Innovative Verification Techniques Used in the Implementation of a Third-Generation 1.1GHz 64b Microprocessor

Victor Melamed, Harry Stuimer, David Wilkins, Lawrence Chang, Kevin Normoyle, and Sutikshan Bhutani

Sun Microsystems Inc., 901 San Antonio Road Palo Alto, CA, 94303-4900, USA. Mail Stop: SUN03-315,
{victor.melamed,harry.stuimer, david.wilkins, lawrence.chang, fred.lowe,kevin.normoyle, sutikshan.bhutani}@sun.com

Abstract. This paper presents an innovative tool used during the verification of the UltraSPARC #IIIi (TM) processor. UltraSparc #IIIi operates in a multi-processor environment. Verifying the robustness of the cache coherency maintaining parts of the design was one of the main challenges facing the functional verification team. The team adopted a combination of standard, "classic" techniques and methodologies, as well as some new innovative approaches. This mixture of old and new led to a well-balanced, robust verification flow which enabled finding the majority of the design problems (bugs) early in the pre-silicon stage of the project. This paper discusses an internal tool (Sniper) (patent pending) which increases the processor bus activity in a way which would uncover subtle coherency problems.

1 Architecture Overview

The UltraSPARC #IIIi processor is a third generation 64b 4-instruction issue SPARC(TM) RISC processor which supports high-end desktop workstations and workgroup servers with focus on higher system integration and cost reduction. The chip operates at 1.1GHz to 1.4GHz. The processor core uses a 14-stage pipeline described in [1,2,3] that supports the concurrent launch of up to six instructions which can consist of 2 integer operations, 2 floating point operations, 1 memory operation and 1 control transfer instruction.[5] On-chip, level 1 caches include 64KB 4-way data cache, a 32KB 4-way instruction cache, a 2KB 4-way data prefetch cache, and a 2KB 4-way write cache. The instruction and data caches have data parity protection. The design includes a 1MB on-chip, (64B line size) level 2 cache that is used for both instruction and data caching. It implements a pseudo-random cache line replacement algorithm. The level 2 cache is a writeback, write-allocate cache, supporting the MOESI coherency protocol. [5] The proprietary 200MHz, 128 bit system interface (JBUS) enables processors and other bus agents to communicate via the shared address and data bus with 3.2GB/s maximum bandwidth.[5] The on-chip memory controller

D.A. Peled and M.Y. Vardi (Eds.): FORTE 2002, LNCS 2529, pp. 360–363, 2002.

supports 133MHz double data rate (DDR1) SDRAM and provides 4.2GB/s per processor off- chip memory bandwidth. A JBUS to PCI bridge was designed simultaneously with the processor and provides access to IO devices which reside on a PCI bus.

2 Functional Verification Strategy

The team adopted a combination of standard, "classic" techniques and methodologies, as well as some new innovative techniques. This combination led to a well-balanced, robust verification flow which enabled catching the majority of the design problems (bugs) in the pre-silicon stage and increased the productivity of the verification team. The UltraSPARC #IIIi processor is binary compatible with previous SPARC processors, so the first natural step in the verification process was to port a significant set of existing Sparc assembly tests which were used to verify previous processors. Many new test cases were written manually but there was an emphasis on the usage of random test generators. Monitors were written to check the correctness of the internal and external interface protocols, as well as to detect internal conditions which were considered errors. This paper will focus on a tool (Sniper) used in the verification process to increase the processor bus activity in a way which would uncover subtle coherency problems. It is important to note that Sniper would not be effective without the strong foundation provided by the overall verification environment.

3 Sniper

Early on in the project the team realized the need for a programmable, high level bus functional model of UltraSparc #IIIi (TM) that could be instanced in our simulation environment (in place of multiple instances of the verilog model of the CPU) to generate system bus traffic. This high-level model, written in C++, was called a Simulated JBUS Master (SJM), and models a JBUS compliant processor with a MOESI coherent cache. A simple command language controls the SJM's behavior to provide a very flexible stimulus generator for our verification environment.

Sniper is a tool which was built on top of SJM. The tool works by keeping a history table of the transactions on the processor bus (JBUS) and occasionally issuing transactions to increase the stress on the bus. The transactions issued by Sniper use addresses which are closely correlated to the addresses which appeared on the JBUS previously. They are also non-destructive, meaning that the Sniper generated transactions never modify the data that they access.

Sniper can be tuned through configuration files. The main configuration parameters are: frequency of transaction injection, length of history table, types of transactions that Sniper is allowed to issue and address generation algorithms based on the address history.

Typically, Sniper chooses an address from the address history table (either randomly or the N-th oldest address) and then "shoots" in one of two ways. In

a "surrounding" mode Sniper randomly generates an address which is close to the chosen one and can be bigger, equal or smaller. In a "prefetching" mode the address can only be equal or bigger. In both cases the maximum distance between the chosen address and the sniper address can be configured.

Sniper knows which JBUS agents generate the addresses that appear on the bus. It does not enqueue the addresses that it generated itself in the address history table. Enqueueing Sniper's own addresses would create a positive feedback loop and would decrease the efficiency of the tool.

Sniper is essentially a bus noise generator where the noise can be tuned to have a high correlation with the signal. The typical behavior of Sniper will result in lines being "snooped away" from their normal place of residence. Thus, typical Sniper-generated bus activity will include one Sniper-generated transaction to "steal" the line into the cache of a Simulated JBUS Model and then one more transaction where the user of the line requests the line (back). Obviously this increases the traffic on the processor bus and stresses the processor's internal queues and FIFOs, but more importantly, the same line is going back and forth between processors, thus testing logic which may be stalling or bypassing information based on an address match.

Almost any test case which passes without Sniper enabled should pass with Sniper in use since it does not modify any data. Only the tests which perform diagnostic reads from the cache and expect a cache line to be in a particular state are an exception and cannot run with Sniper, and test cases such as these are rare.

The concept was extended to the PCI bus (the system's IO bus) in our environment with only slight modifications. The PCI bus does not support cache coherency, but the idea of generating transactions based on the history of the bus is still valid.

The next extension of Sniper is interesting for bus-to-bus bridges which are historically very difficult to verify. In a configuration where a bridge chip connects bus A and B, Sniper may monitor bus A and issue transactions on bus B, and vice versa. This idea has not yet been implemented.

Sniper does not have to be limited to monitoring bus interfaces. It can also be extended to monitor internal chip signals and interfaces. This would add a predictor property to Sniper since it would know in advance what transactions the processor is about to present to the bus.

A last idea for further Sniper development would be to generate an address as a function of several entries in the history table. Some linear regression curve functions are the obvious first choice here.

4 Some Results, Conclusions

A typical bug caught by Sniper can be described with the following statement: "This is a corner case of a corner case. The test is aligning two transactions next to each other, now comes Sniper and adds a third one". The first family of bugs that Sniper targeted well were corner cases where transactions with related

addresses have to appear almost simultaneously. Other tools scored poorly in identifying those types of bugs. A "typical" bug is often found by many tests and tools independently. The bugs found by Sniper however, were extremely hard to duplicate without it, even after the bug was already known and another test or tool was tweaked to attempt to address it.

The other family of bugs that Sniper uncovered were related to overflowing internal processor queues. Since the tool generates a fair number of transactions on the bus, it helped fill up the bus-related queues in the design and thus tested the conditions where these queues are full or almost full. Other tools also scored well locating this type of bugs. Only 2–3% of all logic bugs found on UltraSparc #IIIi could be credited to Sniper. One reason for this is that the tool was deployed when the design was already in a fairly mature state. However, most of these bugs were very subtle and thus would have been very difficult to find without Sniper.

References

1. Heald R. et al "A 3rd generation Sparc V9 64-b Microprocessor" IEEE JSSC, pp. 1526–1538, Nov. 2000
2. Lauterbach G. et al "UltraSPARC-III: a 3rd generation 64b SPARC Microprocessor", ISSCC Digest of Technical Papers, pp 410–411, Feb. 2000.
3. Heald R et al, "Implementation of a 3rd Generation SPARC V9 64b Microprocessor", ISSCC Digest of Technical Papers, pp 412–413, Feb. 2000
4. Normoyle K. "Introducing the UltraSPARC(TM)-IIIi Microprocessor", Microprocessor Forum, Oct. 2001
5. George Konstadinidis et al, "Implementation of a Third generation 1.1 64b Microprocessor", ISSCC2002.