of work for the processor vendor as well. Presilicon verification is also virtually nonexistent and the design often requires several mask turns. Finally, the processor is technology dependent and requires almost as much effort to go into a different process as the original port.

One methodology that successfully addresses these issues is HDL (Hardware Description Language) synthesis. A number of underlying technologies make this methodology work. The increased density in our standard cell technology allows the implementation of dense data path functions and is area-effective in general. ICBD's standard cell design flow for top-down design methodology is robust and mature. Synthesis is becoming more and more powerful and is capable of highly complex designs. ICBD's RAM generation is also a key component that delivers good performance for cache applications. Processor vendors in the embedded market have shifted their design paradigms as well. They are no longer custom designing everything and are using HDL and synthesis more and more.

The methodology of porting cores using HDL synthesis incorporates an existing standard tool flow. This is a major advantage. An efficient tool flow is essential in today's ASIC market. Significant effort has gone into making ICBD's as efficient as possible to handle the high-integration market. The goal for this methodology is to leverage the standard tool flow as much as possible. In essence, processor HDL is transferred from the processor vendor. After necessary changes in certain configurations at the HDL level, the HDL is verified through functional simulation. It is then fed into the synthesis tool to obtain a netlist that is subsequently fed into the conventional tool flow.

The rest of this paper will discuss in more detail the methodology and how it was used to implement the Coldfire 5202 microprocessor from Motorola in a test chip.

Methodology Overview

Since the processor cores developed from this methodology are going to be used in superchips, they need to be designed with ease of integration and customer needs in mind. Testability, customizability, technology independence, minimum die size, and thorough presilicon verification are all goals that are important to delivering a successful core for superintegration.

The testability of a processor core has different constraints than a standalone processor because the functional pins are not visible when the core is integrated on a superchip. Pins of processors have traditionally been multiplexed out in test mode, which takes additional effort, and fault grading the functional vectors is not always easy. Our new methodology uses full-scan test patterns that require only a few scan ports that are needed anyway for other ASIC logic and can be effectively fault graded to determine the quality of the vectors. This approach to testing is compatible with HP testing standards and minimizes the cost of testing.

Having the HDL for the processor means that changes to the processor can be done at the source level rather than the artwork level. While changes to the instruction set architecture are nontrivial and are not recommended by the processor vendor, changes to the cache and bus configurations are within the realm of possibility. Customers can often save area by cutting out an unused block on the processor or reducing the cache size. The functionality of the design can be verified before silicon to bolster confidence for first-time success in the customized processor.

Our methodology also enables technology independence through logic remapping. The HDL can be simply recompiled to target a different technology, assuming that the technology has a standard cell library capable of synthesis. This is true for porting a processor to a new technology at ICBD, a second source, or a dual source.

Area is a very important consideration as well. If the area of the synthesized core is not competitive with custom-laid-out processors, customers will not adopt this strategy regardless of how good the methodology. Standard cell density has increased to the point where even data path blocks like ALU, barrel shifter, and register files can be implemented in a reasonable amount of area. Another flexibility in using standard cells is that a processor core can be compiled with the desired target frequency. If the nominal frequency is faster than the target, cells can be sized down to save some additional area.

Having the HDL source for the processors also means that the design can be simulated, both at the RTL (Register Transfer Language) level initially and at the gate level at the end. Vectors can be run to verify the functionality and timing of the design.

To ensure functionality, three different approaches can be used, either alone or in combination. RTL and netlist verification can run precaptured vectors from the processor vendor. These vectors are diagnostics, benchmarks, or random instructions that processor vendors themselves use. The netlist can also be compared with the vendor's design using formal verification methods. Lastly, an environment in which random instructions are generated can be set up locally to subject the design to new random instruction testing.

Timing is verified with a combination of static and dynamic timing analysis. An even greater advantage for the customer is the ability for the entire superchip to be simulated in a timing-accurate fashion since the processor is in the same library as the ASIC logic. Previously, such system-level simulation was only possible using a hardware modeler or a software model that was not timing-accurate.

Design Flow

An ICBD design automation group has a series of supported design tool flows. A modified version of their Standard Tool Flow 2 (STF2) is used to carry out our methodology. In this way, our methodology leverages a proven methodology for doing HDL-based design. Synthesizing microprocessor cores becomes an extension of the current capability.

This paper focuses only on those aspects of the methodology that are unique to porting microprocessor cores. The process is illustrated in Fig. 1. The methodology incorporates standard ICBD tool flows. For Verilog-based designs, the STF2 is used. For VHDL (Very High-Speed Integrated Circuit Hardware Description Language) designs, an HP proprietary VHDL tool flow is used. These flows are simply encapsulated as design processes in Fig. 1.

Inputs. The processor vendor needs to have the following list of items to feed into our tool flow:

- Design Specifications. Timing, functionality, and pin descriptions of the processor. Most of this information can be obtained from a data book if available. For newer cores a data book may not be available, but internal documentation that will eventually be part of the data book will suffice.
- Behavioral HDL. A Verilog or VHDL model of the processor core. This HDL model does not necessarily have to be synthesizable. As long as it models the cycle-to-cycle behavior of the design, it can be made synthesizable with some rewriting of the HDL.
- Functional Vectors. Verification vectors in Verilog or VHDL format. These are run on their respective simulators. They can also be translated so that they can be run on the testers as well. These vectors enable presilicon verification.
- Synthesis Scripts (optional). These are used to synthesize the HDL into standard cells. They are only available for cores that have been previously synthesized.

Outputs. For use in superchip integration and product prototyping, the ICBD CPU team provides design groups and customers with the following:

- Megacell. Processor core with all requirements for inclusion in the HP intellectual property library. Deliverables include ERS, gate-level netlist, behavioral model, data sheet, etc.
- Test Chip Data. Mask, packaging, test vectors, and test program input for test chip fabrication and test. A core without a test chip will not have this.

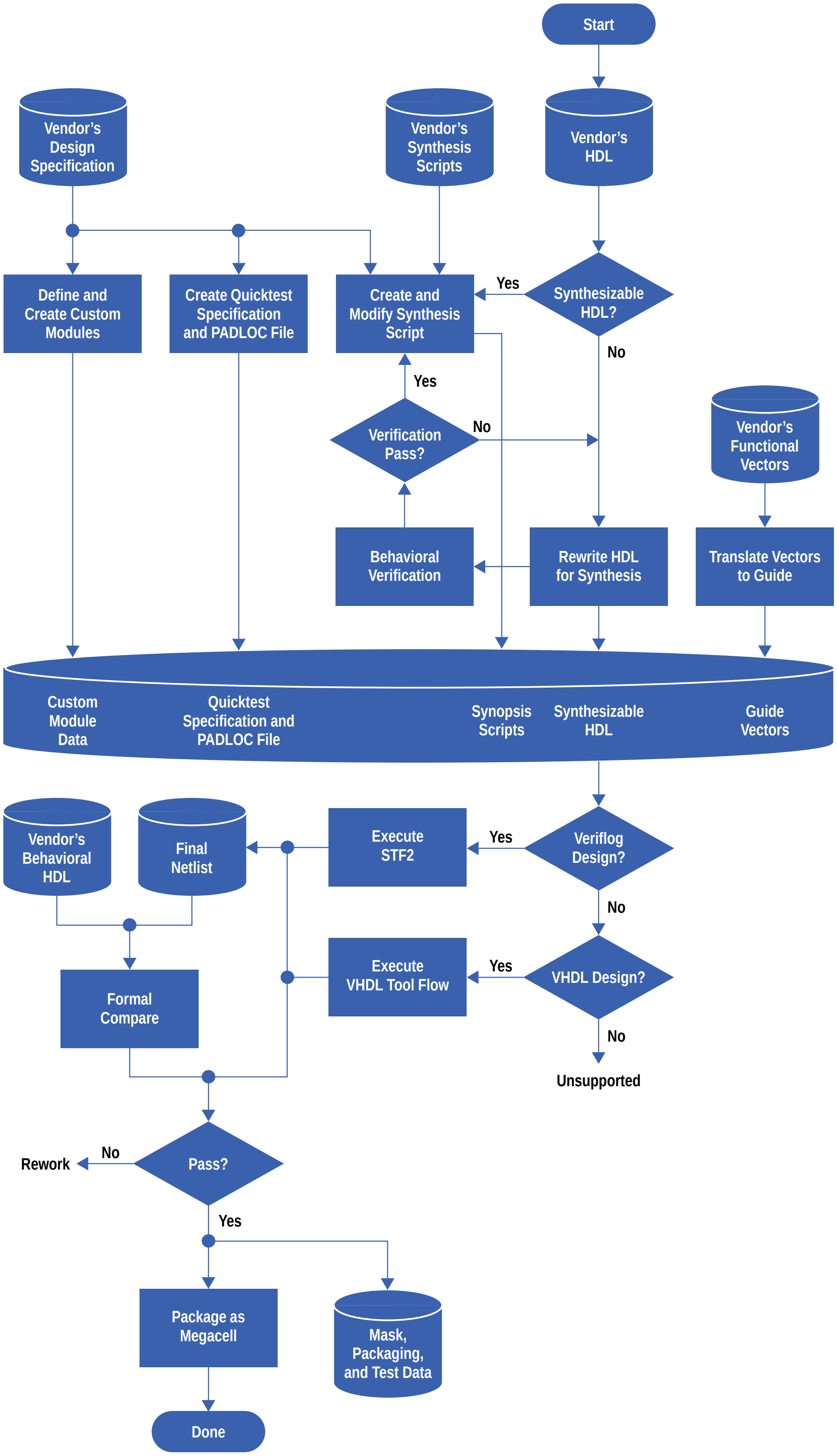


Fig. 1. *Processor core tool flow.*

Rewrite HDL for Synthesis. The HDL that is transferred from the processor vendor may not be synthesizable since it may have been written only as a model, not as a synthesis source. Current synthesis tools have limitations on the type of HDL constructs allowed and yield very poor-quality circuits for HDL not written with synthesis in mind. ICBD has a set of HDL coding guidelines that need to be followed when rewriting the HDL. This step may warrant some iterations in synthesis to explore the optimal mapping of specific behavioral constructs. Regardless of what changes are made to the HDL or the quality of synthesis achieved, the changes should not alter the functionality. In fact, any changes should be thoroughly verified by running regression vectors as discussed next.

Behavioral Verification. Even though behavioral verification is a part of STF2, this portion of the flow focuses on the extra verification that results from recoding for either customization or synthesizability. The verification is an extensive simulation of the altered HDL code with the vendor-provided vectors. In the future, this step may be augmented by formal verification. This subject will be revisited when the verification of the test chip (discussed later) is analyzed.

Create and Modify Synthesis Scripts. The purpose of this task is to create Synopsys Design Compiler scripts that can be used to compile the standard cell portion of the core consistently and systematically. As mentioned earlier, there may not be existing synthesis scripts if the processor has never been synthesized before. In other cases, there will be a full suite of scripts along with design constraints. In still other designs, portions of the control logic will have synthesis scripts and the data path will not. This represents the design methodology of many processor vendors.

In case synthesis scripts need to be created, design specifications and the actual HDL are the best sources. ICBD has a generic synthesis script that can be used as a template to help create these scripts. Even when all of the scripts exist, modifications are often needed to make them work in ICBD's environment and libraries. The processor vendor's approach to synthesis may not be the most efficient approach in ICBD's technology. Furthermore, the vendor's approach may not use the most up-to-date synthesis technique available in the latest release of the synthesis tool. Many trial-and-error runs of different approaches may be needed to determine the best synthesis approach.

Create Quicktest Specification and PADLOC File. Quicktest generates a test template given a target tester. The PADLOC (pad location) file is an input to Quicktest and other tools such as the router. This step is only needed if a test chip is planned for the particular processor core. The need for a test chip is determined on a core-by-core basis. The first core in an architecture, a core with major customization, and a customer need for prototyping are all reasons for a test chip.

The Quicktest specification is used to generate a test program for the processor core test chip. The Quicktest specification documentation describes how to create the Quicktest file from the design specification. The PADLOC file is used to place the test chip pads around the core logic. It contains placement data for the pad ring. The PADLOC specification document describes how to create a PADLOC file from the design specification.

Identify and Create Custom Modules. Not all blocks in a microprocessor can be implemented as standard cells, although the list of such blocks is becoming shorter and shorter. This task identifies all blocks within the core that should be implemented as custom logic and creates the identified blocks along with all the models and information required to use them in the downstream standard cell design methodology.

To identify blocks that should be custom, the HDL and design specification must be studied along with the design goals. Blocks are custom-designed to meet area or performance goals unachievable with standard cells. Typically these blocks will be limited to memory arrays. However, they may also include structured data path logic for implementing highly regular or speed-critical circuits, such as a large multiport register file, multiplier, or barrel shifter. This step may involve feasibility studies in which candidate blocks are partially designed or estimated for both standard cell and custom implementations. The following items must be produced for each block selected for custom implementation:

- Verilog or VHDL model with timing
- Synopsys timing with pin timing
- Cell3 LEF file (Cell3 is an automatic place-and-route tool from Cadence)
- Artwork database
- Sunrise (an automatic test vector generator from Sunrise) or ATG (an in-house HP tool similar to Sunrise) fault model.

Translate Vectors. By translating simulation-based verification vectors into Guide format, the standard path to testers and later portions of the tool flow is established. Guide is an in-house HP tester independent vector translation tool. This step may require that custom programs or scripts be written and supported to translate from unknown formats. Therefore, it is advisable to require the processor vendor to supply vectors in some known format like Verilog, for which there is a clear path to Guide. The vectors are needed for functional and diagnostic purposes only. Manufacturing test will not run these vectors and will rely on I_{ddq}, stuck-at, and at-speed scan testing.

Execute Standard Tool Flow 2 (STF2). As mentioned above, Standard Tool Flow 2 is a design flow supported by an ICBD design automation group. It includes Verilog simulation, Synopsys synthesis, Cell3 place and route, ATG and Sunrise full-scan testing, artwork and mask generation, and Quicktest and Guide test program and vector creation. There is extensive documentation on the entire tool flow. A variation of STF2 is STF3, which supports partial-scan testing. This may be an option for cores that would realize significant area savings from it without the loss of appreciable test coverage. The VHDL tool flow is derived from an HP proprietary VHDL design flow. So far, no core has been developed using this design flow.

Package Core as Megacell. The final step in the process is to create the data necessary to offer the core as a megacell to HP customers and ICBD design centers. The requirements for this type of product are currently being defined. The release of any core will adhere to the standards set up and provide all the models, documentation, and support required.

Test Chip Experience

The Coldfire test chip (Fig. 2) is a test chip implementing the Coldfire 5202 processor from Motorola. Coldfire is a new line of embedded microprocessors that improves performance over the 68000 architecture while maintaining compatibility with most of the 68000 instruction set and minimizing area. The Coldfire 5202 has a 2K-byte 4-way set-associative cache, a debug unit, and JTAG capability (IEEE 1149.1 boundary scan test capability) along with the core as implemented by Motorola.

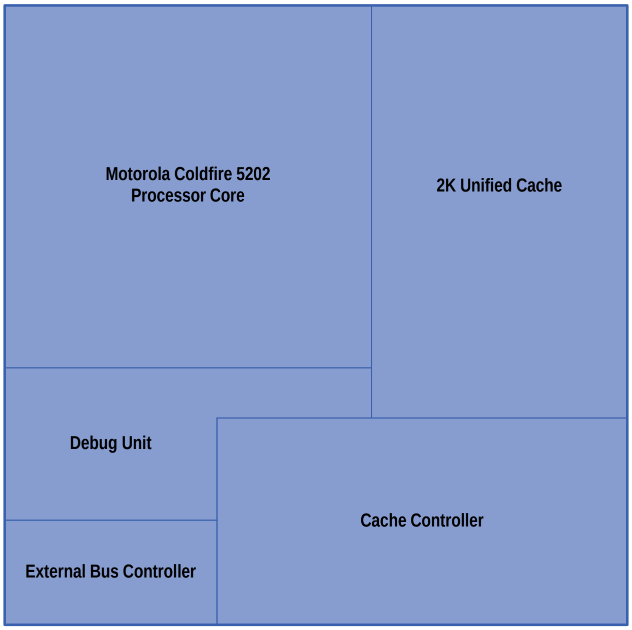


Fig. 2. *The Coldfire test chip floor plan.*

Design Transfer

The Coldfire test chip team received a brief course on the architecture and a tape of the HDL source for the Coldfire 5202. The tape contained synthesizable HDL for every block in the design. The cache memory blocks and some tag maintenance logic were custom-designed at Motorola and had only a behavioral representation. Each synthesizable block also had a constraint file used in Synopsys. There was also a top-level synthesis script. This represented all that was needed to get started with the port. Subsequently, Motorola has sent test vectors and fielded questions from ICBD. The level of support at Motorola has been very good.

Making the Coldfire Test Chip

The Coldfire test chip has been targeted as a test vehicle for this methodology and offers early prototypes for customers interested in using the Coldfire 5202. The Coldfire test chip is implemented in a 0.5-μm technology and takes advantage of the process's increased density. A fairly high frequency of 50 MHz was targeted to show the scalability of the methodology.

Custom Modules. The team knew at the outset that Motorola had designed custom cache RAMs and tag logic and that custom designing caches for every core would not fit into our methodology. The generated WEST SRAM available from an HP RAM group provided the solution. These RAMs are designed for ASIC integration and good performance. However, to use these RAMs, there were two extra requirements that were specific to this cache. First, it had to be byte-writable, that is, each byte of a multibyte word must be written separately. The second requirement is that the bits of one column needed to be reset in one cycle. This is used to invalidate the cache valid bits at startup or in case of an invalidate instruction. The RAM group took the first request and incorporated byte writability (actually bit writability as implemented in the artwork) into the generator. The one-cycle invalidate was not changed in the generator. Instead, the valid bits were turned into flip-flops on this test chip for schedule reasons. This incurred an area penalty and will hopefully be fixed in the RAM in the future.

Rewrite HDL. The interface to the cache RAM was asynchronous in the Motorola implementation. The WEST SRAM is synchronous. This means that addresses, data, and other control signals need to be set up before and held until after the rising edge of the clock. Motorola latches the various signals on the rising edge of the clock before feeding them to the RAM. With this scheme, the signals would have missed the setup time on the WEST SRAM. If no latch is used, then the hold time cannot be met. As a result, negative-level-sensitive latches are used to provide the necessary hold requirement. The control signals that Motorola uses are sufficient to generate the WEST SRAM control signals. The 4-way set-associatively can be easily implemented as four data RAMs and four tag RAMs, each representing one way in the cache organization.

Another change made to save some area was the removal of the JTAG block. The JTAG methodology does not fit into the superintegration process since this process has its own mechanism of doing boundary testing. The JTAG logic is a fairly modular block that can be stripped out with minimal perturbation to the rest of the design. The necessary logic needed in place of the JTAG block is encapsulated in its own level of hierarchy. The necessary JTAG-like functionality has been replaced by a scan wrapper that better fits ICBD's superchip test methodology.

One final change is in the area of testing. Motorola uses a test mode called ad-hoc mode to test the cache RAM circuitry. ICBD uses BIST (built-in self-test) instead. As a result, the logic that makes the cache RAM controllable and observable and interlocks the pipeline is no longer needed. By rendering the ad-hoc mode inactive, all logic associated with this test mode can be minimized.

Synthesis Script Modification. The synthesis script that came from Motorola put highly detailed constraints on each block. The reasons are twofold. Motorola was concerned with the speed of the synthesis job if the entire design were compiled at once and wanted to be able to do most of the compilation at the block level. Secondly, Motorola had good ideas about where Synopsys should be spending its time and wanted to influence the tool in that direction. A makefile generator was used to piece together the numerous block-level compiles and their respective constraint files. This approach does not necessarily yield the best design in ICBD's technology, as was found during initial synthesis trials using Motorola's script. The block-level constraints were often unrealistic and Synopsys was spending optimization cycles on the wrong circuits. As a result, the synthesis scripts were overhauled to put constraints only at the top level, that is, the I/O specifications of the chip.

A hierarchical compile at the top level replaced the block-level compile. As a result, Synopsys has more freedom in partitioning the time. The compile time is long but not intolerable. The entire synthesis job from reading in HDL to outputting an optimized netlist at 50 MHz takes 48 hours on an HP 9000 Model 755 server. For fast turnaround needs, such as testing a quick fix of a bug, block-level constraints obtained from hierarchical characterization and the write script of the previous synthesis run can be used to compile at the block level. To get even better area, the entire design has been compiled with the hierarchy flattened so that interblock optimization can be performed during synthesis.

Verification. All the changes mentioned earlier have been simulated extensively to make sure that the desired functionality is achieved without having broken some other part of the design. Once the functionality is determined, then the HDL is synthesized to obtain a netlist from which both static timing and dynamic timing analysis are done. The majority of the emphasis in timing verification centers on static timing analysis. Synopsys' timing analyzer is used to generate timing reports. False paths and multicycle paths have been carefully reviewed to make sure that there is no escaped path in the report. Both maximum and minimum paths are reported to expose possible setup and hold violations.

The vectors run on the design include benchmark, diagnostic, and RIS vectors from Motorola. Motorola has developed a sophisticated random instruction sequence (RIS) generator that can be tuned to generate instructions in an area of interest along with random interrupts and exceptions to perturb the processor. In future Coldfire cores, the ability to generate RIS vectors will be incorporated into the verification process. This time, Motorola has generated all the RIS vectors and sent them to ICBD. Verification using a more formal method of binary decision diagram comparison has also been pursued using Motorola's in-house tool. This step will not be available for every core since most processor vendors do not support this methodology.

Simulation of the netlist had some hurdles. One is the inability of the netlist to reset properly. This problem has its roots in the way the reset logic was done in the HDL. Instead of using an explicit reset inference on all flip-flops, the reset logic became part of the input logic. Depending on where the reset was structured in the logic, it might or might not cause a particular flip-flop to reset correctly. In fact, this problem is more general. Every time a reset-like signal is used, unknown states (Xs) are not guaranteed to be suppressed. Unknown states are periodically introduced into the design by captured vectors that use uninitialized memory for operations. For example, an uninitialized stack memory may be used to fill a cache line and pushed back to memory. Granted, this is a mere simulation issue. However, it makes verification harder because only after these issues are fixed can other real problems be visible, because problems that would have been masked can then be caught.

Motorola will restructure their HDL to avoid this problem in the future. However, for the Coldfire test chip, several steps have been taken to remedy the problem. Every flip-flop and latch in the design is reset using a force-and-release pair upon startup. When unknown states are introduced into the system, the pattern is intercepted and given a random value instead. Since unknown states are essentially conditions in which the state of one or more bits is unknown, randomizing these patterns effectively gives an unknown pattern without the simulation nightmare.

Test Strategy. To make the test work with STF2 as mentioned earlier, the core is full-scan except the register file, the instruction buffer, and the latches in front of the cache RAM. The latches can also be tested using the latest methods from ATG and thus offer virtually no degradation of the test coverage. The cache RAM has BIST circuitry testing all eight RAMs in parallel. The BIST mode is encoded in the test mode pins available on the Coldfire 5202. A limited number of functional vectors run in verification are also ported to the testers.

Technology Independence. The entire core is technology independent. The only technology dependent portion of the Coldfire test chip is the pads. Since the prototypes are targeted to be used in Motorola's 5V evaluation boards, the 3V and 5V pads in the HP CMOS14 library are used. Since only I/O pads exist, input and output only pads are made by tying off the appropriate enable signals. The pads are instantiated only for the test chip. Synthesizable HDL versions of the pads do exist and can be synthesized to buffers when the megacell becomes available.

Results

The Coldfire test chip is the first trial of the proposed methodology. The performance target of 50 MHz has been met with no custom cells or modules. Both small die size and high test coverage were achieved by this chip. Higher row utilization is only limited by extreme congestion spots like the barrel shifter, register file, and pipeline control block. An even smaller version is possible with a few modifications in key areas. In addition, future versions are not expected to have the overhead of the invalidate registers implemented as flip-flops. The gates may also be resized to meet the actual target frequency.

The desired changes have been successfully implemented. Motorola's custom cache was turned into synthesizable control and generated WEST SRAM. The JTAG has been removed with minimum changes to the original HDL. BIST and test circuitry have been added. All of these changes have been verified at the functional and netlist levels. Being able to make changes at these levels and maintain high confidence in the design is an invaluable advantage with this approach that would not have been possible with artwork porting.

Data management that is needed to maintain the coherency of the design is an important aspect of the project that cannot be overlooked. Problems in this area occurred fairly early in the project. Scripts were written to make use of lineup files, that is, lists of designs with specific revision numbers that go together for a particular simulation or synthesis run. Changes that are not yet released are made in private directories that can be part of a private lineup file. The massive verification effort requires jobs to be run at every available time, using every available open Verilog license. Scripts have also been written to use HP Task Broker to get maximum efficiency of the available resources.

Conclusion

Porting processor cores using the new ICBD methodology of standard cell synthesis has been shown to be a viable alternative to the traditional artwork port. HDL porting has the advantages of testability, technology independence, customizability, efficient area use, system simulation capability, and presilicon verification. It is also a straightforward methodology to support since virtually all components of it are already in use in the HP Standard Tool Flow 2.

The approach has its disadvantages. It cannot be applied indiscriminately on any processor core. Many cores designed today still do not have synthesizable HDL. The synthesizability of the core may also run the gamut from being very easy to extremely difficult, depending on a host of issues such as clocking strategy, coding style, and architecture complexity. The need for customization puts even higher expectations on the quality of the HDL. Trying to change the functionality of a design written with raw Boolean equations and flip-flop instantiations is almost as daunting as editing a netlist. Therefore, the selection of a microprocessor vendor may depend on the vendor's design methodology. For cores that do not have synthesizable HDL, artwork porting may still be the only option.

HDL porting will become increasingly feasible with better synthesis tools and denser and faster technology. The advances in these two areas have now reached a threshold at which implementation of entire microprocessor cores with standard cells compiled using HDL synthesis is practicable. As more processors are designed using HDL and synthesis, this methodology will become more general. As the speed of the technology increases, the level of processor performance achievable using this methodology also increases. Silicon compilation is slowly becoming a reality. IC porting in the future should reach a level similar to porting software today, as designs are targeted to different technologies with a few changes in the synthesis and constraint scripts.

Acknowledgments

The author would like to thank the entire Coldfire team: Mark Reed, Paul Chenard, Tom Thatcher, and especially Jay McDougal for his work on synthesizable core methodology and synthesis techniques. Thanks also go to Tim Brown and Neal Jaarsma for looking into the superintegration and test aspects of the core. The team appreciates the support from the Motorola Coldfire team, especially Jeff Freeman and his timely responses and advice.

General-Purpose 3V CMOS Operational Amplifier with a New Constant-Transconductance Input Stage

Design trade-offs for a low-voltage two-stage amplifier in the HP CMOS14 process are presented and some of the issues of low-voltage analog design are discussed. The design of a new constant-transconductance input stage that has a rail-to-rail common-mode input range is described, along with the rail-to-rail class-AB output stage. The performance specifications and area of this amplifier are compared with a similar design in a previous process, CMOS34.

by Derek L. Knee and Charles E. Moore

Experience gained over the last few years within the design centers of the HP Integrated Circuit Business Division (ICBD) has shown that a general-purpose operational amplifier is a fundamental building block for many mixed-signal integrated circuits. These general-purpose operational amplifiers are typically used in support functions and not in the high-frequency differential signal paths.

With the recent process release of AMOS14TB, the analog version of the HP CMOS14TB IC process, the logical step was to design a general-purpose operational amplifier for use with mixed analog/digital chips using AMOS14TB. However, from an analog standpoint, the technology change from CMOS34, the most recent process in which analog circuits had been implemented, to CMOS14 was quite severe because of the power supply reduction from 5V to 3.3V. Because of the lower supply voltage specification, new circuit design techniques needed to be developed and the general-purpose operational amplifier was chosen as one of the test vehicles to achieve this goal. The amplifier was also integrated onto an AMOS14 test chip.

Design Objectives

Because of the usefulness of the previous CMOS34 general-purpose operational amplifier, the electrical specifications for the AMOS14 version were derived from the CMOS34 amplifier. The power supply range was altered because of the technology change. Other parameters such as input offset voltage, input referred noise, and size were to be minimized, while open-loop voltage gain, gain margin, phase margin, and power supply rejection ratio were to be maximized. A list of the design objectives is shown in Table I.

Configuration

Based on the design objectives shown in Table I and the experience of the authors in the design of previous general-purpose operational amplifiers, a two-stage configuration

with a class-AB output stage was chosen. This configuration is capable of satisfying the power and load requirements. An added constraint for the AMOS14 version (based on limitations of the previous versions) is the specification for constant small-signal bandwidth, independent of the common-mode input range, CMIR. The amplifier has a differential input, and the common-mode input voltage is the average value of the two input voltages. CMIR is the range over which the common-mode input voltage is expected to vary. A small-signal bandwidth that is independent of CMIR implies that the input differential stage has a constant small-signal input transconductance, g_m, over the full CMIR, even if the CMIR is as large as the difference between the power supply rails.

Leveraging the new AMOS14 circuit design from the existing CMOS34 design was difficult because the power supply voltage range is reduced while the PMOS and NMOS transistor thresholds, V_{tp} and V_{tn} respectively, are essentially unchanged. The power supply range for AMOS14 is reduced by 33% from that of CMOS34. This power supply reduction is fairly significant for analog designs in which devices are connected in series. The outcome was that new low-voltage design techniques had to be employed to implement the equivalent operational amplifier in AMOS14 technology.

Table I
Design Objectives for AMOS14 Operational Amplifier

Parameter	Target Value
Single-supply operation	$2.7V \le AV_{DD} \le 3.6V$
Temperature range T_{op}	$0°C \le T_{op} \le 110°C$
Outputs	Single-ended
Low quiescent power consumption	$I_{DD} \le 1$ mA
Small-signal bandwidth f_o (unity gain)	$1\ MHz \le f_o \le 5\ MHz$
Small-signal bandwidth	Independent of CMIR
Slew rate SR	$1V/\mu s \le SR \le 5V/\mu s$
Output voltage range	$AV_{SS} + 0.2V \le V_{out} \le AV_{DD} - 0.2V$
Common-mode input range CMIR	$AV_{SS} \le CMIR \le AV_{DD}$
Load capacitance range	$C_{load} \le 100$ pF
Load resistance range	$R_{load} \ge 300\Omega$

Constant-Transconductance Differential Input Stage

To obtain a differential input stage that operates over a rail-to-rail input voltage range requires an NMOS and PMOS pair driven in parallel. Because of complementary biasing requirements, special circuit design precautions need to be taken to ensure that the overall g_m, or the sum of the individual transistor g_ms, remains constant over the CMIR. Without this added circuitry, the frequency compensation could not be optimized over the CMIR.

The requirements for the constant-g_m input stage are:

- A simple circuit with a minimum number of components
- Low-voltage operation
- Input devices operating in the square-law region where g_m is highest.
- Constant-g_m control circuitry operating in a closed-loop mode with the input differential devices to exhibit smooth transition regions over the CMIR
- Constant-g_m control circuitry that does not use reference voltage trip levels to control the differential bias currents, thus avoiding coupling supply noise into the input stage.

An extensive search of the literature[1-14] could not locate a circuit that met this list of requirements. Therefore, a new constant-g_m input stage was needed.

If I_{tn} and I_{tp} are the tail currents of the NMOS and PMOS differential pairs respectively, then the following relationship is required for any common-mode input voltage:

$$\sqrt{2K_nI_{tn}} + \sqrt{2K_pI_{tp}} = g_m = \text{Constant,} \qquad (1)$$

where

$$K_n = \mu_n C_{ox}\frac{W_n}{2L_n} \qquad (2a)$$

and

$$K_p = \mu_p C_{ox}\frac{W_p}{2L_p}. \qquad (2b)$$

In equations 2a and 2b, μ is the carrier mobility under the channel, C_{ox} is the transistor gate capacitance per unit area, W is the transistor gate width, and L is the transistor gate length.

If the PMOS and NMOS transistors are sized so that $K_n = K_p$ then equation 1 can be rewritten as:

$$\sqrt{I_{tn}} + \sqrt{I_{tp}} = \text{Constant.} \qquad (3)$$

A new feedback control loop circuit was designed that controls the bias currents in the NMOS and PMOS differential pair transistors so that equation 3 holds for all common-mode input voltages. This new circuit is shown in Fig. 1. It uses what the authors refer to as the 4I/I principle.

In Fig. 1, transistors N0A, N0B, N1A, and N1B form the NMOS input section. Devices P0A, P0B, P1A, and P1B form the PMOS input section. These two sections together form the input stage to an operational amplifier. The output currents from these sections—I_{OPP}, I_{OPN}, I_{ONP}, and I_{ONN}—are summed in the first gain stage, described below. It is the overall g_m of these NMOS and PMOS input devices that is held constant over the CMIR.

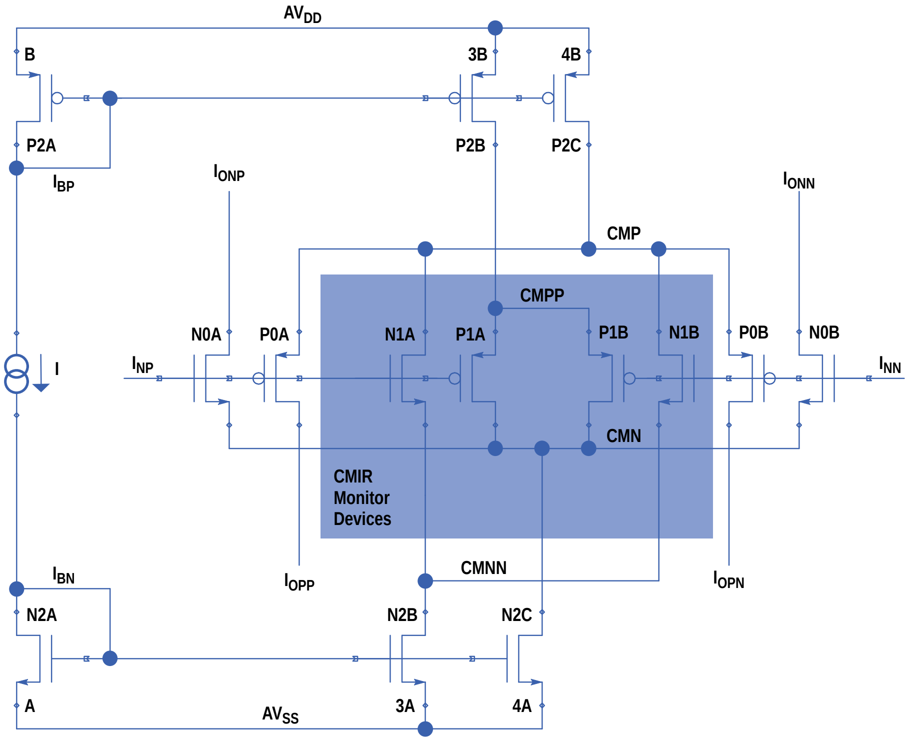


Fig. 1. *Constant-transconductance amplifier input stage.*

The current mirror N2C biases the NMOS input pair and the current mirror P2C biases the PMOS input pair. The NMOS CMIR monitor devices, N1A and N1B, are biased by N2A and N2B at a current of 3I. The PMOS CMIR monitor devices, P1A and P1B, are also biased by P2A and P2B at a current of 3I.

For midsupply common-mode input range, both the NMOS input section and the PMOS input section are biased on. The PMOS CMIR input monitor devices, P1A and P1B, source a current of 3I to the node CMN. This 3I source current is added algebraically to the 4I current sink of N2C, resulting in the NMOS differential pair, N0A and N0B, being biased at a current I. Similarly, the NMOS CMIR input monitor devices, N1A and N1B, sink a current of 3I from the node CMP. This 3I current is added algebraically to the 4I source current of P2C, resulting in the PMOS differential pair, P0A and P0B, being biased at a current I. Therefore the NMOS and PMOS input sections are both biased at I for the midsupply common-mode input.

For common-mode inputs near AV_{DD}, the NMOS input section is biased correctly, but the PMOS input section is off. The current source devices P2B and P2C are also off and the PMOS CMIR monitor devices, P1A and P1B, supply no current. Since no current is added to the current source N2C, the NMOS differential pair, N0A and N0B, is now biased at a current of 4I. A similar argument holds for the PMOS devices when the common-mode input is close to AV_{SS}, and the PMOS transistors are biased at 4I.

The differential input sections will be biased in one of the following modes:

1. The NMOS devices biased at 4I and the PMOS section with no bias current for low CMIR:

$$\sqrt{4I} + \sqrt{0I} = \text{Constant.} \qquad (4a)$$

2. The PMOS devices biased at 4I and the NMOS section with no bias current for high CMIR:

$$\sqrt{0I} + \sqrt{4I} = \text{Constant.} \qquad (4b)$$

3. Both sections biased at I when the CMIR is such that both N2B and P2B are biased correctly:

$$\sqrt{1I} + \sqrt{1I} = \text{Constant.} \qquad (4c)$$

The closed-loop CMIR monitor circuitry smoothly controls the transition between these three modes of operation. This is demonstrated in Fig. 2. The x-axis of Fig. 2 represents the CMIR from AV_{SS} to AV_{DD} (rail to rail). The upper curve shows the overall g_m or the sum of the NMOS and PMOS input stage g_ms, while the lower curves show the individual g_ms of the input sections as a function of CMIR. The overall g_m has a total variation of only 5%. This number includes the second-order effects of subthreshold operation and output conductance.

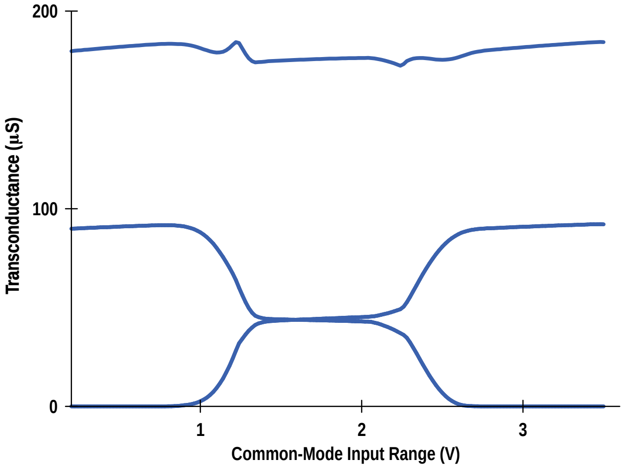


Fig. 2. *The x-axis represents the common-mode input range (CMIR) of the circuit of Fig. 1 from AV_{SS} to AV_{DD} (rail to rail). The upper curve shows the overall g_m. The lower curves show the individual g_ms of the input sections as a function of the CMIR.*

Fig. 3 shows the simulated variation of the intrinsic input offset voltage, V_{os}, as a function of the CMIR. This curve shows one of the limitations of a complex input differential pair input structure: the input offset voltage varies as each of the input differential pairs is activated or deactivated. During the transitions between modes, the common-mode rejection ratio, CMRR, is reduced.[1-12] Therefore, the design of Fig. 1 attempts to minimize the width of these transition regions with respect to CMIR.

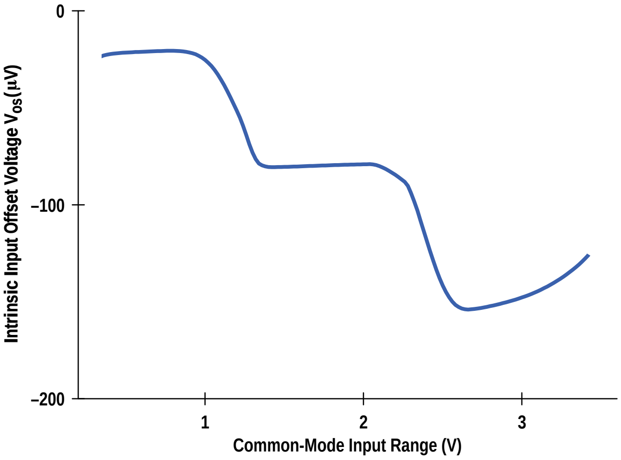


Fig. 3. *Simulated variation of the intrinsic input offset voltage, V_{os}, as a function of the CMIR.*

First Gain Stage

The first gain stage sums the four output currents from the input differential stage: I_{OPP}, I_{OPN}, I_{ONP}, and I_{ONN}. The criteria for selecting the best gain stage were:

- The stage should use wide-swing cascode current sources.
- It should interface easily with the following class-AB output stage or second gain stage.
- It should not add any additional noise or offset to the input stage.

The gain and current summing stage selected is shown in Fig. 4.[14] This stage reduces the transistor count considerably because of its compact integration with the class-AB output stage (see next section).

Second Gain Stage

The criteria used in the selection of the class-AB output stage implementation were:

- Simple and high-speed design
- No complex active or amplifier feedback paths in the AB control circuitry
- Low-V_{DD} operation

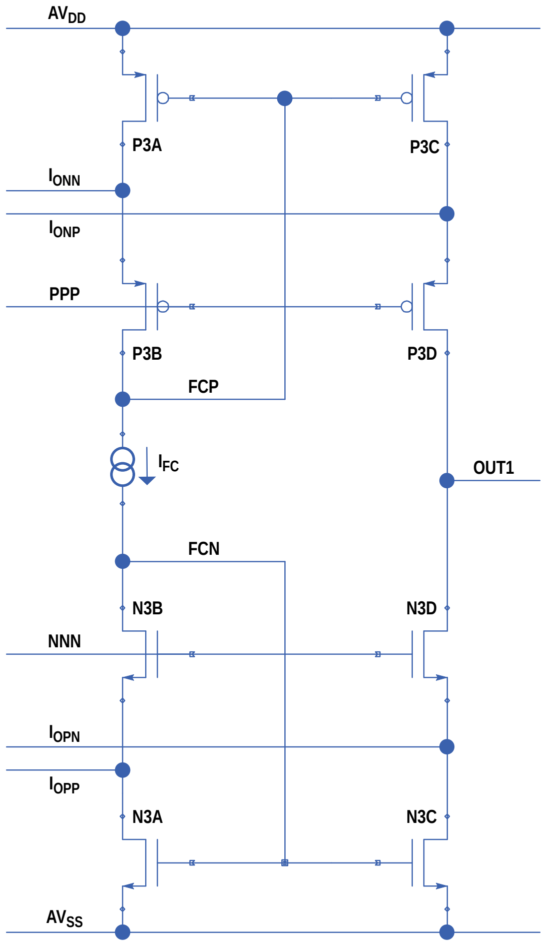


Fig. 4. *Schematic diagram of the first gain stage, which sums the four output currents from the input stage.*

- Good power supply rejection ratio
- No direct dependence on supply voltage for bias current setup
- No noise or offset to be added to the first stage of the amplifier.

The output stage chosen is shown in Fig. 5. The circuit shown in Fig. 5a is a simplified version of the output stage. The schematic in Fig. 5b shows the implementation of the AB output stage integrated together with the first gain stage. This output stage was first developed for 5V operation[15] and later modified for an all-digital process.[16]

The output stage uses common-source output devices for low-voltage operation. The theoretical minimum supply voltage is twice the MOS threshold voltage plus a saturation voltage. The complementary output devices PDR and NDR are driven by complementary common-gate level shifters, PAB and NAB. The first-stage input signals are fed into the output stage at nodes PDRV and NDRV. During quiescent operation, PAB and NAB are biased in the conducting state. The potentials at PDRV and NDRV are established to minimize the quiescent current through the large output driver devices, PDR and NDR. This biasing arrangement is established through two translinear loops. The loop that biases PDR consists of P5A, P5B, PAB, and PDR. Similarly, NDR is biased by the loop consisting of N5A, N5B, NAB, and NDR. For a short tutorial on translinear theory see **reference 17**.

During a negative slew at the output, the gate voltage of NDR is pulled high. Since the bias voltage ABN is fixed, the device NAB will shut off. The device PAB will be then conducting the full bias current, I_{FC}, which will result in an increase in the gate-to-source voltage of PAB and consequently a reduction in the gate-to-source voltage of PDR. A similar operation occurs during positive sourcing when the bias voltage for NDR is reduced.[15]

The integration of the class-AB stage and the first gain stage has two major advantages. The first advantage is the floating current source, I_{FC}, which is set up through two additional translinear loops: N5A, N5B, NFC, N3A and P5A, P5B, PFC, P3A. Because of the floating nature of the bias devices NFC and PFC and NAB and PAB, this current source contributes much less to the noise and offset of the amplifier. Secondly, the variation of the output quiescent current is reduced because the floating current source of PFC and NFC tracks the AB current source of NAB and PAB.

Final Circuit and Results

The complete schematic for the AMOS14 operational amplifier is shown in Fig. 6. This figure shows in detail the cascode current source implementation.

Fig. 7 shows the open-loop small-signal frequency response and phase characteristics of the amplifier driving four different load combinations. These are 10 MΩ‖1 pF, 10 MΩ‖100 pF, 300Ω‖1 pF, and 100Ω‖100 pF. Fig. 8 shows the small-signal frequency response and phase characteristics of the amplifier for different CMIR values ranging from AV_{SS} to AV_{DD}. Note that the unity-gain frequency f_o is essentially independent of CMIR.

The small-signal step response is shown in Fig. 9 for the same load combinations as Fig. 7. The large-signal step response, indicative of the amplifier's slew rate, is shown in Fig. 10. The artwork layout for the operational amplifier is shown in Fig. 11.

Table II illustrates the overall similarities of the AMOS14 operational amplifier to the CMOS34 version. In summary, the AMOS14 design achieved a 2× improvement in bandwidth, a 2.5× increase in class-AB output drive, and a 3× improvement in slew rate in a third of the area while at the same time including the additional constant-g_m circuitry.

Table II
Amplifier Process Comparison

Parameter	AMOS14	CMOS34
Supply voltage	2.7 to 3.6V	4.5 to 5.5V
Supply current	625 µA	750 µA
Common-mode input range CMIR	AV_{SS} to AV_{DD}	AV_{SS} to AV_{DD}
Constant-g_m input stage	Yes	No
Input stage g_m variation, $AV_{SS} \le CMIR \le V_{DD}$	$< \pm 5\%$	50%
Intrinsic input offset voltage, $CMIR = AV_{DD}/2$	-80 µV	-120 µV
Resistive load	300Ω min	300Ω min
Capacitive load	100 pF max	100 pF max
Maximum output drive current I_{max}	±5 mA	±2 mA
Maximum output swing at I_{max}	$AV_{DD} - 0.25$	$AV_{DD} - 0.3$
Minimum output swing at I_{max}	$AV_{SS} + 0.25$	$AV_{SS} + 0.3$
Open-loop gain (no load)	>100 dB	>100 dB
Slew rate	6V/µs	0.5V/µs*
Unity-gain bandwidth, f_o	4 MHz	0.5MHz*
Phase margin **	55 degrees	49 degrees
Gain margin **	-19dB	-8 dB
PSRR+, $AV_{SS} \le CMIR \le AV_{DD}$	>70dB	>80 dB
PSRR−, $AV_{SS} \le CMIR \le AV_{DD}$	>70db	>80 dB
Cell size	251 µm × 141µm	460 µm × 210 µm

* Depends on CMIR.
** Absolute worst-case conditions for I_{bias}, AV_{DD}, R, C, models. See Fig. 7.
CMIR = common-mode input range.
PSRR = power supply rejection ratio.

Acknowledgments

The authors thank their project manager, Rajeev Badyal, for his encouragement during the development of this amplifier.

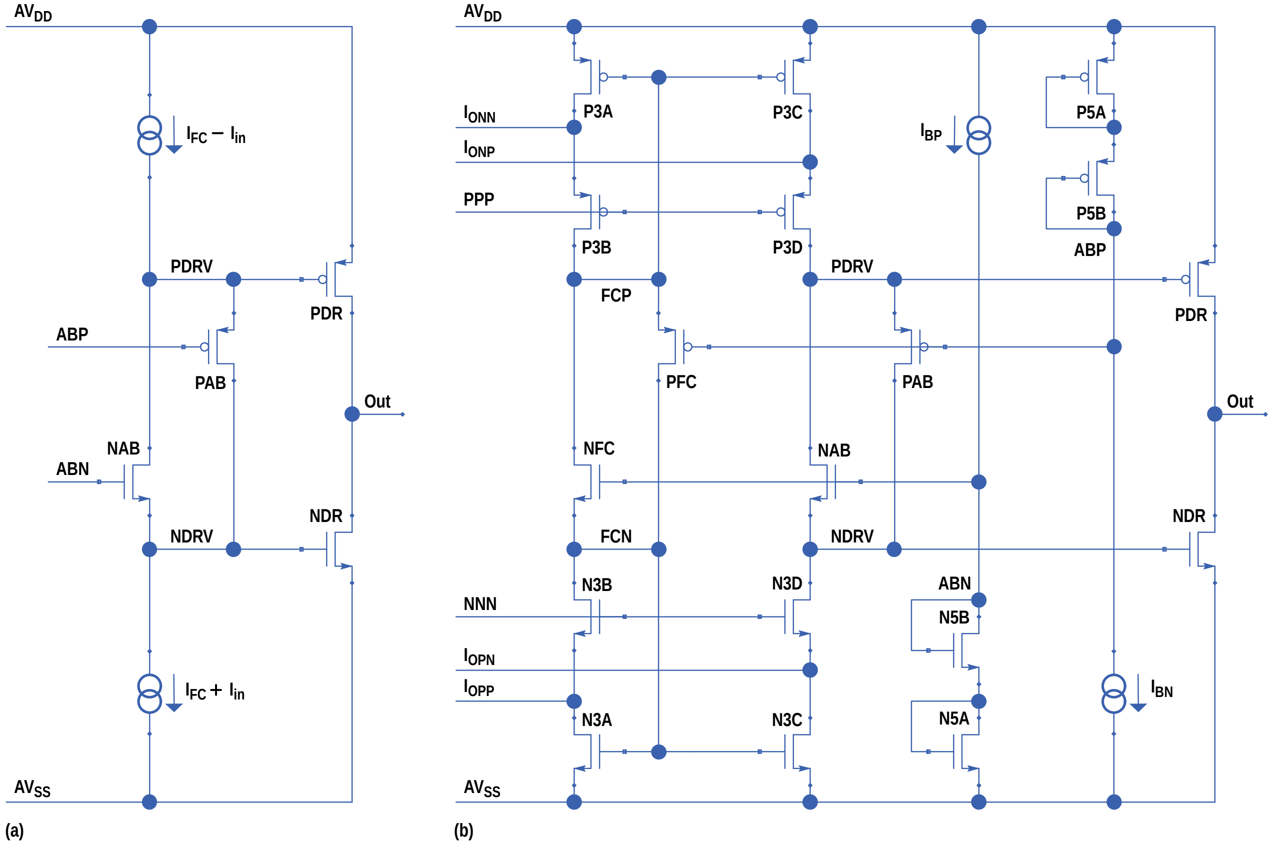


Fig. 5. *(a) Simplified schematic diagram of the output stage. (b) Integration of the output stage with the first gain stage.*

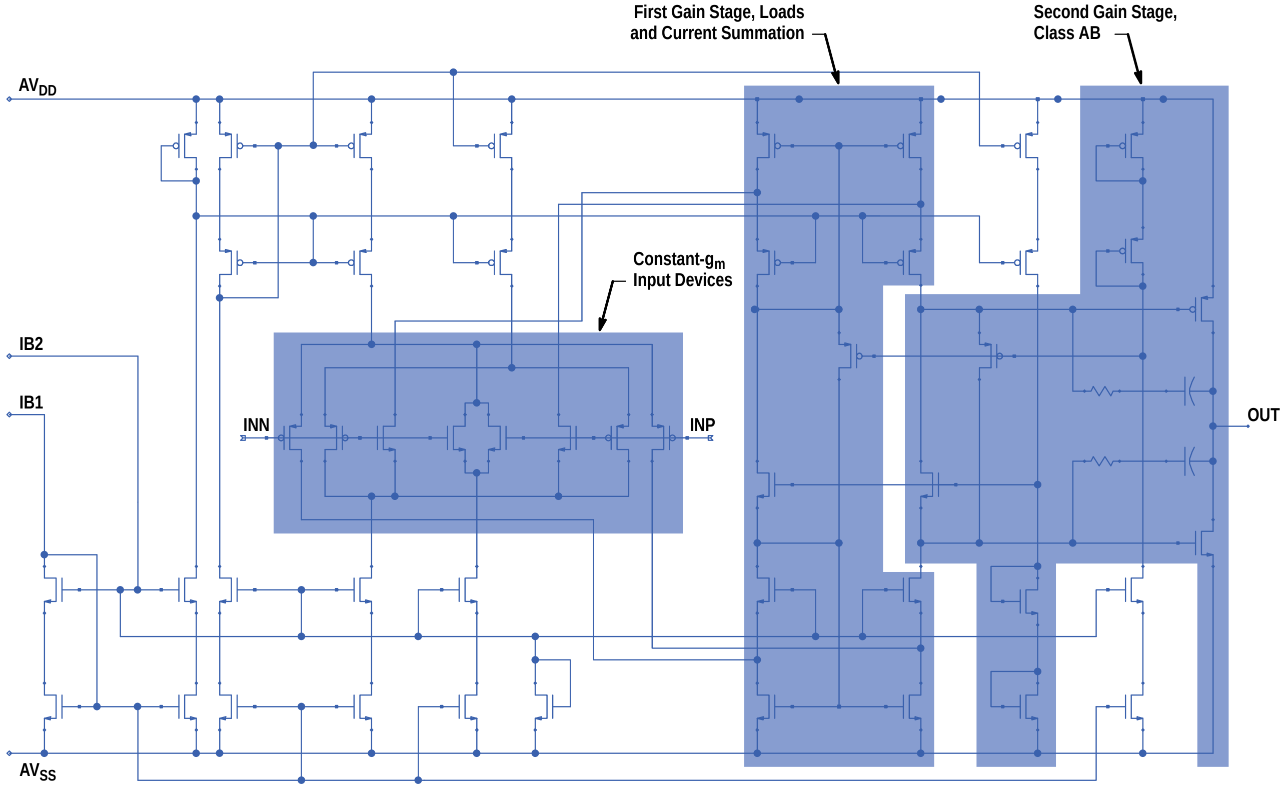


Fig. 6. *Complete schematic diagram of the AMOS14 operational amplifier. The unshaded area contains bias support circuitry.*

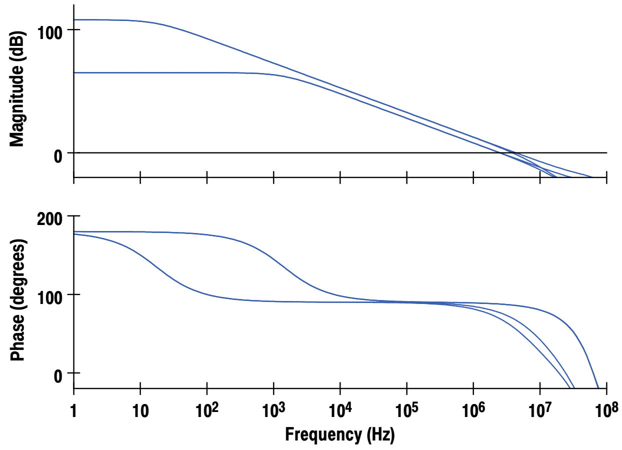


Fig. 7. *(top) Open-loop small-signal frequency response for different load conditions. (bottom) Phase response.*

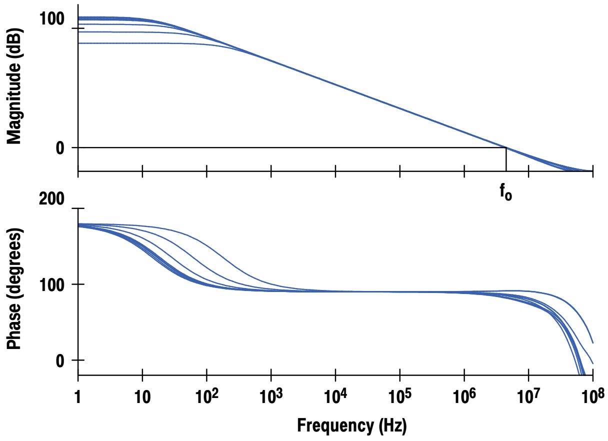


Fig. 8. *(top) Small-signal frequency response for different CMIR values. (bottom) Phase response.*

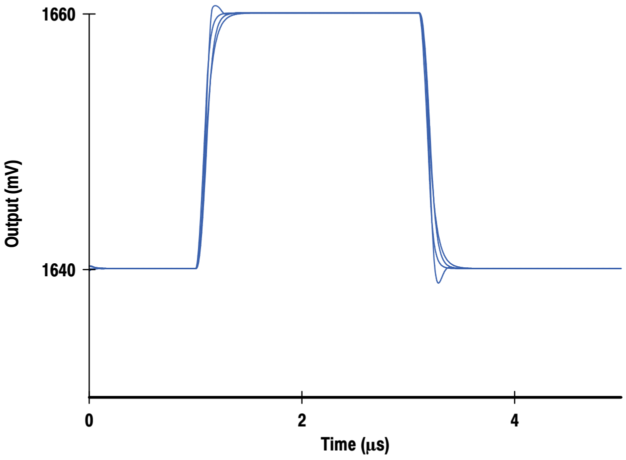


Fig. 9. *Small-signal step response for different load conditions.*

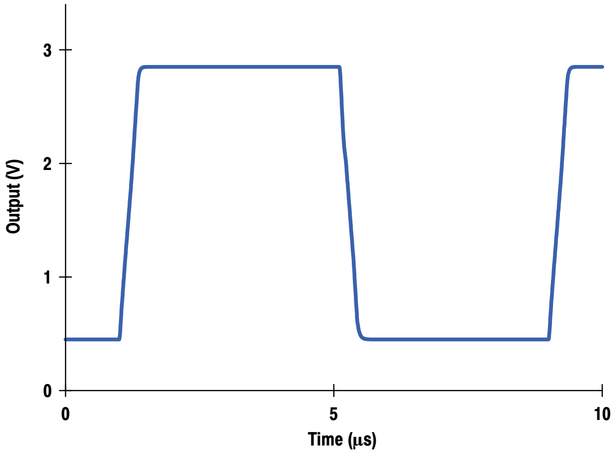


Fig. 10. *Large-signal step response, indicative of slew rate.*

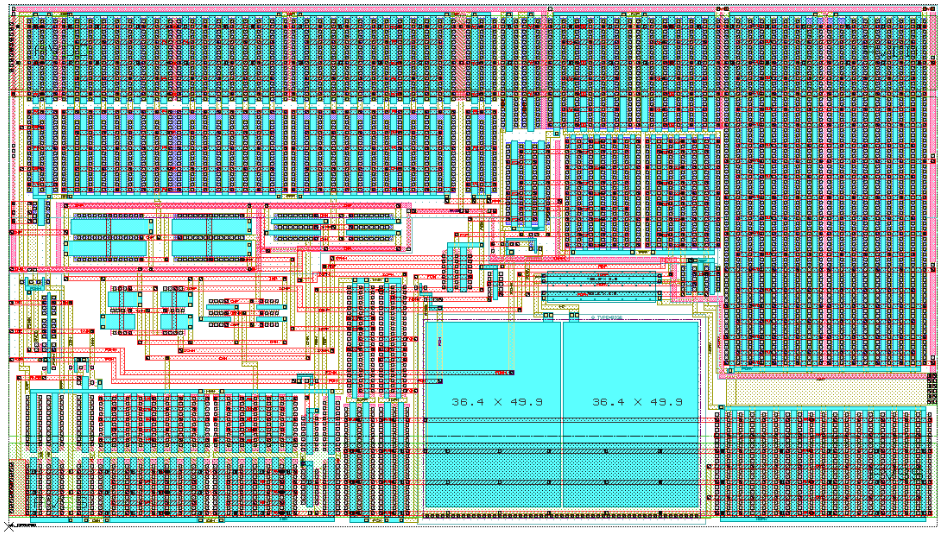


Fig. 11. *Operational amplifier layout.*

References

1. J.H. Huijsing and D. Linebarger, “Low-voltage operational amplifier with rail-to-rail input and output ranges,” *IEEE Journal of Solid-State Circuits*, Vol. SC-20, December 1985, pp. 1144-1150.
2. M.J. Fonderie and J.H. Huijsing, *Design of Low-Voltage Bipolar Operational Amplifiers*, Kluwer Academic Publishers, 1993.
3. J. Fonderie, M.M. Maris, and E.J. Schnitger, “1-V operational amplifier with rail-to-rail input and output ranges,” *IEEE Journal of Solid-State Circuits*, Vol. SC-24, December 1989, pp. 1551-1559.
4. J.A. Fisher and R. Koch, “A highly linear CMOS buffer amplifier,” *IEEE Journal of Solid-State Circuits*, Vol. SC-22, June 1987, pp. 330-334.
5. M. Steyaert and W. Sansen, “A high-dynamic-range CMOS op amp with low-distortion output structure,” *IEEE Journal of Solid-State Circuits*, Vol. SC-22, December 1987, pp. 1204-1207.
6. J.N. Babanezhad, “A rail-to-rail CMOS op amp,” *IEEE Journal of Solid-State Circuits*, Vol. 23, December 1988, pp. 1414-1417.
7. T.S. Fiez, H.C. Yang, J.J. Yand, C. Yu, and D.J. Allstot, “A family of high-swing CMOS operational amplifiers,” *IEEE Journal of Solid-State Circuits*, Vol. 24, December 1989, pp. 1683-1687.
8. M.D. Pardoen and M.G. Degrauwe, “A rail-to-rail input/output CMOS power amplifier,” *IEEE Journal of Solid-State Circuits*, Vol. 25, April 1990, pp. 501-504.
9. R. Hogervorst, R.J. Wiegerink, P.A. de Jong, J. Fonderie, R.F. Wassenaar, and J.H. Huijsing, “CMOS low-voltage operational amplifiers with constant-g_m rail-to-rail input stage,” *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1992, pp. 2876-2879.
10. J.H. Botma, R.F. Wassenaar, and R.J. Wiegerink, “A low-voltage CMOS op amp with rail-to-rail constant-g_m input stage and a class-AB rail-to-rail output stage,” *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1993, pp. 1314-1317.
11. R. Hogervorst, R.J. Wiegerink, P.A.D. Jong, J. Fonderie, R.F. Wassenaar, and J.H. Huijsing, “CMOS low-voltage operational amplifier with constant-g_m rail-to-rail input stage,” *Analog Integrated Circuits and Signal Processing*, Vol. 5, Kluwer Academic Publishers, 1994, pp. 135-146.
12. J. Huijsing, R. Hogervorst, J. Fonderie, B. van der Dool, K.-J. de Langen, and G. Groenevold, “Low-voltage signal processing,” *Analog VLSI: Signal and Information Processing* (Ismail and Fiez, editors), Chapter 4, McGraw-Hill, 1994.
13. S. Sakurai and M. Ismail, *Low-Voltage CMOS Operational Amplifiers: Theory, Design and Implementation*, Kluwer Academic Publishers, 1995.
14. R. Hogervorst, J.P. Tero, R.G. Eschauzier, and J.H. Huijsing, “A compact power-efficient 3V CMOS rail-to-rail input/output operational amplifier for VLSI cell libraries,” *IEEE Journal of Solid-State Circuits*, Vol. SC-29, December 1994, pp. 1505-1513.
15. D. M. Montecelli, “A quad CMOS single-supply op amp with rail-to-rail output swing,” *IEEE Journal of Solid-State Circuits*, Vol. SC-21, December 1986, pp. 1026-1034.
16. W.-C.S. Wu, W.J. Helms, J.A. Kuhn, and B.E. Byrkett, “Digital-compatible high-performance operational amplifier with rail-to-rail input and output ranges,” *IEEE Journal of Solid-State Circuits*, Vol. SC-29, January 1994, pp. 63-66.
17. J.H. Botma, R.J. Wiegerink, A.J. Gierkink, and R.F. Wassenaar, “Rail-to-Rail Constant-g_m Input Stage and Class-AB Output Stage for Low-Voltage CMOS Op Amps,” *Analog Integrated Circuits and Signal Processing*, Vol. 6, Kluwer Academic Publishers, 1994, pp. 121-133.

Improving Heat Transfer from a Flip-Chip Package

The lid of an ASIC package can significantly increase the temperature of the die by impeding heat transfer. In flip-chip packages the backside of a die can be exposed by eliminating the lid, thus allowing a heat sink to be attached directly. Numerical finite difference methods and experimentation were used to investigate the differences between lidded and lidless flip-chip designs. The results demonstrate that a lidless package is a superior design because of the increased thermal conductivity between the die and the heat sink.

by Cullen E. Bash and Richard L. Blanco

The cooling of electronic components has traditionally been considered as two separate problems: optimizing the internal thermal path within the package, and cooling the packaged component by optimizing the external thermal path. While this method has the advantage of being partitionable and therefore solvable independently by separate organizations or companies, it fails to engineer the thermally optimum solution. This is especially critical for high-power dice, which typically require custom heat sinks.

The electronics industry is moving in the direction of lidless flip-chip packages, which create new possibilities for cooling the dice. Processor chips from other major electronics suppliers are currently available in lidless packages because of their thermal and cost advantages.[1]

As an experiment to improve the design of a high-power processor package, the HP PA 8000 processor, a proposed design of a lidless package was compared to the traditional lidded package currently in use. An example of a lidless package using an air cooled heat sink has been discussed in an earlier paper.[2] In the present investigation, the proposed design uses the evaporator of a heat pipe assembly to contact the die, thus replacing the lid. This concept has the additional benefit of reducing the cost of the package by eliminating the relatively expensive lid.

The investigation began by constructing finite difference models of the lidded and lidless packages. The purpose of the models was not to correlate with measured results but to aid in understanding the magnitude of the relative improvements of the lidless design. After reviewing the results, laboratory measurements were made of the two designs and the relative improvements in thermal performance were recorded.

Two different methods were chosen to cool the packages. The heat pipe employed in the current HP PA 8000 design was a natural choice because of its practicality. Additionally, because of concerns about thermal gradients in the aluminum heat pipe evaporator and the difficulty of matching these to the boundary conditions in the finite difference model, a very efficient but impractical liquid cooled heat sink was chosen. The liquid cooled heat sink is highly efficient and behaves like an isothermal block, which is easily modeled.

For consistency throughout this paper, the term *aluminum evaporator heat sink* refers to the aluminum evaporator on the heat pipe assembly that directly sinks heat from the package. Likewise, the term *copper block heat sink* refers to the copper block on the liquid cooled heat sink that acts in the same capacity.

Package Construction

The lidded and lidless package designs are shown in Fig. 1 for the aluminum evaporator heat sink. Both packages are constructed identically between the printed circuit board and the die. Mounted on an FR-4 printed circuit board is a plastic socket containing 1089 contacts made from 0.025-mm gold plated molybdenum wire (see Fig. 2). A ceramic land grid array package rests on the socket, making electrical contact between the die and the board. The processor die is attached using flip-chip technology,[3] resulting in about 2500 solder bump connections encapsulated by an underfill material between the ceramic substrate and the silicon die. Fig. 3 shows the lidless package, plastic socket, and printed circuit board assembly. The aluminum carrier shown in the picture is used to support the assembly. The heat sink has been left off so that the assembly can be seen more clearly.

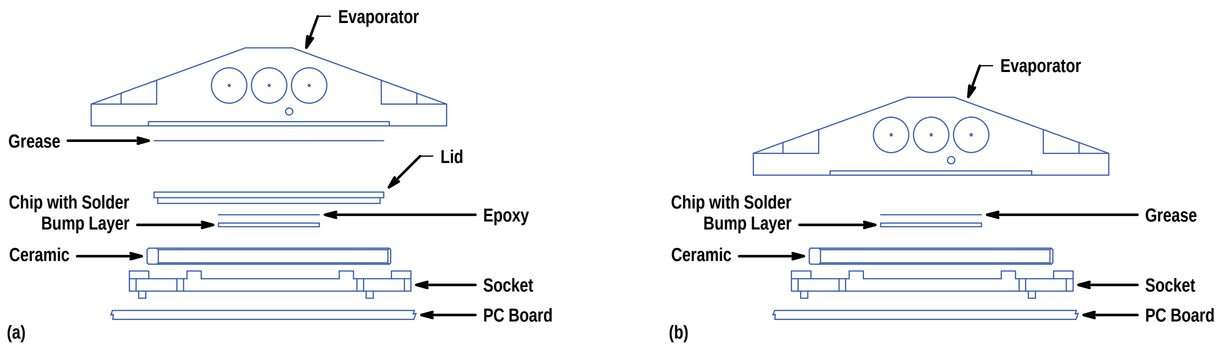


Fig. 1. *Package designs. (a) Lidded. (b) Lidless.*

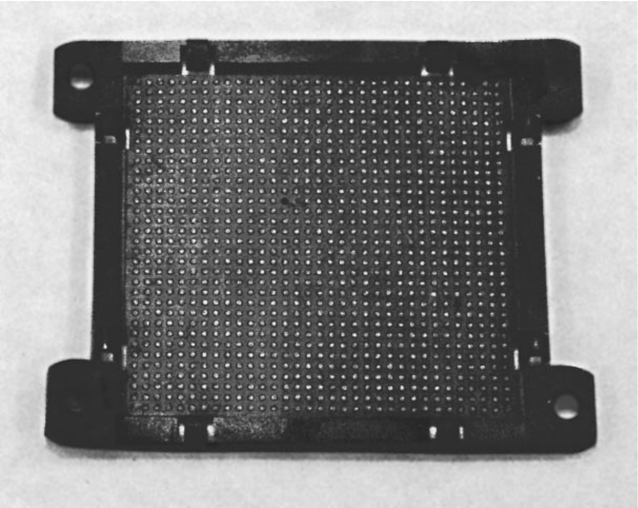


Fig. 2. *Plastic socket with 1089 gold plated molybdenum wire contacts.*

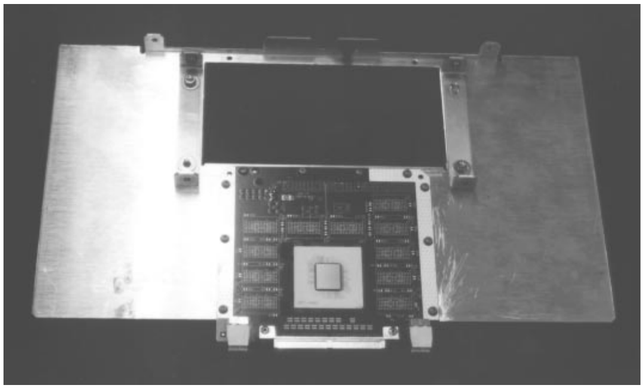


Fig. 3. *The experimental assembly without the heat sink, showing the printed circuit board, socket, and lidless package.*

The lidded design uses silver-filled epoxy between the die and the lid to enhance thermal performance. The lid is fabricated from a Kovar ring brazed to a sheet of tungsten copper. Fig. 4 shows the lidless and lidded packages side by side for comparison. A more detailed description of the lidded package can be found elsewhere in the literature.[4]

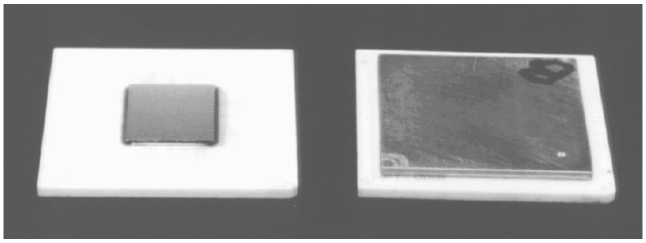


Fig. 4. *Lidless and lidded packages used in the experiment. The lidded package is on the right.*

The lidless design uses Dow Corning 340 thermal grease as the thermal interface above the die. This is a conservative choice considering that there are thermal greases available that have thermal conductivities more than three times that of Dow Corning 340.[5]

Measurement Technique

To compare the thermal performance of the two packages, a thermal test die with a temperature-sensitive resistor was placed into each package to allow direct measurement of the die temperature. The packages were each tested on the same socketed printed circuit board connected to an HP 75000 data acquisition system and a power supply. An HP 9000 Series 300 workstation with data acquisition software, HP VEE, displayed the die temperature as a function of time while the power supply provided the power to the die.

The two thermal test dice were calibrated in a Delta Design 9000 Series convective oven. Resistances were captured with the data acquisition system at four different temperatures ranging from 18 to 90 degrees Celsius. A least-squares fit was obtained for each package and the results were placed into HP VEE.

Four experiments were undertaken comparing each package—lidded and lidless—cooled by each of the heat sinks—the aluminum evaporator and the copper block.

Copper Block. The copper block heat sink was used to provide an isothermal surface to the package to which it was attached. This was accomplished via an efficient liquid cooled heat sink mounted to the backside of the highly conductive copper block as depicted in Fig. 5. The liquid cooled heat sink consists of a partially hollowed aluminum block through which water is cycled. The water is cooled by ambient air via a heat exchanger. Measurements showed that the surface of the copper block was kept isothermal to within 3°C, which indicated that the liquid cooled heat sink was functioning as intended.

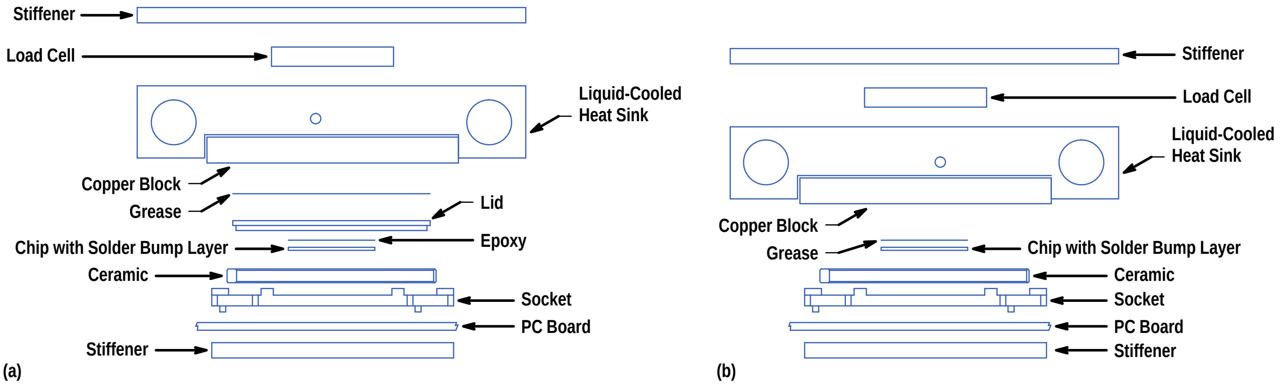


Fig. 5. *Liquid heat sink assembly. (a) Lidded package. (b) Lidless package.*

Each package was tested with the copper block heat sink by compressing it between the upper and lower stiffeners with a C-clamp. The setup was similar to Fig. 6 but with the heat pipe replaced by the liquid cooled heat sink. A load cell was employed to measure the compressive force being generated by the clamping assembly and stiffener plates were used to distribute the C-clamp load. Each assembly was compressed to 150 pounds to ensure comparable contact resistance between the two packages. Three thermocouples were placed within the copper block to record the heat sink temperature.

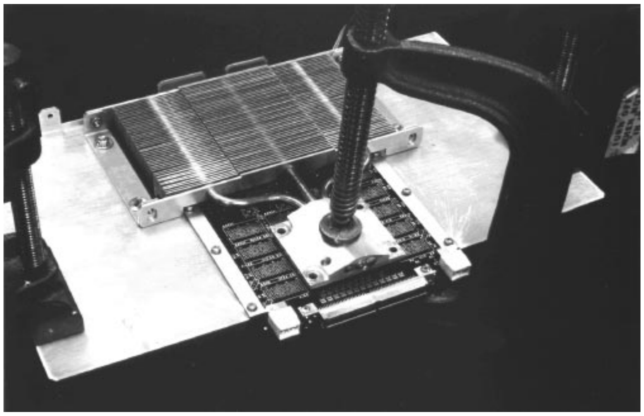


Fig. 6. *Experimental setup with heat pipe.*

Aluminum Evaporator. The aluminum evaporator is cooled by a heat pipe assembly. The assembly is constructed of three sintered copper pipes with water as the working fluid mounted planar to the evaporator, and thin aluminum fins are attached to the opposite end of the pipes. Heat from the aluminum evaporator enters the pipes, causing the water to vaporize. The steam is condensed at the other end of the pipes by air flowing over the fins. The water then returns to the evaporator via capillary action, thus completing the thermodynamic cycle. Upon measurement, it was discovered that the aluminum evaporator was indeed isothermal like the copper block, although at a higher temperature.

The aluminum evaporator was used to test the thermal performance of the packages in a manner similar to the copper block. A clamping assembly comparable to that used for the copper block was employed (the clamping assembly is shown in Fig. 6 with the heat pipe). The entire assembly was placed in a wind tunnel with a nominal velocity of 1.8 meters per second. A single thermocouple was placed near the evaporator plate/package interface to record temperature.

Data Comparison Methodology

Thermal resistance will be used throughout this paper as a means of comparing the data obtained from modeling and measurement. It is defined by equation 1 and frequently calculated using empirical data with equation 2:

$$R = L/(kA) \tag{1}$$

$$R = \Delta T/Q \tag{2}$$

where L is the thickness of the material, k is the material thermal conductivity, A is the cross-sectional area, ΔT is the measured temperature difference, and Q is the heat flow. By definition, thermal resistance is applicable for one-dimensional, steady-state heat transfer with no internal energy generation. In electronics packaging one rarely encounters one-dimensional heat transfer and there is significant internal energy generation in the silicon die. Additionally, it is rarely ever known explicitly how much heat is flowing into the heat sink relative to that being absorbed by the board. Typically, if no additional information is known it is assumed that all of the heat is dissipated into the heat sink. Nevertheless, with the restrictions on equations 1 and 2 and the unknowns involved, thermal resistance remains a useful quantity for the comparison of similar packages on similar printed circuit boards and will be used in that capacity in the interpretation of results.

Modeling Technique

A software tool employing a finite difference method was used to create models to represent the cooling of the packages under test.[6] One model was created for the lidded design and a second was created for the lidless design. With each model, either the copper block or the aluminum evaporator could be activated as the heat sink.

Two simplifications were made in modeling the packages. Components of the model that were thin layers, such as the epoxy and grease layers, were modeled as internal plates with only one-dimensional conduction, normal to the surface of the layer. Secondly, to simplify the model and reduce large grid aspect ratios and thus convergence time, geometry that was nearly coincident and thermally insignificant was spatially aligned. For example, the plastic socket housing is 0.7 mm larger than the ceramic but was modeled as the same overall size.

The FR-4/copper multilayer printed circuit board was modeled as a solid FR-4 block with a single layer of copper of thickness equivalent to the combined thicknesses of the copper layers in the board. The conductivity of the multilayer printed circuit board was calculated to be equivalent to the copper and FR-4 material in parallel, while the conductivity of the single copper layer placed within the modeled printed circuit board was made equivalent to the copper and FR-4 material in series. Only solid copper planes were included in the model since discontinuous signal planes have been determined to be inconsequential in conducting heat.[7]

To simplify the 1089 individual metallic contacts of the socket in the plastic housing, a block of equivalent conductivity to the 1089 individual 0.025-mm-diameter molybdenum wires was combined in parallel with the conductivity of the plastic housing.

Similarly, the solder bump layer with underfill was modeled as the area of 2500 solder bumps in parallel with the area of the underfill compound, with the conductivity of the internal plate appropriately weighted by the product of the thermal conductivity and the area of each material.

The copper block was modeled as an isothermal volume with a negative internal power source (i.e., a sink). The evaporator assembly, while more difficult to approximate, was modeled as an aluminum block with negative internal power sources that were of the same volume and in the same locations as the heat pipes used in the experiments. The actual cross sections of the heat pipes were modeled as squares because of the orthogonal limitations of the software tool.

The models were constructed to calculate conduction through the package to study the effects of various constructions and materials. To simplify and reduce convergence time, cooling from natural convection was not considered. This method allows good comparative results for small changes in materials but does not yield results that could be directly compared with measurements. Nevertheless, the purpose of the modeling was not to correlate numerical data with experimental data, but rather to determine whether experimentation would be worthwhile.

After the models were created, grid sensitivity calculations were done to ensure that the results were not affected by numerical computation errors induced by grid size or aspect ratios.

Modeling Results

Copper Block. The modeling results for the copper block are presented in Table I. These results show that the thermal resistance between the die and the heat sink of the two package styles was identical, within modeling error and for the assumptions made in the model.

Table I
Modeled Thermal Resistance for Copper Block

Package Type	Thermal Resistance (°C/W)
Lidded	0.21
Lidless	0.21

Aluminum Evaporator. The results for the aluminum evaporator are shown in Table II. Again, the thermal resistance between the die and the heat sink was nearly identical between the two designs. The model shows a small benefit in the lidded design.

Table II
Modeled Thermal Resistance for Aluminum Evaporator

Package Type	Thermal Resistance (°C/W)
Lidded	0.24
Lidless	0.26

Modeling Summary. Given the considerable assumptions and simplifications, it was difficult to draw a strong conclusion based solely on the modeling results. Considering the small differences between the two designs, it was very compelling to construct the packages and measure them.

Measurement Results

Temperature measurements were taken for each of the four package and heat sink combinations. The results are presented in Tables III and IV. Included in each table are the power dissipation, heat sink temperature, die temperature, and thermal resistance. The thermal resistance column refers to the thermal resistance between the die and the heat sink. It includes the separate resistances of the die, epoxy and lid (if applicable), thermal grease, and a portion of the heat sink through which the thermocouples were embedded.

Copper Block. Table III displays thermal data from each package using the copper block heat sink and Dow Corning 340 thermal grease at the heat sink/package interface. Note that the thermal resistance decreased by 50% with the removal of the lid.

Table III
Thermal Performance of Packages with Copper Block

Package Type	Power Dissipation (W)	Heat Sink Temperature (°C)	Die Temperature (°C)	Thermal Resistance (°C/W)
Lidded	93.3	40.2	55.1	0.16
Lidless	93.3	40.6	47.6	0.08

Aluminum Evaporator. Data from the two packages with the aluminum evaporator acting as the heat sink and Dow Corning 340 thermal grease at the interface is presented in Table IV. Note that both the packaged die temperatures and the heat sink temperatures increased using the aluminum evaporator because it is not as efficient as the copper block. The thermal resistance decreased slightly for each package type over that obtained in Table III. This is most likely because of differences in thermal grease application or thermocouple placement. Finally, the thermal resistance decreased 53% upon removal of the lid. As expected, the decrease in thermal resistance is independent of the type of heat sink used.

Table IV
Thermal Performance of Packages with Aluminum Evaporator

Package Type	Power Dissipation (W)	Heat Sink Temperature (°C)	Die Temperature (°C)	Thermal Resistance (°C/W)
Lidded	85.8	63.9	77.1	0.15
Lidless	85.3	66.2	72.2	0.07

Measurement Summary. The measured thermal resistance of the lidded package compares very favorably with measurements taken by other investigators.[4]

Table V displays the amount of power that can be dissipated by each heat sink/package combination at equivalent die and air temperatures. The heat sink thermal resistance refers to the thermal resistance between the heat sink thermocouples and the ambient air. The results indicate that the lidless package is a significantly better performer than its lidded counterpart. The lidless package attached to the copper block is able to dissipate 34% more power, or 62 watts more than the lidded version. Likewise, for the aluminum evaporator, the lidless package is able to dissipate 15% more power or 15 watts more. Note that a larger relative improvement is realized by using a more efficient heat sink. These calculations assume no losses other than through the heat sinks but clearly show the superiority of lidless package designs over lidded.

Table V
Allowable Power Dissipation for Equivalent Die Temperatures of 110°C and Air Temperature of 50°C

Package Type	Heat Sink Thermal Resistance (°C/W)	Interface Thermal Resistance (°C/W)	Calculated Power Dissipation (W)
Copper Block:			
Lidded	0.17	0.16	180
Lidless	0.17	0.08	242
Aluminum Evaporator:			
Lidded	0.46	0.15	98
Lidless	0.46	0.07	113

The superiority of the lidless package over the lidded, while expected, may not be as obvious to predict as it first appears. One of the main arguments for keeping the lid on the package is that it decreases the heat flux by increasing the surface area through which heat can leave the package to the heat sink. By a one-dimensional analysis, it can be shown that the thermal resistance of the lid is an order of magnitude less than that of the epoxy. This indicates that using the lid as a heat spreader to decrease the heat flux through the package is not necessarily a bad idea. Rather, it is the bonding of the lid to the die with a layer of epoxy that makes it a relatively poor thermal solution. If a lid must be used for reasons other than thermal performance, it is clear that an effort should be made to reduce as much as possible the thermal resistance of the bonding material by decreasing its thickness and/or increasing its thermal conductivity.

Summary and Conclusions

The results from the modeling showed that the thermal performances of the packages were very similar and the lidless design warranted further investigation through lab measurements.

Comparison of the thermal resistances of the two package styles was very consistent for both the copper block and the aluminum evaporator measurement methods. Both measurement methods showed about a 50% improvement in thermal resistance in the lidless design.

While impractical for low-cost computer systems, the liquid cooled copper block measurements determine some limits of cooling of the HP PA 8000 die. The lidded design could dissipate 180 watts of power while the lidless solution could dissipate 242 watts while maintaining the temperature of the die within the limits for reliable operation.

The measured results indicate that the lidless package is thermally superior to the lidded design. For the aluminum evaporator, 15 more watts could be dissipated while maintaining the same die temperature. This is of particular significance because a heat pipe assembly is one of the present cooling designs for the HP PA 8000 processor.

To obtain the thermal performance required in next-generation chips, the cooling design will need to be solved as a coupled problem, considering the complete thermal path originating from the surface of the die and ending in the cooling air. The lidless package is one possible solution.

References

1. G.B. Kromann, “Thermal Management of a C4/Ceramic-Ball-Grid Array: The Motorola PowerPCTM and PowerPC 604TM RISC Processors,” *Twelfth IEEE SEMI-THERM Symposium*.
2. C.D. Patel and R.L. Blanco, “Thermal Management of Future High-Performance CPU Modules,” *1995 Hewlett-Packard Design Technology Conference.*
3. R.R. Tummala and E.J. Rymaszewski, *Microelectronics Handbook*, Van Nostrand Reinhold, 1989.
4. V.K. Nagesh, B. Afshari, K. Chen, C.C. Chang, P. Dawson, J. DelCampo, R. Kaw, and J. Leibovitz, “Area Array C4 SMA Packaging Technology,” *1995 Hewlett-Packard Design Technology Conference.*
5. C.D. Patel, personal correspondence with R.L. Blanco and C. E. Bash, November 1995.
6. *FLOTHERM User’s Guide, Version 1.4*, Flomerics Ltd., Surrey, England, 1993.
7. K. Azar and J.E. Graebner, “Experimental Determination of Thermal Conductivity of Printed Wiring Boards,” *Twelfth IEEE SEMI-THERM Symposium*.