WHITE PAPER

cadence®

how big can you dream?™

VERIFYING THE CRAY SV1 SUPERCOMPUTER MEMORY ARBITER USING MODEL CHECKING

TABLE OF CONTENTS

TABLE OF FIGURES

INTRODUCTION

Verifying the memory system of a vector supercomputer presents a number of challenges. This paper discusses some of the verification hurdles that were encountered during the verification of the Cray SV1 supercomputer. In particular, we present the challenges encountered when verifying the memory arbiter. This page describes the difficulties encountered when verifying the logic through conventional simulation and why it was not sufficient in this case. The chip containing the arbiter logic was first fabricated without doing any model checking. Several problems were discovered following fabrication, leading to the decision to apply model checking for the respin. In this paper, the Cray verification team, led by Jon Bagge, details the results of using model checking on the arbiter logic, describes the bugs which were found, and finally shows that the respin was a success—due in part to the application of model checking.

They begin by presenting an overview of the SV1 memory system design. Then a historical perspective and motivation section provides some context for our approach. Finally they describe the methodology used and the results that followed.

MEMORY SYSTEM DESIGN

The final stage of the memory system design contains an arbiter, which selects one of five different input ports for access to a single output port. The large number of end cases associated with the arbitration scheme made the design difficult to verify.

One of the input ports is special and always has priority over the other four. The remaining four input ports can have a variety of different reference types on them, and the arbitration scheme varies depending on what is on each port. The references coming in on all five ports may be destined for one of two memory sections. However, the output port can never send two references to the same section on two consecutive clocks. This restriction results in many more end cases in the priority scheme. An input port reference may have priority, but if it is destined to the same section as a reference that left the output port on the previous clock, it must be blocked—and the next highest priority input is selected for output.

The arbiter also contains a counter for each section. The arbiter is incremented each time it sends a reference to the memory section and decremented each time the section responds with a done. When the count hits a predetermined value, the arbiter has to stop sending references to the section until the section responds with a done. *See Figure 1* for a basic block diagram of the design.

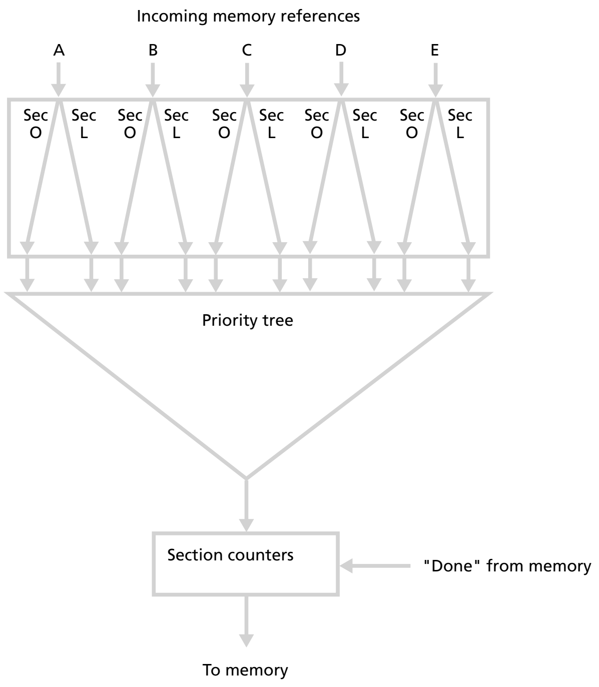

Figure 1. Block diagram of the memory arbiter

MOTIVATION AND HISTORICAL PERSPECTIVE

Historically, the team conducting this study relied on the extensive use of conventional simulation to verify their vector designs. This left them with a wealth of test code to exercise every aspect of new vector supercomputer systems. The Cray SV1 verification project started out in the same manner, where the team ran as many simulation cycles as required until it seemed that the design was fully verified.

The verification environment includes a combination of event-driven and special-purpose cycle-based simulators. The vector processor-based cycle simulator used for the system simulation of the SV1, processed 90 parallel test streams. Each stream included four processor chips with four memory arbiters. The team ran system simulation for 17 weeks, around the clock, at 10 cycles per second. The result was approximately 148 billion simulated arbiter cycles.

From the start of the verification project, it was apparent that the memory arbitration unit would be difficult to verify. Because of the complexity of this logic block, changes made to fix a problem resulted in the creation of other problems. This iterative debug process continued until the team thought they finally had it right. After all of the test code was running cleanly for approximately one month, the chip was released and built. When the prototype hardware came back from fabrication, several bugs were discovered in the arbiter. These resulted in a variety of memory ordering problems that required a re-spin of the processor chip. Conventional simulation had failed, and they needed a better way of verifying the memory arbiter to ensure success for the re-spin.

METHODOLOGY

The subtle interactions and dependencies between sequences of memory references made the SV1 memory arbiter an enormous verification challenge. To mitigate the risk of introducing other latent design bugs, the team decided to apply formal methods using Cadence FormalCheck Model Checker—a product of Bell Labs Design Automation at the time of this study.

After a brief learning period of less than a week, the team understood the basic concepts of model checking and how to use FormalCheck. They were then able to begin writing constraints and properties for the queries needed to verify the SV1 arbiter. Within a query, properties represented the behavior to be tested while constraints were used to express the operating environment for the design block. The biggest challenge was gaining an adequate understanding of the logic connected to the arbiter block to permit constraint definition. The creation of the actual constraints, properties, and queries within FormalCheck was a simple process.

The team began by writing a very simple query, which checked that each incoming memory reference would eventually receive a reply. Then, by developing several simple queries of this type, they were able to validate most of our constraints. The team then created several queries that were designed to fail. For example, one of them specified that if a reference comes in, it never goes out. This was done to make them comfortable with FormalCheck as a debugging tool and to help them understand the design.

After the team had written several simple queries, they worked their way up to more complex and interesting ones. During this process, most of their time was spent learning about the design they were testing. The team thought they knew how the design worked, but they found that there were behaviors they didn't fully understand or that were not reflected in the design specification. They spent approximately six weeks working on failing queries. Bugs in the design caused some of these failures. Many of the failures reflected problems with the constraints. After the six weeks they had a total of 21 constraints, which they were confident correctly described the operating environment of the design.

The team wanted to convince themselves that these constraints were correct and that they were not over-constraining the design. They accomplished this by creating a Pearl® script, which mapped the constraints into assertion checkers for use with conventional simulations. The assertion checkers monitored the logic during the simulation runs using all of their code to test for possible violations of the constraints. They did find one violation, which turned out to be a bug that was also found the same day by a FormalCheck query.

After using the tool for a total of seven weeks, the team was only able to enable four of the five input ports. They found that with all five input ports enabled, the queries would run out of memory. To improve verification coverage they had five versions of every query—each with a different input port disabled. This created a total of 230 queries—each running for an average of four hours. They wanted to enable all five ports to achieve full coverage and to reduce the query run time by a factor of five. With support from Cadence, the team discovered an advanced feature called clock abstraction. Since their design only used the positive edge of the clock, they were able to abstract away the clock and have FormalCheck assume that the clock was always on a positive edge. After removing the clock, they were able to enable all five ports and get a complete verification of the design using 46 queries which ran for an average of five hours each. Since completing the project, clock abstraction has been made an automatic feature in the latest release of FormalCheck.

When all five ports of the design were enabled, the team discovered additional problems in the design and with their queries. After three more weeks of work, all queries were running to completion and the team had achieved what they felt to be complete coverage of the design. They fully verified the SV1 arbiter in ten weeks using a single engineer with model checking versus the first spin of the part where they had three engineers simulating the design for several months without achieving anywhere near full coverage. They estimate the effort as 400 man-hours for model checking and 2000 man-hours for simulation, respectively.

The next generation of the Cray SV1 will reuse a large portion of the memory arbitration unit, which is logically equivalent to the original design. Using the same queries they had already written, the team verified the new logic in less than a week and found one bug in the process. Because of their improved understanding of the design, they were able to find three additional bugs before running the first query—simply by looking at the design.

RESULTS

The bugs found in the arbiter logic by FormalCheck can be classified into four different categories:

- Bugs that would never have been found through our conventional simulation
- Bugs they might not have found through simulation
- Bugs they may have found if they had run simulation long enough
- Bugs they would definitely have found through conventional simulation

Table 1 provides a breakdown of the bugs by type.

Three of the problems found are classified as Type 1 bugs. Two of these were performance-related problems. If one of these bugs had been missed, it would have had a small performance impact on the design. Another bug was related to extra latches that Cadence Design Systems, Inc. 10/99 Page 5 were causing unnecessary special case handling within the design. The team did not spend the time it would have required to verify if the latches would have ultimately caused a logic problem or not.

Type 1 bug, Would not find with simulation, 3 total

2—Performance related

1—Extra latches, added complexity

Type 2 bug, Not likely to find with simulation, 2 total

1—Extreme end case, may not occur in actual hardware

1—Ordering problem, would have caused a design respin

Type 3 bug, Might find with simulation, 3 total

3—Ordering problem, would have caused a design respin

Type 4 bug, Would find with simulation, 1 total

1—Basic functionality bug

Table 1. Breakdown of bugs discovered with FormalCheck

Two of the problems found are classified as Type 2 bugs. One of these was such an extreme end-case that, if it hadn't been found, it might have only caused a few unexplained interrupts in the field. The other Type 2 bug would have caused a re-spin, after what would have been an extended troubleshooting period, if it had been left in the design.

Three of the problems found are classified as Type 3 bugs. These were all problems where the wrong input port was selected for output. If the team had run the correct diagnostic for an extended time period, these problems may have been found through conventional simulation. If any of them had been missed, it would have caused a re-spin. The last problem is a Type 4 bug. This was a basic functionality bug that would have failed quickly under conventional simulation.

Unexpectedly, the team gained a much clearer understanding of the arbiter logic itself. Even though they thought they had a good understanding of the logic, they were surprised to find out how much they did not know. They often discovered, after tracking down failing queries, that their constraints were wrong and they had made incorrect assumptions about the logic feeding the arbiter.

SUMMARY AND CONCLUSIONS

The respin of the processor chip was released and built. The part came back and has run to specification. There appear to be no problems with the arbiter that gave the team so many problems before using FormalCheck to verify it.

The decision to use FormalCheck on this highly-specialized piece of the processor design was a good one. It saved the team from having to do another costly and time-consuming re-spin of the processor chip. It has also allowed them to verify the same logic on the next-generation system in less than a week, obtaining complete functional coverage.

cādence®

Cadence Design Systems, Inc.

Corporate Headquarters
555 River Oaks Parkway
San Jose, CA 95134
800.746.6223
408.943.1234
www.cadence.com

Stock #3324 09/01