The role of two-cycle simulation in the S/390 verification process

by Gary A. Van Huben

Microprocessor design techniques have evolved to a point where large systems, such as S/390® servers, can be constructed using relatively few, but very complex, application-specific integrated circuits (ASICs). Delivery of a quality design in a timely fashion requires that several design activities progress simultaneously, with different types of verification used within the various design disciplines. This paper discloses a simulation method capable of functionally verifying a physical implementation of the design at a system level. The aggressive design schedule undertaken on the S/390 Parallel Enterprise Server G4 program required additional advances in simulation beyond those employed in the development of the IBM Enterprise System/9000® (ES/9000®) processor family. A new type of cycle simulation was developed to supplement the incumbent strategy of using conventional cycle simulation to verify system function combined with Boolean equivalence tools to perform logical-to-physical comparisons. This two-cycle simulation method was invented to verify areas such as logic built-in self-test (LBIST), array built-in self-test (ABIST), clock trees, firmware level-sensitive scan design (LSSD) rings, and large custom arrays, which are typically omitted by existing system verification methods. The creation of the two-cycle simulation model is discussed, along with several uses of the model and the types of errors uncovered.

Introduction

One of the greatest problems surrounding large submicron logic design projects today is the need to begin detailed physical design of various components while the system-level function is still being developed. This is particularly apparent in large array designs where early floorplanning, layout, and wiring analysis may cause several iterations of the array design while the functional designers attempt to simulate the interaction of the array controller with the rest of the system. A common method for handling this situation is to build a simulation model comprising the actual control logic coupled to an array macro. As the design matures, the array macro is updated to describe the array behavior with greater accuracy. In today's

©**Copyright** 1997 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

0018-8646/97/$5.00 © 1997 IBM

environment, time-to-market schedules do not include sufficient time to update and maintain macros for all components that require early detailed physical design.

Another problem facing logic designers of large ASICs surrounds verification of nonmainline functions such as clocks and scan rings, where the behavior tends to be abstracted or completely omitted from the register transfer language (RTL) description. This is especially prevalent in synthesized designs, where it is more desirable to allow the synthesis and post-physical-design tools to perform clock-balancing and scan-ring optimization. During the development of the ES/9000* processor family, these portions of the physical design were verified using specialized tools which performed connection and electrical checks, but never verified their operation during system-level functions.

One common approach to solving these verification problems is the use of a circuit simulator to functionally verify the operation of a physical design. This approach has limitations, such as size and speed, which render it impractical in large S/390* systems comprising millions of gates. At the same time, the problem cannot be ignored, since today's technology advances allow large arrays and control logic to fit on the same ASIC. This increases the importance of finding problems prior to releasing the design to manufacturing, since a single problem in an array or clock circuit may render the system useless or nonfunctional in a hardware test environment. The remainder of the paper focuses on several uses of a two-cycle simulation model to verify various functions of the S/390 Parallel Enterprise Server G4 machine.

Array verification

One of the differentiating factors in the G4 program was the close interaction between the array and logic design teams. Historically, arrays used in S/390 processors were treated as commodities where high-level behavioral macros were used in simulation models. The actual array designs were not truly verified in the system environment until engineering hardware was exercised on the test floor. Several factors in the G4 program rendered the array verification a critical link to the success of the project. For example, the performance of the machine relies heavily on the implementation of the large arrays, especially the L1 and L2 caches. Since so much of the system function depends on the interaction of the arrays with the surrounding logic, it was imperative to accurately verify their function in a system environment prior to the initial tape-out. In response to this need, a plan was established to verify three aspects of the array design: "black box," internal array behavior, and array built-in self-test.

- *"Black box" verification*

As stated earlier, parallel design efforts played a key role in the success of the G4 program. For example, logic simulation models were built as soon as sufficient I/O information was available for the arrays. This meant that a simplistic functional macro was essentially substituted for several hierarchical levels of array design. As the array design matured, changes were made which were intended to improve the physical design while preserving the logical behavior. Once this deviation occurred, the macro representation used in the regular cycle-simulation model became a "black box." It thus became the responsibility of the two-cycle model to verify these black boxes. For example, the data buses associated with the L2 cache were logically arranged in such a manner that the first 64 bits represented the data and the last 8 bits represented the error checking and correction (ECC) bits. All schematics which interfaced with the L2 cache black box assumed this convention. However, the cache-array designers found it beneficial to scramble the bits in order to optimize timing and wiring. Thus, a two-cycle model, which incorporated the actual array schematics, was built to find mistakes in which the incoming data were scrambled differently from the outgoing data. Since many people were involved in the detailed implementation of the black boxes, it was not surprising that several mistakes of this kind were found.

A second aspect of the black box which required verification was the logical timing of the read and write operations. For example, many of the arrays utilized an array clock which was aligned with a trigger clock. In order to prevent write-through and LBIST problems, L1-only latches were inserted between the array interface and the actual static random-access memories (SRAMs). These L1-only latches were used for the data path, the address, and the write-enable signals. For a write operation to occur in the real design, it necessitated both an L1 (latch) clock and the array clock to propagate the data from the interface through the L1 latches and into the SRAM. With respect to the logic design, this translated into data being written one full cycle after presentation to the array interface. These characteristics had to be properly modeled in the black box macro behavior for use in the regular single-cycle simulation model. Sometimes these characteristics were miscommunicated or improperly coded in the high-level behavior, thus rendering the results of the single-cycle simulation model invalid. Once again, it was the responsibility of the two-cycle model to verify that the arrays were operating in synergy with the surrounding control logic.

The above approach was taken because most of the black box was still in the conceptual design stage when the need arrived for a functional single-cycle simulation model. Certainly, an alternate approach is available for situations where the underlying design is understood at the time a simulation model is required. In this case, the black box could be defined as the actual arrays and registers, but would exclude the support logic and any

physical design changes such as data scrambling. For example, consider a macro which consists of an SRAM, the support logic (for decoding address bits, enabling read and write modes, multiplexing data ports, etc.), and observation registers used for hardware debugging. In the G4 design methodology, most of the design was unknown at the time a functional simulation model was required. Therefore, the entire macro was treated as a black box; a simple array behavioral description was written to emulate the correct logical function, but it omitted much of the detail. Once the real design was available, a two-cycle model ensured that the underlying detail did not break any of the logical assumptions imparted in the black box behavior. However, if the underlying detail is known, or will be available in the required time, the alternate approach would include the entire design in the single-cycle model and would actually exercise the address decoders, data port multiplexors, etc., but would black-box the RAM and registers using a simplified single-cycle latch and array primitive. Eventually a two-cycle model may be desired to verify the real latch and RAM designs, but the surrounding support and debugging logic will already have been verified. This approach provides the advantage of maintaining only a single macro design to service both simulation environments. The substitution of latch and array primitives can be handled by the tools themselves. However, if the underlying macro detail is significant, or the macro is subject to constant iteration, the single-cycle functional simulation may experience a productivity delay due to either reduced performance or constant churn. Each candidate for black-boxing must be evaluated independently on the basis of the goals, schedules, characteristics, complexity of the design under consideration, and the maturity of the simulation environment. Depending on these factors, one approach may be more desirable than the other.

- *Internal array behavioral verification*

Once the array interface issues (such as data scrambling and logical timing) were verified, emphasis was placed on qualifying the behavioral representation of the internal array design. In the G4 project, each array macro was represented by an accurate VHDL [1] description. Unlike the high-level black box behaviors which modeled basic read and write operations, these behaviors described all aspects of the array design, including responses to nonmainline stimuli such as ABIST, LBIST, scan, and device leakage (IDDQ) testing. The smaller arrays, such as the buffers and register files, were verified using the exhaustive array verification process discussed in a companion paper in this issue [2]. However, the large arrays such as the directories and caches used the following deterministic array verification process to qualify the behaviors. Qualification of these behaviors was deemed critical in ensuring that the simulation results were achieved against accurate behavioral representations of the array circuit designs.

The first step in the deterministic array verification (DAV) consisted of creating a series of test vectors for the internal circuit simulator (Parch) in order to exercise the physical design of the arrays. This step was performed by the array designers using test vectors whose outcome was known or predictable. This step was iterated until the array designers felt they had enough test vectors to satisfy them that the actual array circuitry was performing properly.

The next step necessitated development of a special process which takes Parch test patterns and converts them to simulation test cases using an internal simulation application program interface called SIMAPI. These SIMAPI test cases could be used to exercise an independent two-cycle model constructed from the detailed array behavior. Since the Parch test patterns contained both the input stimulus and the output responses of the array I/O, they could produce self-checking SIMAPI test cases by using the output responses as *expected* results. A third step consisted of a derivative of the two-cycle simulation model build process in which pieces of the process were extracted and customized for building automated stand-alone array models. All three steps were integrated into the DAV process, which also included several data management techniques to ensure synchronization of the input Parch test patterns and array behavior with the generated array two-cycle model and SIMAPI simulation test cases.

The deterministic array verification process proved to be very valuable in locating inaccuracies in the array behaviors. Some examples of the kinds of problems encountered include the following:

- Array behavior when multiple late-select lines (which are normally orthogonal) were activated simultaneously.
- Array behavior when normally orthogonal operational modes were activated simultaneously. An example of this would be pulsing the scan clocks while ABIST was active.
- Array behavior during an unusual or illegal clock or control sequence.

Although this technique does not guarantee that every conceivable operation works correctly, careful generation of Parch test patterns does result in thorough validation. Furthermore, this technique provides the critical verification link between the physical design and a behavioral description capable of participating in high-speed cycle simulation. By centering the system verification strategy around cycle simulation, the dependency on slower circuit and event simulators is

reduced, and more sophisticated system-level functions can be exercised.

• *Array built-in self-test (ABIST)*

Once the behavioral descriptions of the arrays are qualified either through exhaustive or deterministic array verification, complex functions such as array built-in self-test (ABIST) can be simulated. Using a combination of the two-cycle model in conjunction with powerful simulation engines such as ZFS (the internal cycle simulator), the G4 project was able to simulate entire ABIST sequences on all of the large arrays in the system. Some arrays were exercised using a suite of ABIST programs, each averaging several thousand cycles, while other arrays required one or two programs which ran several hundred thousand cycles. This approach not only enabled verification of the ABIST logic and the arrays, but the simulations concluded that the clock logic was working properly, multiple copies of the same array macros ran identically, and all interconnections were properly wired.

Perhaps one of the most interesting uses of the ABIST simulation was in the area of multiple-input shift-register (MISR) signature predictions for those arrays which were included in an early-release test chip. By the time the array behaviors and two-cycle model were validated, the arrays on the test chip had been thoroughly examined in the laboratory. Sufficient ABIST exercises had been run in the laboratory to produce what were believed to be "golden" MISR signatures. These same ABIST programs were injected into the two-cycle model to ensure that the same MISR signatures could be obtained through simulation. This exercise served as an ABIST certification process for the two-cycle model which proved beneficial in two ways. The primary benefit came when the design was modified in a way which changed the MISR signatures. The two-cycle model was used to *predict* the new signatures so that manufacturing would know, upon producing the first wafer of the new tape-out, whether the arrays were defective. A secondary benefit occurred in the latter stages of the project, when changes were made to the array design which were not intended to change the MISR signatures. Once again, the two-cycle model was used to immediately confirm that the expected MISR signatures were still the "golden" signatures.

Scan-ring simulation

Unlike its predecessors, the G4 machine was heavily dependent upon LSSD scanning for many aspects of power-on, system reset, and hardware initialization. Therefore, it was critical to find a means of verifying the thousands of registers linked on the scan chain. The two-cycle model enabled new types of verification tests to be developed to certify the scan operation at several levels of the design. Although it is possible for a single-cycle simulation model, using simple latch models, to test the scan connections, the two-cycle model affords one the opportunity to simulate an actual scan operation interacting with the real system clocks. This level of detail requires an accurate multiclock latch model which can accept all of the necessary clock connections and mimic the L1-to-L2 transitions during the proper clock phases. Opportunities exist in this environment to detect nonscannable pieces of the design due to errors in the clock connections or mistakes in the clock-sequencing logic. Since it is still beyond the practical limits of cycle simulators to emulate all of the scanning necessary to power-on the entire machine, the problem was attacked in a piecewise fashion.

First, the scan chain on each chip was simulated to ensure that every shift-register latch (SRL) was properly connected. For both the processor and L2 chips, the architectural verification program (AVP) or random test-case simulation environment made possible a clever approach to verifying the scan operation. The chip would run several thousand cycles of normal system operation, and at a random point simulation would pause. The system mode registers would be saved, and the chip would be placed into scan mode. The chip's scan clocks would be pulsed once for each register on the chip, with the chip scan-out pin wrapped back to the scan-in pin. In effect, the entire scan ring was rotated until data in a particular register propagated through every other register and back to their starting point. Once completed, the chip would be restored to system mode and the system clocks would resume. As long as the scan rings were connected properly, the random or AVP test-case simulation would continue flawlessly, with the scan operation having no impact. In order to ensure that the chip was really scanning, the scan-out pin was monitored for a random data pattern. In addition, the test was also run using an incorrect number of scan-clock pulses to ensure that the random simulation failed.

Upon reaching system simulation, a final scan-ring check was made to certify the interconnections among all of the chips, as well as proper operation of the clock chip. This test usually consisted of loading a known data pattern into a predetermined position in the scan chain. After instructing the clock chip to perform the required number of scan-clock pulses, the data stream was monitored for the expected pattern. A combination of the exhaustive scan-ring test at the chip level, coupled with the certification of the clock operation and scan connections at the system level, provided the necessary confidence that the hardware could be initialized via scanning.

Logic built-in self-test (LBIST)

Although the concept of logic built-in self-test (LBIST) has been around for many S/390 generations, none of the

prior verification environments afforded the opportunity to test the logic design in a system configuration prior to tape-out. Not only did the two-cycle model permit the LBIST logic to be exercised to the point of generating a full-system signature, but it assisted greatly in ensuring that the clock control logic could sequence the processor and L2 chips through an LBIST operation.

Another application of the LBIST verification was to use the two-cycle model to cross-check the test-pattern generation model. This proved to be very beneficial, since many pieces of the test-pattern generation methodology were developed during the G4 project. With the rising costs of processing test data on wafers at the foundries, efforts to minimize the generation of erroneous test data translated to real savings in development expense. This was achieved by generating LBIST signatures on the two-cycle model in a system environment, and comparing them with similarly generated signatures on the test-pattern generation model. Initially, a goal of attaining ten matching signatures was established. As a result of this endeavor, errors were discovered in the test pattern generation tools, as well as in the macros used to represent nonstandard components such as arrays, and in the custom latches in the processor. Upon rectifying these modeling discrepancies, an environment was created to permit golden LBIST MISR predictions for the G4 chip set.

Rounding out the LBIST verification was another type of test known as an *ac-skewed load* test, in which the L1 and L2 portions of a latch are loaded with orthogonal values. The trigger clock is applied first, and is followed on the next cycle by the latch clock. This type of test is normally used with real hardware to ensure that the L2 can perform the required transition during the C2 cycle and can physically deliver the new state to the next L1 in the scan ring on the following C1 cycle. Since the two-cycle model ignores physical timing constraints, it plays no part in verifying the ac functionality of the machine. However, it proved valuable in detecting modeling problems with the latches and arrays. These escaped functional simulation, because functional simulation always starts with matching data in the L1 and L2 portions of a latch. This exercise further assisted in instilling the proper confidence level in the behavioral representation of many of the fundamental design components.

Clock verification

Errors in the design of clock logic often result in hardware engineering changes, yet prior S/390 machines neglected to simulate the gate-level design of the clock logic at the system level. Although it is true that certain components of the clock circuitry, such as phase-locked loops, cannot be simulated in a cycle simulator, much of the design in the complex S/390 clock environment lends itself well to two-cycle simulation.

In the G4 machine, a multitude of test plans were developed to exercise different facets of the clock logic. The design under test covered the clock chip in the system as well as the on-product clock-generation (OPCG) logic contained in the processor and L2 chips. The following are examples of the types of tests run:

- Cycling the clock chip through the various modes such as LBIST, ABIST, and SOCE (Stop On Count or Error), and ensuring that the proper clock sequences were generated. This test uncovered a design problem in the clock chip in which the resulting number of clock pulses generated for a SOCE command were incorrect.
- The serial interface between the clock chip and the processor was monitored for proper operation.
- Various types of clock stop/start sequences were run to ensure that all of the chips in the system stopped and started in tandem. One of these tests uncovered a problem in which the arrays could not be written using chip single-cycle mode. Fortunately the arrays worked in system single-cycle mode, which allowed the specialized millicode to be rewritten without the need for a hardware engineering change.
- Although the phase-locked loop itself could not be exercised, all of the logic required to scan-initialize it to the various run-time modes was verified.

The two-cycle model also proved beneficial in isolating a design problem first discovered on the engineering test floor. The model proved to be accurate enough to uncover a problem in an array design in which a certain start/stop sequence of the clocks resulted in an extra write operation taking place in the array. This eventually led to a new type of clock-verification test for future tape-outs.

Design data verification

An integral part of the G4 verification plan was the use of simulation to verify as much of the firmware design data as possible. This was especially important because development of much of the code was isolated from the hardware design. In addition, pieces of the engineering hardware test plan depended heavily on the ability to load specialized millicode into the caches via the design data. Therefore, several types of special-purpose two-cycle models were built to remove as many design data errors as possible prior to receiving engineering test hardware.

One model was established solely to test the specialized millicode which was loaded into the L2 cache. Although this code was locally developed, it had to interface with S/390-level initial millicode load (IML) code. Since this test was designed to run on a nonstandard hardware configuration, it was imperative to ensure that the existing S/390 code could support this type of engineering debug exercise. The next test simulated the array characterization

data, which describe the manner in which the array control logic must be scanned in order to perform atomic read and write operations on the arrays. The process for creating these data was prone to mistakes, so the two-cycle model served as a qualification tool for the array characterization data. Several problems were encountered in which the characterization data were incorrectly initializing the array control logic.

Perhaps the most impressive use of the two-cycle model at a system level was for verification of the service element software. In this environment, the actual service element code was attached to the two-cycle model through a special interface called the hypervisor. The role of the hypervisor was to intercept the signals between the service element and hardware interface and to replace them with the corresponding simulation application program interface (SIMAPI) commands to load and read the appropriate registers in the model. Use of the two-cycle model in this fashion enables complex S/390 software to be debugged and tested to gain preliminary confidence prior to enlisting expensive test-floor machine time to run the code.

Correlation with single-cycle simulation models

One of the most powerful weapons in the G4 verification methodology is the extensive use of cycle simulation at various levels of the design ranging from the functional units comprising the processor all the way up to symmetrical multiprocessor (SMP) systems with multiple processors. Although the most accurate form of simulation would be to use a two-cycle model exclusively, this is simply impractical. To begin with, the model requires the physical design to reach a certain state of completeness, which is counterproductive to rapidly advancing the logic design. Second, a two-cycle model runs twice as slowly as a single-cycle model, and the complexity of the G4 simulation test plan requires the faster single-cycle model. In addition, the models focus primarily on mainline system functions. This translates into models utilizing numerous black boxes which sacrifice slower gate-level accuracy for faster high-level descriptions. These high-level macros are often written early in the design cycle, while the details of the black boxes are still being implemented. In this type of environment, the risk exists that a high-level description was based on an inaccurate assumption, or that a design change later in the cycle was never propagated back into the high-level macro.

To minimize this risk, the two-cycle model was used to cross-check the single-cycle model. In the case of the L2, the random simulation environment was enhanced such that the same code could exercise either type of model. This permitted the simulation team to select a seed from a single-cycle simulation run and exercise it on both models. In addition, several functional simulation test cases would be run against both models to ensure that the physical design exhibited the correct functional behavior. Performing both of these activities prior to tape-out helped instill confidence in the congruence between the two simulation models. This, in turn, assisted in validating the millions of single-cycle simulation results. In the case of the processor, correlation with the single-cycle model was aided by an environment centered around AVPs. Thus, it was simply a matter of running the same AVPs on both models and ensuring that the results were identical.

Results and conclusions

Perhaps the greatest testimonial bestowed on the two-cycle simulation effort is the integral role it played in ensuring that the circuit design functioned in a manner equivalent to that of the corresponding simulation models. Evidence of this became apparent with the first hardware release, which powered on in an unprecedented fashion by allowing the test team to exercise all supported mainline function. The contribution associated with two-cycle simulation is more apparent when contrasted with the test-floor experience on past S/390 machines. For example, in an earlier project a coding problem in a checking tool permitted a number of chips to be released with nonfunctional scan rings. Although the functional logic was fine, this error required an emergency hardware release to allow engineering test to continue. In another escape, the scanning function was inhibited by incorrect implementation of tie-ups and tie-downs on some of the physical books. During the G4 project, these types of problems were detectable with two-cycle simulation.

This is not to say that two-cycle simulation is a guarantee to a successful power-on. As with any simulation endeavor, the results are only as good as the model and the type of test vectors exercised. In the initial G4 hardware, a problem was encountered on the test floor with respect to a window condition involving the scanning operation. The two-cycle simulation model verified the scanning operation using only a single test case in which the normal system clocks were always stopped during the *trigger* phase. However, a design problem was found in some of the arrays such that interrupting the system clocks to perform scanning during the *latch* phase could result in a one-cycle window where the output data came from the wrong address. The problem was accurately reproduced and debugged in a two-cycle simulation environment, and the scan-verification test was enhanced to prevent this type of error on subsequent releases.

Many advances have been made in recent history with respect to large-scale simulation. Some, such as event simulators, help close the accuracy gap by allowing the designer to simulate a behavioral representation containing logical and physical design characteristics.

Other advances, such as cycle simulation, focus on speed and performance by assuming that certain physical constraints are met. Finally, advances in the area of cosimulation permit design components accustomed to event simulation to interact with components that benefit from cycle simulation. However, two-cycle simulation provides a new dimension to the verification world by affording a method to exploit the performance of regular single-cycle simulation and augment it with system-level operations governed by the intricacies of the physical design.

Acknowledgments

None of the disclosed advances in simulation verification would have been possible without the contribution of many individuals. The author would especially like to acknowledge both the abundant contributions of Rick Seigler and Daniel Beece to the development of the two-cycle simulation model and their assistance in developing many of these verification techniques. The following people also deserve mention for their outstanding contributions in the areas of two-cycle verification as well as for providing valuable information for this paper: Bruce Wile, William Huott, Timothy McNamara, Timothy Koprowski, Steven Licker, Edward Kaminski, Jr., Thomas Gilbert, Ken Shepard, Thomas Foote, Bryan Robbins, Tim Charest, William Dachtera, Cara Hanson, and Michael Mullen. Finally, a special thanks to Bing-Lun Chu for his thorough review and valuable contributions to this paper.

*Trademark or registered trademark of International Business Machines Corporation.

References

1. "VHSIC Hardware Description Language," IEEE Standard 1076, IEEE Standards Board, 345 E. 47th St., New York, NY 10017, 1994.
2. K. L. Shepard, S. M. Carey, E. K. Cho, B. W. Curran, R. F. Hatch, D. E. Hoffman, S. A. McCabe, G. A. Northrop, and R. Seigler, "Design Methodology for the S/390 Parallel Enterprise Server G4 Microprocessor," *IBM J. Res. Develop.* **41**, 515–547 (1997, this issue).

Received December 4, 1996; accepted for publication May 5, 1997

Gary A. Van Huben *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (vanhuben@vnet.ibm.com).* Mr. Van Huben joined IBM in 1986 and is currently an Advisory Engineer. He has held various design assignments involving the S/390 I/O subsystem, central processor, and storage controller. His most recent assignment was logic design for the S/390 G4 L2 cache and dataflow controllers. In addition to logic design, Mr. Van Huben has also worked on various design processes and methodologies within the S/390 organization. He served as technical team leader for the data management and design control process governing the S/390 ES/9000 product line. Mr. Van Huben received his B.S. in electrical and computer engineering from Clarkson University in 1986; he currently holds two IBM Invention Achievement plateau awards for nine patent filings.