Implementation of the PowerPC 601 microprocessor

by T. B. Brodnax
R. V. Billings
S. C. Glenn
P. T. Patel

To produce a marketable PowerPC™ microprocessor on a short development schedule, the logic had to be designed in a manner flexible enough to allow quick modifications without sacrificing high performance and density when customized cells were required. This was accomplished for the PowerPC 601™ microprocessor (601) with a high-level design-language description, which was synthesized for a gate-level implementation and simulated for functional verification. In a similar way, the physical design strategy for the 601 struck an attractive balance between a highly automated, flexible floorplan and the additional density that had to be available for limited, well-conceived manual placements. Finally, a rigorous test strategy was implemented, which has proved very useful in analyzing the processor and in assembling 601-based systems. Careful adherence to this methodology led to a successful first-pass physical implementation, leaving the second iteration for additional customer requests.

Introduction

When the alliance between Motorola Inc., IBM, and Apple Computer Corporation was formed to develop state-of-the-art microprocessors, a complete family of products was planned. Three of the originally announced processors would take two or three years to develop; it was uniquely the goal of the 601 project to bring to market quickly an attractive design that balanced cost and performance. An impressive set of architectural features were to be included in a die 10.95 mm square, running at 50 MHz under worst-case processing conditions. By each of these criteria, the PowerPC 601™ microprocessor meets or exceeds the initial goals. This paper describes the design methodology which was essential to this development, with specific details about logic design, physical design, testability, and the verification approach.

- *Overview of design methodology*

The design tools for the PowerPC 601 resulted from evolutionary changes to the tool set used to build the previous IBM RISC System/6000® (RS/6000) chips [1–3]. Two PowerPC 601 design tasks were performed on mainframes. Now, however, the proprietary tool set runs entirely on a workstation platform, using a common database for enhanced productivity. The 601 logic designers described the chip in a proprietary high-level language called Design Structure Language (DSL). The DSL compilers accept hardware constructs in a program-like manner and support many levels of hierarchy for a macro design approach. **Figure 1** shows the flow of operations from the DSL designs to two distinct tasks—behavioral verification and logical-to-physical design. Early in the DSL-level design, and more intensely as the gate-level design became available, the testability features were

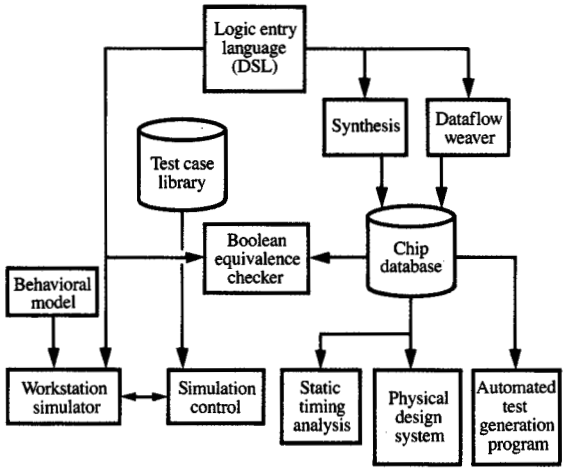

Figure 1

PowerPC 601 microprocessor design methodology.

integrated and evaluated. To further ensure that the verification model was equivalent to the gate-level description, a Boolean comparison was made between the two designs.

• *Computing resources*

The shared resources for the PowerPC 601 synthesis and simulation consisted of more than 100 clustered IBM RS/6000 workstations running under the control of a resource manager called *Qmanager*. Qmanager used a prioritized queue structure to dispatch, generate, and then simulate test cases in the background on the designer's workstations, thus taking advantage of any unused CPU resources. A dedicated server, *MenuCtrl*, was used for the automatic management and submission of test cases to Qmanager. The defect management server, *Xorbit*, provided the designer of a particular logical unit with the ability to manage reports on logic defects found during the development phase of the PowerPC 601. *BlackHole* maintained simulation history and status files, which served as the ultimate record of design progress and as the basis for all simulation status reporting. Qmanager also distributed synthesis jobs and testability jobs to the clustered workstations. These jobs did not require the automated gathering of statistics that the verification jobs required, but they did benefit from the load-balancing routines of Qmanager.

Logic design

• *Logic synthesis*

The 601 control logic was partitioned into 38 random logic macros (RLMs). The IBM synthesis tool, BooleDozer™, implemented the logic in a technology-independent form, then mapped it into the technology library developed for the 601 [4]. The library was composed of "books" which had been implemented by the physical designers. The resulting logic from the initial mapping was optimized for area. At this point, BooleDozer applied a number of transforms to the logic to attempt to realize the timing relationships in the logic designers' assertion files, even if the macros grew in cell count. The physical designer could control the amount of area he was willing to sacrifice for a given timing improvement. The transforms performed simple tasks—increasing power (and size) of a gate, duplicating logic to alleviate fan-out problems, swapping pins, or remapping a function to faster gates (e.g., AND/OR to NAND/NAND). The assertion files were critical; overly relaxed assertions might have allowed synthesis to leave unacceptable paths when RLMs were tied together, while overly aggressive assertions might have contradicted each other, causing timing problems and burdensome extra logic. Late in the development process, the global assertion file generator partitioned chip timing paths into RLM assertions, which saved roughly 10% of our RLM cells (a much-needed improvement for our floorplan team) and helped reduce some of the critical paths. Synthesis also provided two ways for the user to "hide" an intended implementation from synthesis. This feature was used sparingly, since the transforms usually optimized the logic as well or better than the designers could.

• *Logic design hierarchy*

For many of the challenging functions in our dataflow, customized memory elements, multiplexors, and arithmetic functions were employed. These circuits (designated off-the-shelf, or OTS) provided optimal timing and density. In parallel with the synthesis of the RLMs, the dataflow weaver (**Figure 1**) took all chip DSL and mapped the OTS circuits (which required no synthesis) into one file with the OTS circuits connected and with prototypes for the RLMs. These prototypes were "footprints" which contained the signal I/O for each RLM. The weaver output, a gate-level description of the chip's OTS blocks with RLM prototypes, was the "parent" input to our parent/child program. The "children" were the individual gate-level files for the RLMs. A "flattener" then tied the OTS gates and the individual RLMs into one "flattened" description of the entire chip.

• *Static timing analysis*

Static timing analysis for the 601 was performed using IBM's mainframe-based tool, the Early Timing Estimator (ETE). Design-specific input to ETE included the gate-level description, capacitances and resistance–capacitance (RC) delays, chip input/output assertions, and the clock waveform. Estimates based on fan-outs and input gate capacitances were used before floorplan-based RCs and capacitances were available. The technology-based timing library was composed of rules generated by the physical designers responsible for each macro. The rules included five coefficient delay equations—the intrinsic delay, two capacitance-based terms, and two terms based on input rise or fall times. Timing rules for memory elements and latches typically had checks to ensure that the input data arrived before the clock by a margin equal to or exceeding the setup time defined in the rule. These elements also typically included clock-to-data-out, which initiated a new cycle. **Figure 2** shows the typical latch scheme of the 601; logic gates were placed between latches using the L2 clock phase and latches using the L1 clock phase, facilitating simple static timing analysis. (An exception for cycle stealing is presented in the next section.) If a path was too long, ETE detected that a check had been missed and reported a negative slack for the path. It then became the responsibility of the logic designers to correct such paths.

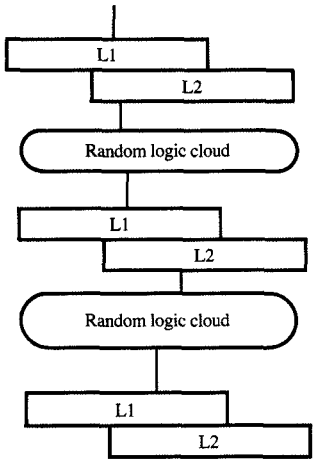

Figure 2

Ordinary latch usage.

• *Tuning methods*

Wherever possible, traditional simple methods were employed to tune paths with negative slacks. These methods included altering logic, changing fan-outs, altering the floorplan to improve loads, requesting enhancements to the technology library, and modifying the timing assertions that were given to logic synthesis. An interactive graphics browser, annotated with timing information, was employed by the designers in order to understand paths more quickly than by simply browsing through ETE reports.

Not all tuning methods used during 601 development were straightforward, however. The designer of the instruction fetch and dispatch logic encountered a problem for which a more elaborate solution was necessary. A typical timing problem for an aggressive dispatcher is in decoding instructions to determine their dispatch destinations. In the 601, the dispatcher took advantage of two things to relieve this problem. First, the cache data were available shortly before the end of the cycle in order to allow the data-in rotator to load data appropriately; second, LSSD (level-sensitive scan design [5]) rules allowed the placement of logic between any two latches having different clock phases. **Figure 3** shows how the rotated data are loaded into the L1 registers in the instruction queue while the L1 clock is high; during this clock phase, polarity-hold latches reflect their input on the output after some buffer-like delay. However, the L2 clock

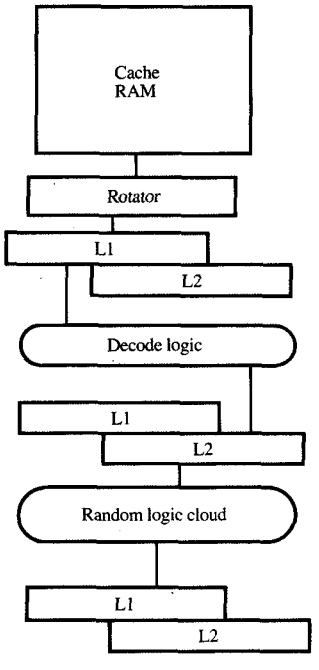

Figure 3

Cycle-stealing use of latches in the 601.

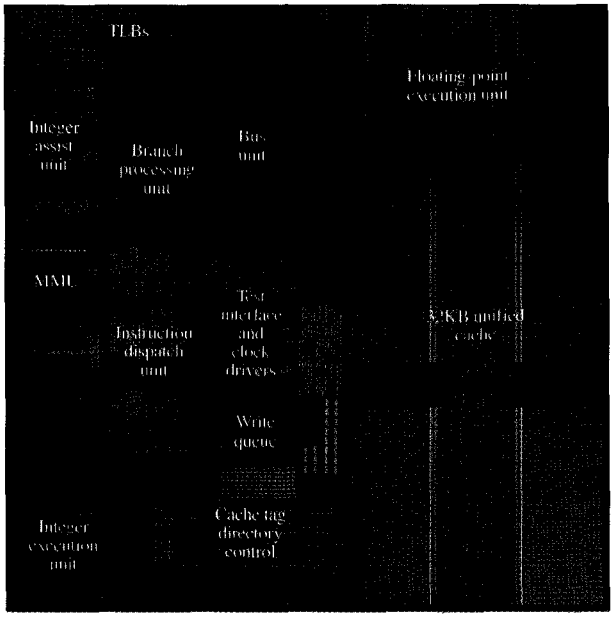

Figure 4

PowerPC 601 microprocessor die photograph.

is low, so the cache data do not pass to the output of the L2 latch until some time after the L2 clock rises (designating the beginning of the next cycle). The 601 dispatcher decodes instructions from the L1 latch elements in the instruction queue so that decoding can be started on the same cycle as cache access. The decode information is then stored in L2 latches. During the beginning of the cycle, when the decodes are actually finished, the L2 clock is high; thus, the decode information is passed directly through the latches to the dispatch logic.

Using the L1–L2 logic had several positive effects. First, the decode logic is started in the cache access cycle without having to be broken up by the setup time of an L1 latch to store partial results. Also, the decode logic is kept together, resulting in simplification and allowing a minimal space implementation. Because of the time gained by moving the decodes into the cache access clock, very little tuning had to be done here. Another benefit of this approach is that the data-in to data-out path is faster than the clock to data-out path in the latches used in the 601. This means that the latch burden was smaller for this section of logic than for sections using the standard approach.

The unfortunate effect of using the L1–L2 logic was that static timing analysis was made more difficult. Since there was no longer a simple launch point for the L2 data being fed by the decode logic, the standard static timing methods did not work here. We did use the standard methods, except that the setup times into the L2 latches were set at the rise of the L2 clock, and then the arrival time was adjusted back enough to make the setup, and the launch time was adjusted forward by the same amount. This scheme extends the range of paths ordinarily analyzed by our timing tool, ETE, without sacrificing the thoroughness that has contributed to past successes.

- *Boolean equivalence checking*

The scheme of simulating a compiled model of the DSL would have been inadequate if the gate-level description which fed the physical design of the chip had not been equivalent to the compiled model being simulated. This concern was dealt with by comparing the model with the gate-level description by doing a Boolean equivalence check. This check was done on each RLM, verifying that synthesis made no logical mistakes in its transforms.

Physical design

The PowerPC 601 microprocessor was fabricated using IBM's proprietary 0.6-μm minimum-feature process technology. The technology includes four levels of signal interconnection, a silicide layer for local interconnections, and a fifth, coarse level of metal for connecting chip I/Os to the module substrate. As a cost-saving measure, one wiring plane was deleted when wiring the chip. The chip die, shown in **Figure 4**, contains 2.8 million transistors in a 120-mm^2 area. The chip is packaged in a 304-pin ceramic quad flat pack using IBM C4 flip-chip solder-ball technology.

Physical design for the PowerPC 601 microprocessor was done with the IBM VLSI Interactive Design Automation System (VIDAS). This has proved to be a successful method for semicustom integrated circuits for each of the RISC System/6000 chips. As in most VIDAS-based chips, many critical design processes for the 601 were conducted in parallel. The primitive library, comprising RLM books and OTS books assembled at the transistor level, was developed in parallel with global floorplanning, logic design, logic verification, timing analysis, and tuning. Top-down floorplanning was followed, except for pieces of the chip for which the need for high customization was identified early. The eight-way set-associative cache RAM and its associated TAG directory are two obvious circuits which required a high degree of customization. The flexible manner in which control logic was synthesized and laid out allowed the 601 logic designers to make reasonable changes during the week when lithography masks were built for the chip. Finally, the VIDAS system had the necessary hooks for physical design checking, which minimized the possibility of any physical design errors.

• *Global floorplanning*

The 601 was designed on an image of multiple stacks, or placement columns. No perimeter area was allocated for input/output circuits; these were placed within the stacks, as can be seen in the die photo of Figure 4, where most of the off-chip drivers are in the third stack from the left. Placement of macros (both RLM and OTS macros) was done in two steps. First the macros were assigned to stacks. This was followed by placement within the stacks; initially a coarse placement was done, and this led to finer stack placement. Both floorplanning steps were done with a tool called CPLACE. The chip floorplanner could control as many of the placements manually as he believed necessary by interacting with CPLACE.

The next step was to route the power buses with a tool called POWERBUS. This program connected the power service terminals (PSTs) of each macro on a given plane and routed other planes at user-specified frequencies. Once the power buses were placed, the wiring evaluation (WEVAL) tool was run. WEVAL performed global routing on the chip, which was divided into user-specified grid sizes. WEVAL reported high congestion areas where wiring densities would lead to unwirable nets, suggesting areas where adjustments were necessary. Following wiring evaluation and its resulting floorplan adjustments, channel optimization (CHOPT) placed wiring bays in areas identified by WEVAL in the dataflow and reserved necessary porosity in the RLMs. A tool called RLMIOS was used to place control logic input/output pins in the optimal locations before the RLMs were actually built.

The hierarchical nature of the design system can be appreciated by looking at Figure 4. Notice that the space above the cache had some special constraints on it, primarily the effect of the cache's low porosity. A physical block had to be placed there which was the correct size and had a low number of fan-outs to or from the rest of the chip. The floating-point unit was a good fit when turned sideways and treated as one wiring stack. VIDAS had the flexibility to support this.

• *Primitive design*

The circuit library used for the 601 chip was designed jointly at IBM Austin, Texas and IBM Burlington, Vermont. The circuit library was divided into two portions—RLM circuits and OTS components. The RLM circuits (low-level functional blocks such as inverters, NAND gates, and And-Or-Invert circuits) were used to construct the control logic for the chip dataflow. The OTS components included more complex functions such as data registers, multiplexors, adders, and memory arrays. These components were highly customized. The 32KB cache is the best example of the densities possible in this arrangement; it contains almost two thirds of the transistors in roughly one third of the chip area (Figure 4). The OTS components were designed in four-bit increments and stitched together to obtain a desired component width. The density of the circuit designs benefited from the use of silicide local interconnections between diffusions and polysilicon and studded contacts between wiring levels. Another aspect of the macro design is the timing characteristic of the function. The macro designer runs a transistor-level simulator to generate the ETE library discussed in the previous section.

• *Control logic*

The control logic construction began in parallel with the global wiring effort. RLMs consist of a number of rows of standard cells that are wired to one another in accordance with the logic specification from synthesis. In addition, the previously mentioned RLM constraints (I/O locations and porosity) were honored. Placement of the RLM books was done with CPLACE, while the wiring was done with a maze router, LGWIRE. Global wiring at the chip level was also done by LGWIRE.

• *Tuning interaction with floorplanning*

Throughout the entire design process, timing information was provided by an RC delay estimator and calculator and the static timing tool, ETE. The RC estimator–calculator was run in mixed mode, using actual calculations for wired nets and estimates for the unwired nets. This allowed the designers to tune their logic with the most accurate delay estimates and calculations available.

• *Physical design checking*

The final phase of the 601 physical design was to check the design to ensure that no process ground rule was violated and that the physical implementation matched the logic design. Using the Hierarchical Design Verification (HDV) system, each RLM and OTS was checked for ground rule constraints and logical-to-physical comparisons. At the global level, HDV verifies that no global wire crosses macro internal wires, no macro shape interferes with other macro shapes, and intermacro connections are correct; it also checks for global wire ground rule violations. HDV provided fast responses, which allowed logic changes late in the design process without affecting the schedule.

Functional verification

Pre-manufacturing verification of the PowerPC 601 microprocessor required the development of several key methodologies for chip simulation. Both gate-level simulation and behavioral simulation were used effectively during the design and development stages of the project from initial logic entry to silicon production. A custom hardware platform was designed and built to accommodate silicon verification after manufacturing. This evaluation board (called the JUDGE board) was designed in parallel

with the 601 and was available at the same time as the first physical chip implementation.

Functional simulation allowed the logic designers to correct any defects and to verify the processor completely before implementing it in silicon, thus significantly reducing the time and expense required to develop the 601. In particular, behavioral simulation had the advantage of allowing the processor to be modeled in a variety of configurations with the inclusion of high-level memory and I/O simulations, while gate-level simulation modeled specific timing relationships among the different components of the chip. The goal of functional verification was to ensure that the high-level description (DSL) of the chip implemented the function specified by the PowerPC Architecture™ and met the stated goals of the PowerPC 601 microprocessor (e.g., performance, architectural enhancements). The functional verification process involved running streams of instructions (test cases) through the processor model and comparing the results with those obtained from a reference model. The reference model was an architectural simulator, which is essentially the same for any PowerPC™-compliant microprocessor with few changes for processor-specific architectural features.

During simulation, the processor model could be simulated by behavioral models of various system components. Because the 601 is a general-purpose microprocessor, the behavioral models were written to simulate many different types of systems. Thus, by adjusting the behavior of the stimuli at the chip boundary, the designers were able to simulate the processor for virtually any environment in which it might be used, before any specific system environment had been specified.

The chip verification methodology was key in delivering a high-quality first implementation within the severely constrained schedule. The following sections discuss the simulation environment, the verification methodology, and the JUDGE board used for hardware bring-up and verification.

- *Simulation environment*

Two simulators were used by the 601 project: Texsim, a two-state behavioral simulator, and Aussim, an event-driven gate-level simulator. Texsim was used to verify the logical correctness of the design, while Aussim was used to verify that the physical implementation of the design functioned properly.

Included in the Texsim simulation environment were simulations written in DSL; they simulated system components such as main memory, I/O space, the bus arbiter, and an alternate bus master. Most simulation was run in single-cycle mode, wherein only one clock phase is used. This was possible because Texsim was delay-independent and ignored race conditions; thus, the L1 latches always caused a cycle delay, while L2 latches always passed data from input to output in a single cycle. Some simulation was done with a multicycle model, in which the full two-phase clocking was implemented. The multicycle model allowed verification of logic which related to clock generation and control, such as LSSD and self-test function, as well as the engineering support processor (ESP) interface (used for chip/software bring-up). The final type of Texsim model was the multiprocessor (MP) model, a single-cycle Texsim model that used three 601 processors in the same system model.

There were several support tools for general Texsim simulation and debugging. Test cases contained initial conditions, an instruction stream, and final conditions. They could be generated by hand, or by a test case generator (random test program generator—RTPG) which included the reference model in order to correctly generate ending conditions. It was possible to generate an *all events trace* (AET) of the execution of a test case on the simulation platform. The AET could be viewed with a debugging tool which displayed a cycle-by-cycle trace of any signal in the model (including both latch and combinatorial signals, as well as the contents of array structures). The designer could use this debugging tool to find logic bugs quickly—in this environment, finding a logic bug was far easier than in a lab environment where only limited information about the state of the chip on a given cycle was generally available. Finally, it was possible to start the processor model in different states with the same test case. The cache could be started empty or with data from any source, as could the TLB. The procedure was controlled by a program called the run-time executive (RTX), which initialized the model for each test case (or injected a test case into a model which was already running).

The Aussim environment was somewhat less elaborate than the Texsim environment (Texsim was the primary simulator used during the development of the 601). Aussim is an event-driven gate-level simulator which uses information from static-timing runs to generate a time-based simulation (in contrast to the cycle-based simulation of Texsim). Events were scheduled on gate inputs based on the arrival time of the signals in the static-timing run. Aussim test cases were all hand-generated. There was a display tool for Aussim test cases which allowed designers to view internal signals on a time-step by time-step basis. The Aussim model was generated from synthesized logic, so not all of a designer's internal signals were available as in Texsim, but the model more closely resembled the actual implementation of the processor.

- *Uniprocessor verification methodology*

Three major cycle simulation strategies were used to verify the 601: unit simulation, architectural verification (AVP),

and implementation verification (IVP). Each strategy was emphasized in a different stage of the project. The first stage of the design process was initial high-level design of the processor, which was completed in approximately two months. After extensive review of the design from the first stage, the second stage of the design process was entered: design entry, in which initial DSL entry occurred. This stage took from six weeks to two months. The third stage involved tying the different modular pieces of the design together. Finally, in the fourth stage, the design was placed in various simulated system environments.

The first cycle-simulation verification strategy to be emphasized was unit simulation. This occurred during and after initial DSL entry and consisted of testing each of the major functional modules of the processor independently. The different functional modules of the 601 included the bus interface unit, the cache unit, the instruction fetcher and branch unit, the integer execution unit, the integer assist unit, the floating-point unit, and the test interface unit. The interfaces between these units were defined (from both a functional and static-timing perspective) in the initial design phase (stage one above). Thus, each module could be simulated independently by building a behavior simulation for the rest of the processor around it. A number of different methods were used to implement unit simulation.

For pieces of logic which contained a small number of inputs and easily predictable outputs (i.e., little internal state), exhaustive simulation was often used. This ensured that the function which was intended to be implemented actually was implemented (it did not ensure that the correct function was intended in the first place). By breaking the module into submodules, the debugging process at the module level was made easier. If the submodules work as they are supposed to, a failure at the module level implies that either the submodules' definition or their interconnection is wrong.

For the integer and floating-point execution units, standard test cases were used, and the instructions were fed to the units one at a time from a simulated memory hierarchy. The results of the test cases typically included registers contained in the unit under test, so the standard method of checking the end results made sense in the unit simulation environment for these modules.

For some of the more control-flow-oriented units (e.g., the cache and the fetch unit), a more rule-oriented approach was taken. For the fetch unit, a set of rules about the behavior of the module was defined. Then a random stream of instructions was fed to the design, and the behavior of the design was checked against the rules. It was assumed that the execution units would correctly execute an instruction once it was dispatched to them, so there was little or no result checking for the fetch-unit simulation. When a rule was broken, a snapshot of the state of the simulation was taken. The snapshot included the previous several hundred cycles and a random number seed so that the simulation could be recreated once the bug was fixed (in order to verify the fix).

The second cycle simulation strategy to be emphasized was the AVP strategy, which was used once the functional modules had been tied together to form a chip simulation model. The AVP strategy was directed toward verifying the architectural correctness of the design, and involved running automatically generated test cases through the model. The test cases increased with difficulty as the implementation of the design became more functionally stable. The first AVP test cases had only one instruction each; next came test cases with ten instances of the same instruction, and finally, alternating instructions. After this stage, more tools were introduced into the methodology.

The first phase of AVP testing used a fairly static set of test cases (several hundred at each of the three levels). A single set of biasings in the RTPG can actually generate many different test cases, varying anything from the translation state of the machine (virtual versus real mode) to the instruction stream and dependencies, to exceptional data cases. A given set of biasings for RTPG is referred to as an RTPG menu. The platform for running cycle simulation was a network of RISC System/6000 computers and RT PCs®. An RTPG menu could be created and then submitted to a set of resource control programs. The program which kept track of the RTPG menus in the system was called *MenuCtrl*, while the program which handled the computation servers was called *Qmanager* (see above). Menu control would submit a menu to Qmanager multiple times, and each submission would create multiple test cases which were then automatically run against the most current chip model. If a test case passed, it was simply discarded, but if it failed, the failing test case and some run-time information (the information needed to recreate the failing condition—e.g., random number generation seeds) were sent to a storage location for fails. Statistics for each menu were kept in a database (BlackHole). Using this strategy, hundreds of millions of cycles of simulation were run on a daily basis.

The third cycle simulation strategy to be emphasized was the IVP strategy. A specific implementation will generally have edge conditions and complicated controls related to a specific set of conditions or operations. The AVP strategy was not biased toward any specific implementation, and therefore was not the best way to find defects in these specific areas (where defects often reside). The IVP strategy was aimed at these areas. The IVP strategy was also used to exercise sections of logic which were never exercised by random test-case generation (e.g., certain hardware bring-up modes).

For each functional unit, specific edge conditions were defined which had to be tested in concert with various

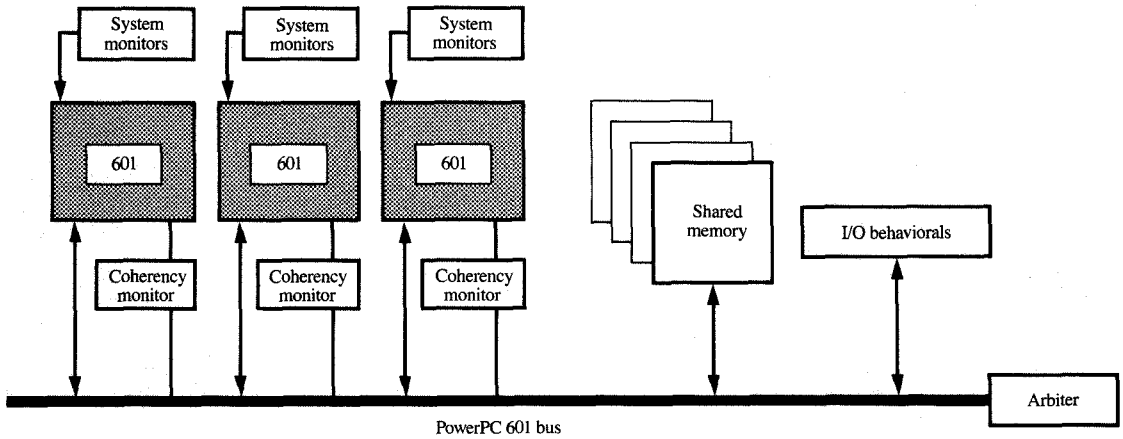

Figure 5

PowerPC 601 multiprocessor simulation model.

adverse situations. For example, the floating-point unit might have had a set of tests involving conditions for which two operands were denormalized numbers. The test cases generated as part of the IVP strategy were sometimes generated automatically but were often hand-coded.

Automatic generation was used for some units which had excessive edge conditions—largely the control-oriented units (instruction fetcher, cache unit, memory unit). It was possible to run a program during simulation which monitored signals in the model for a specific event or set of events. These monitor programs would look for events or sequences of events and then send the test case back to the designer, along with a report of the event which was found. These test cases were then stored as a regression suite for the unit in question. The report of the event found was then put into a database which kept track of which events of the desired set of events had been found. When a set of related events was not found by the monitor programs, the designers could gain insight into which parts of the design were being tested by the random AVP strategy, and could either get RTPG "fixed" in order for it to cover that class of events, or concentrate their efforts on those areas with handwritten test cases.

DSL checkers—simulation monitors written in DSL which monitored the design for invalid states—formed another verification mechanism which was used throughout the development of the 601. These checkers monitored the design, looking for logically incorrect states or state transitions which did not cause the test case to fail, or which would aid the designer in debugging a failing test case.

- *Multiprocessor verification methodology*

Support for multiprocessing was a key goal for the PowerPC 601 microprocessor and for the PowerPC Architecture. This was difficult to achieve, given the vast solution space for MP systems, and in particular the amount and complexity of logic required to support symmetric MP systems. Given the aggressive schedule that had been set for the PowerPC 601 and the requirement for high-quality functionality, the PowerPC 601 had to provide the right set of minimum-complexity features to support MP, and it had to obtain a very high level of functionality in the first physical implementation.

When implementing the MP verification methodology, the verification designers concentrated on verifying multiprocessing on both a uniprocessor model, as described earlier, and a tightly coupled shared-memory system model, as shown in **Figure 5**. The shared-memory model was necessary to ensure shared-memory interaction among multiple processors for the verification of serialization, coherency, and synchronization. The methodology consisted of the combination of IVPs, monitors, and a custom MP test-case generator. Although a subset of IVPs was developed, generating many of the events with IVPs was difficult and time-consuming because they relied on precise sequencing of the implementation. Therefore, the designers used a custom MP test-case

generator and specifically biased menus to produce random test cases for a three-way processor model. Memory coherency was difficult to verify with random test cases, because many types of failures might never cause a test-case failure. To address this problem, a *coherency monitor* was designed and included with each processor in the MP model to monitor bus transactions that did not originate from that processor. If such a transaction was detected *and* if that transaction was marked *coherent*, the monitor would check the cache and all other pipeline positions that required snooping to determine whether a matching address was found. If found, the monitor generated expected signal outputs for the PowerPC 601 as well as expected internal state changes. When the appropriate state change should have occurred, the monitor checked that the changes were correct. If not, the monitor interrupted the PowerPC 601 and terminated the test case. In addition, other monitors were designed to verify memory-cache coherency within the multiprocessor model.

- *Gate-level simulation*

Behavioral simulation is clearly the most efficient method for testing pipeline interaction in an evolving design. Unfortunately, there exists a limited set of tests which require some timing data from the actual implementation. Aussim was used with handwritten tests to test a number of operations: checking for faults on the 601 bus, ensuring that the test features worked but were not enabled during normal operation, ensuring that the power-on-reset function was correct, and ensuring that synchronous I/O functioned as expected.

- *Silicon evaluation board (JUDGE)*

The JUDGE board was designed in parallel with the 601 processor. At the time of first physical implementation, the JUDGE board was available to test the first parts which came from the fabrication facility. The JUDGE board was designed to allow the designers to bring up the 601 chip independently of an actual system in order to avoid having to deal with system bring-up problems (dividing the problem into two pieces reduces it to more easily manageable tasks) and because a system-independent test bed was desirable for a processor which was to be used by many diverse systems. The JUDGE board implemented the bus protocol using FPGAs, which allowed the designers to easily alter the bus parameters. Rather than running an operating system and applications, the JUDGE board used the ESP interface in the 601 to load test cases (just like the ones run in simulation) into the machine. This gave a much more controlled environment for debugging the processor. The JUDGE board could be considered a logical extension of the overall verification methodology for the 601, rather than a custom system designed around the 601.

Test strategy

- *General philosophy*

IBM has used level-sensitive scan design (LSSD) [5] stuck-fault testing as the backbone of its test generation for many years. The LSSD strategy, partitioning the chip into observable scan strings of a few hundred latches, has enabled a variety of chips to be successfully tested for manufacturing defects. The PowerPC 601 also incorporates dynamic built-in self test (BIST), weighted random pattern testing (WRPT) [6], and IBM's newest test dynamic delay transition test with constrained timings.

It is always a difficult challenge to maintain the delicate balance between physical constraints and logical and test features. Since the 601 was a derivative of the RISC System/6000 single-chip (RSC) processor, a test structure already existed at the beginning of the 601 design effort. Because of the rigid test structures put in place by the RSC chip, the 601 designers had only to enhance the existing RSC structures, with careful attention to all new components added to this chip. For example, the simple 8KB cache RAM used on RSC was replaced with an eight-way set-associated, read-modify-write 32KB cache which requires a more complex test strategy.

This elaborate test strategy may seem expensive, but it provides two significant services after the chip has been manufactured. First, the test strategy allows us to detect a variety of manufacturing defects—both those which affect static performance and those which affect dynamic performance. Additionally, the observability features are invaluable in debugging a specific instance of the 601 inserted in a system. Ordinarily, system developers have very little data to analyze when they encounter an unexpected result in a hardware system, but the common on-chip processor (COP) addresses this concern.

- *Basic test elements*

The 601 test strategy starts first at the primitive level, the book. When a book is designed and physically created, it is tested for 100% stuck-fault coverage. All physical designers are responsible for making their designs 100% stuck-fault testable. They also use the stuck-fault patterns in a switch-level simulator to prove equivalency between the test model and the physical model. In many instances, the physical designer must make modifications to enhance testability at the primitive phase of a design. An example of this[1] is a subtle change to the design of a two-to-one multiplexor which makes a significant difference in the

[1] S. F. Oakland and P. E. Perry, "Testability Analysis of Selector Circuits," IBM internal memo, January 1991.

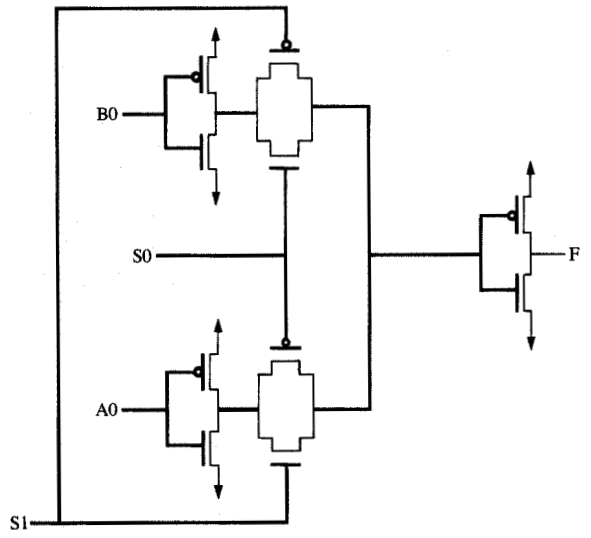

Figure 6

Traditional 2:1 multiplexor.

testability. **Figure 6** shows a more traditional multiplexor design which has untestable faults when both selects are on. **Figure 7**, however, shows that a simple modification leads to a 100% testable version. A known output state of the multiplexor is present when both selects are off, and another known output state is present when both selects are on and opposite values are on the data ports. Testability is evaluated at each level of the hierarchy beyond the primitive book: the RLMs, the merged OTS books, supermacros (combinations of RLMs and OTS smaller than the entire chip), and finally, the chip itself. Extensive test generation and fault analysis along with redesign of the test macros brings these pieces to a 99% test coverage range. At this time, full chip analysis begins, with a search for redundancies and untested faults. Work is coordinated with the logic designers, as they tune their logic, to aid the test engineer in removing global redundancies and to increase fault coverage.

• *Test features*

It has long been known that LSSD static stuck-fault coverage alone leaves a variety of manufacturing defects undetected, encouraging the 601 design team to employ several additional test strategies. Increased clock frequencies and densities emphasize the need for these new strategies, which target the manufacturing defects that LSSD misses, namely opens, resistive paths, and slow transiting paths. These test features concentrate on the dynamic performance issues on this chip, along with I_{DD} testing.

The objectives for 601 test were to achieve static stuck-fault coverage greater than 99.97% and a dynamic transition fault coverage greater than 90%. In addition to LSSD and WRPT tests, the 601 testing methodology also includes static I_{DD} test, engineering support processor functional tests, parametric test, logic BIST, and array BIST. Static I_{DD} test is used to test the limits of the quiescent current and to detect shorts to V_{DD} or ground. The functional tests include "sort AVPs," which specifically exercise certain critical paths of the chip, and a suite of AVPs that exercise the instructions and chip I/O pins. The 601 also has partial Joint Test Activities Group (JTAG) compatibilities built in to allow the use of the IEEE 1149.1 interface features. The COP is used to control the BIST. The logic BIST uses the pseudorandom pattern generator and the multiple-input scan register to generate and compress the signature. Two types of array BIST were implemented. One array BIST tests all of the arrays in parallel and is controlled by the COP. The local cache array BIST, which is initiated by the COP but controlled by the BIST controller, was built into the cache. Both array BISTs could run in a pseudorandom or a deterministic mode. The PowerPC 601 test engineers implemented timing tests based on estimates from the static timing analysis of the chip. Rather than simply running all patterns at a given frequency, the clock pulse is altered depending on the paths being exercised. This dynamic testing provides more timing information than simply the chip's operational frequency.

• *Common on-chip processor (COP)*

The COP serves as a powerful diagnostic engine. It not only controls the chip's BIST, but also serves as the interface for debugging and analyzing the 601 processor chip and surrounding elements on the planar board. As the interface to the ESP, it allows the design engineer to run instructions on the processor, stop the processor, and unload the contents of every memory element of the processor. The design engineer can then analyze unexpected conditions, reload all the memory elements in the processor to any desired state, and restart the processor to continue with the rest of the instructions. During BIST, the COP fully controls all the operations necessary to perform both a global BIST (logic and array) and a cache BIST. It also serves as a powerful diagnostic tool, in conjunction with the ESP, to analyze all array failures. This is accomplished by running BIST on a test planar board with two 601s on it. On this test board, one 601 is a module known to be free of defects and the other is the one under test. Software in the ESP is run to output

information concerning the failing array's data and address, and the possible physical failing location on the chip. The COP, in conjunction with the ESP, provides control and observability to all of the functional testing procedures used on the 601.

Summary

We continue to develop this design system for other projects. As stated above, it allowed us to deliver the first implementation of the PowerPC architecture in less than a year. The core of this design system is now commercially available for consideration by all integrated circuit design centers.[2]

We achieved a successful first physical implementation, leaving the second iteration for additional customer requests. At 66 MHz, the PowerPC 601 microprocessor attains 62.6 SPECint92™ and 72 SPECfp92™, based on measurements on the IBM RISC System/6000 Model 250. The success of the design team in applying state-of-the-art CMOS technology in a compact single chip gives the 601 an outstanding price/performance ratio and makes it well suited for a wide range of system designs [7, 8]. It exceeds the performance of larger processors which use the more complicated (and expensive) BiCMOS technology. The design methodology and its creative application have allowed us to make the 601 available at a fraction of the price, size, and power consumption of processors with comparable performance.

Acknowledgments

This was a unique project with aggressive function, performance, and schedule goals. It required the cooperation of many people from Apple, Motorola, and several divisions within IBM. In addition, the project was viewed as a barometer for the health of the overall relationship of the alliance. In the end, the team accepted these expectations and met all of the project goals. Beyond this specific work, the design methodology represents years of hard work and clear vision from many people within IBM. Finally, the authors are grateful to Terry Potter for his numerous suggestions improving the readability of this paper.

[2] IBM presentation at the 30th Design Automation Conference, June 1993.

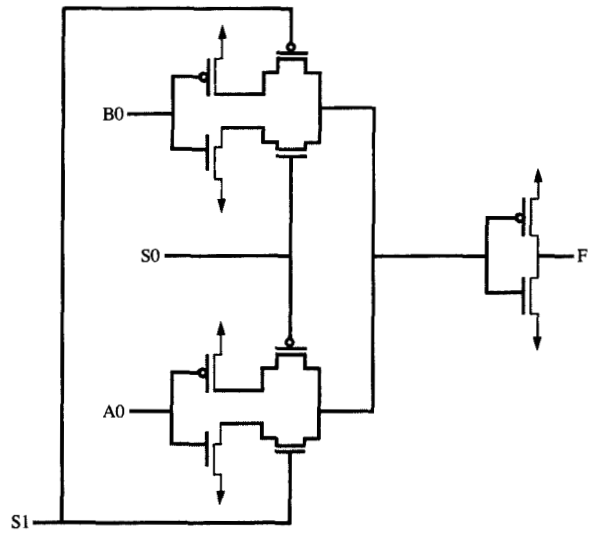

Figure 7

2:1 Multiplexor with testability enhancement.

References

1. C. H. Feistel, H. Hoffman, G. B. Long, and G. A. Nusbaum, "Structured System Design and Verification," *IBM RISC System/6000 Technology*, Order No. SA23-2619, 1990, p. 86; available through IBM branch offices.
2. J. W. Cagle, P. T. Patel, B. I. Waters, and P. G. Villarrubia, "Semi-Custom Chip Design Methodology," *IBM RISC System/6000 Technology*, Order No. SA23-2619, 1990, p. 92; available through IBM branch offices.
3. E. Seewann, S. L. Runyon, R. K. Montoye, Q. Nguyen, and J. C. Ridings, "VLSI Circuit Design for the RISC System/6000 Processor," *IBM RISC System/6000 Technology*, Order No. SA23-2619, 1990, p. 98; available through IBM branch offices.
4. D. Kung, R. Damiano, and T. Nix, "BDDMAP: A Technology Mapper on a New Converging Algorithm," *Proceedings of the 29th Design Automation Conference*, Anaheim, CA, June 1992, IEEE, pp. 484–498.
5. E. B. Eichelberger, "Method of Level-Sensitive Testing a Functional Logic System," U.S. Patent 3,783,254, September 25, 1973.
6. F. Motika and J. A. Waicukauski, "Weighted Random Pattern Testing Apparatus and Method," U.S. Patent 4,688,733, August 1981.
7. C. R. Moore, "The PowerPC 601 Microprocessor," *Proceedings of COMPCON 1993*, San Francisco, February 1993, IEEE, pp. 109–116.
8. Brian Case, "IBM Delivers First PowerPC Microprocessor," *Microprocessor Report*, Oct. 28, 1992, p. 1.

Received October 8, 1993; accepted for publication June 27, 1994

Timothy B. Brodnax *IBM Systems Technology & Architecture Division, 11400 Burnet Road, Austin, Texas 78758.* Mr. Brodnax received the B.S.E.E. degree in electrical engineering from the University of California at Berkeley in 1981 and the Master's degree from the University of California at Los Angeles in 1982. Prior to joining IBM, he was a member of the technical staff for Hughes Aircraft Company in Los Angeles. Mr. Brodnax is currently an advisory engineer, working on POWER/PowerPC microprocessor designs. He holds one patent and has presented papers on various aspects of integrated circuit and processor design.

Richard V. Billings *IBM Systems Technology & Architecture Division, 11400 Burnet Road, Austin, Texas 78758 (RBILL at AUSTIN; rbill@austin.ibm.com).* Mr. Billings is an advisory engineer in design for testing in the PowerPC RISC processor development group. He joined IBM in 1974 after graduating from the State University of New York at Morrisville. His assignments have included device and process design and OEM test manufacturing. In 1986 he acquired his B.S.-equivalent degree while working on test manufacturing in the IBM General Technology Division at Manassas, Virginia. In his current position Mr. Billings has been associated with test development for RISC System/6000 and PowerPC parts; he is lead test engineer for the PowerPC 601 and 630.

Scott C. Glenn *IBM RISC System/6000 Division, 11400 Burnet Road, Austin, Texas 78758.* Mr. Glenn received the B.S. degree in computer engineering in 1989 from Baylor University in Waco, Texas. He joined IBM in 1989 as a software engineer. Since that time he has worked as a verification engineer for the RISC single-chip (RSC) microprocessor and as an engineer on the design and verification of the PowerPC 601 microprocessor. He is currently working as a PowerPC consultant for the OEM area of the RISC System/6000 Division.

P. T. Patel *IBM Systems Technology & Architecture Division, 11400 Burnet Road, Austin, Texas 78758 (PTPATEL at AUSVM6).* Mr. Patel joined IBM in Manassas, Virginia, in 1973 upon receiving the M.S.E.E. degree from the University of Connecticut. He worked in various bipolar circuit design activities there, then transferred in 1978 to Burlington, where he worked on the I2L circuit technology. He transferred to Austin in 1980 and continued to work in the area of VLSI design. Mr. Patel was the lead designer on the memory management chip for the RT PC system; he defined the design methodology for the RT PC, RISC System/6000, and PowerPC 601 designs. He was also the advisor to the design team for the RISC System/6000 processor chip set. He is currently a senior technical staff member in RISC processor development.