Design Defect Diagnosis in SPARC T1 Processor

BrijMohan[1], Lekha Pankaj[2], K, Nutan Hegde[3]
[1,2,3]Assistant Professor, Dept. of EC,
[1]GEC Kozhikode,
[2]AWHEC Calicut
[3]KMCTCEW Calicut

Abstract

Here Verification has indisputably become the primary challenge today with recent industry studies estimating that half of all chips manufactured require one or more re-spins. This work addresses the problem of verifying the correctness of pre-silicon models of a multithreaded multicored processor (SPARC T1) and aims in replicating an industry level problem and solution approach for verifying a real processor design (state-of-the-art). The approach in classifying general type of bugs that can get into the EXU unit of SPARC T1 processor design and introducing design defects in the RTL. Several defective models were developed covering the entire functional blocks in the EXU and several coverage models for the test cases run on the defective design were also developed. This work tends to look back at the fact that verification is a process that is never truly complete. We understand that designs are error-prone and so, the objective of verification is to detect the errors. Yet, no one can really prove that the design is error-free.

Keywords

SPARC T1 Processor, Multicore, verification

1. Introduction

The open sourcing of hardware design of SPARC T1 has enabled challenges around multi-threading and 64-bit hardware design concepts to be explored more freely and openly, and beneficial innovations be achieved. Functional verification of modern processors and complex ASIC designs is a challenging task. The time required to find the exact source of an error in complex designs represents a significant part of the verification process. Design development time is largely influenced by the number of latent defects in the design which in turn is often a function of the completeness of the architectural specification, and the complexity of its implementation. The quality of the verification process is reflected both in the fraction of latent defects that are detected (test coverage) and the mean-time-to-detect. These are the factors that control the cost and quality of designs [5]. Today, design bugs are treated with ad-hoc heuristic techniques that seek to avoid the occurrence of design bugs through software and hardware configuration changes.

The test space of processor design is so huge that it is very difficult to completely specify it. As a result, directed testing (special hand-coded test cases) alone would not be sufficient to find all the design defects. On the other hand, doing exhaustive random testing is not realistic, since it would require a tremendous amount of simulation resources. Hence a dependable number of directed testing along with random testing to catch the corner cases would do proper justice to the verification program. This work mostly tends to look into the direct testing aspects.

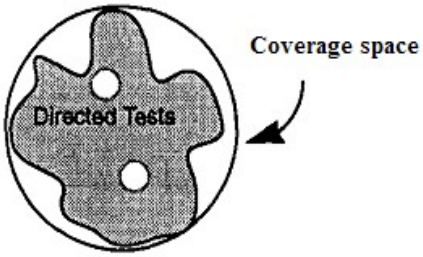

Fig 1. Test Coverage

Understanding the design micro architecture down to the RTL code level for injecting defects and developing new tests (white box approach), promises full visibility and controllability of internal structures and implementations of the design being verified.

2. Implementation of Defect Model

The main approach of this paper is to develop several design defective models at the RTL level and target the bugs through directed testcases .The testcases are developed in SPARC assembly language based on the SPARC V9 architecture. The simulations are run at chip level without the reference model. A regression suite is developed which could be further used to customize new designs.

To develop a defect model, the study focuses on the Verilog RTL source code of the OpenSPARC T1 chip-multiprocessor, the open source version of Sun's commercial UltraSPARC T1 (Niagara) chip-multiprocessor. The RTL level is considered to be very close to the actual hardware implementation. The only design phases separating the RTL level with the actual hardware implementation are logic synthesis, which generates the design's gate-level netlist and place-and-route, which creates the transistor-level layout of the netlist. Therefore, the direct relation between the RTL level and the underlying implementation provides an adequate level of detail that allows the extraction of low-level design bug characteristics.

3. Classification of Design Bugs

Generally bugs can be classified under several categories, some of them being:

i. Logical design bugs: These are caused due to erroneous logic in combinational circuits.

ii. Algorithmic Design Bugs: These are caused by bugs in algorithmic implementation of the design. These design bugs exhibit algorithmic deviations from the design specification and they usually require major modifications to be fixed.

iii. Timing bugs: These bugs are associated with timing correctness of the implementation.

iv. Wrong signal source.

v. Wrong operator.

vi. Incorrect control logic.

vii. Incorrect connections Units

4. Bug Implementation in execution unit

The verilog coding styles of the processor RTL is familiarized and the various blocks of the EXU were identified to inject bugs. Since most of the design files have very large number of instantiations, the simulation of some design files, stating the bugs introduced, is done in Xilinx.

A. Injecting bugs in the ALU unit

The carry generated from the 32nd bit while adding two 64 bit numbers is not considered, Omission of carry generated in the adder logic while adding two 64 bit data. Adder output is implemented correctly only for odd value data in register rs1 (for data1).The least significant bit of rs1 register (for data1) is always stuck to one. Wrong implementation of AND & XOR logic resulting in erroneous implementation of the ALU block. The subtract logic of the add sub unit gives wrong results. Erroneous logics implemented for NAND, XNOR and NOR operations for the ALU block. Wrong logic for 64 bit zero comparator. This results in incorrect logic for several units like

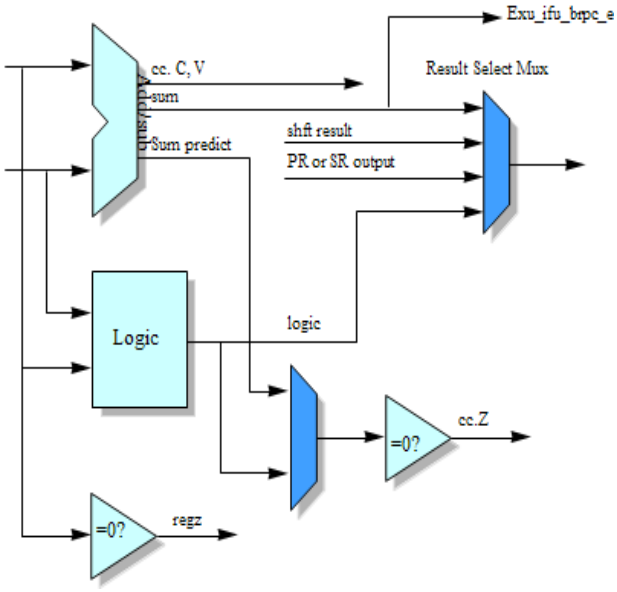

Fig 2. ALU Block

LSU, IFU which checks the address and data value. Error when performing logical or of consequence bits of a 32 bit data resulting in wrong calculation of aluzcmp64. The sum predict block sets the zero condition code register for nonzero results.

B. Injecting bugs in the shifter unit

The left and right shifter units are cross connected, resulting in erroneous results. The left shifting of operand bits by 4, 8, and 16 bits ,results in filling by '1' instead of zero filling.

The SPARC V9 architecture and the assembly language are

familiarized to write test cases. Since the design is a white box, several directed test cases are developed to target the bugs.

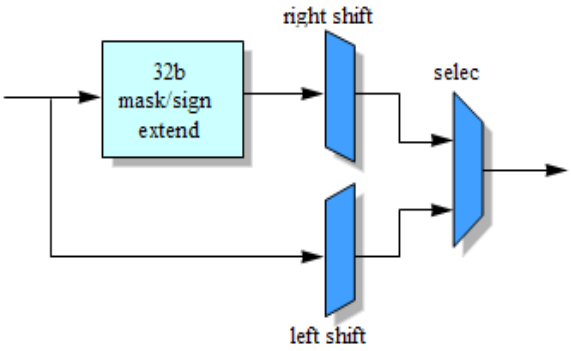

Fig 3. Shifter Block

Long simulations are conducted by targeting the bug injected models with individual testcases at chip level and the results are monitored through log files. Simultaneously the tests are run on the clean model to check the reliability of the tests. The right test cases that hit the bug give a bad trap (which indicates error in test results), while the remaining gives a good trap. All the test cases hit good trap for the clean model. The verification process is improved by creating diaglists to run several testcases together and hence a regression suite is created with the testscases that are developed.

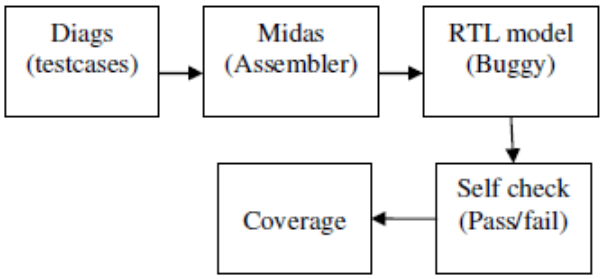

Fig 4. Simulation flow graph

Midas is the diag assembler for SPARC and its main function is to build and link the diag. This calls the assembler to create an executable from the assembly language source of the test. The resulting executable contains some simple reset

code, trap tables, as well as the main diagnostic code. The sims program then takes this executable and creates a memory image file from it. All the code and data sections from the executable are placed into the memory image file. Then the required page table entries and translation storage buffers for virtual memory are added.

It is observed that several testcases could hit the bad trap when run on the buggy model, while the test vectors that could not catch the bug ended in good trap. Several diags are run simultaneously to create a regression suite. The status.log report gives the diag status. The log files shows exactly how the diag is disassembled including the reset, traps and main section of the program.

Status:	mytests	ALL
PASS:	0	0
FAIL:	5	5
Diag Problem:	0	0
License Problem:	0	0
MaxCycles Hit:	0	0
Socket Problem:	0	0
Timeout:	0	0
LessThreads:	0	0
Simics Problem:	0	0
Performance:	0	0
Killed By Job Q:	1	1
Unknown:	0	0
UnFinished:	0	0
flexlm error:	0	0
Diag Count:	6	6
Cycles/Sec:	651	651
K Cycles:	72	72
#Diags Used:	5	5

Fig 5. Status.log of diags run in regression

5. Simulation Timing Reports

It is observed that each testcase took different number of cycles to hit the bug. At times several testcases are unable to hit the bug within the stipulated no. of cycles and results in timed out errors. The maximum cycles are hit before the simulation enters the main section of the testcases. This is probably due to some looping at the boot code level. The testcases targeting complex blocks like div unit take more cpu cycles. Also several testcases are run with the dump on option to record the test inputs hitting the bug. The dumping is halted when the maximum cycles are hit. All the testcases hitting the EXU are set to maximum of 30,000 cycles.

Table 1. Timing Report for Various Diags

Name of Diag	Cycles	Simulation Time(sec)	Cycles/sec	Status	Environment
xu_add1.s	13627	22.58	603.5	Bad trap	core1
exu_add1.s (with dump)	30000	69.25	433.2	Bad trap	core1
Div.s	22834	53.70	425.1	Good trap	core1
Shift.s	14418	22.18	650.0	Bad trap	core1
Logical.s	10714	17.2	622.9	Bad trap	core1

6. Coverage Reports

Coverage metrics are widely used to automatically record information and analyze it to determine whether a particular test verified a specific feature. Coverage metrics can quantify the verification plan accomplished so far and can pinpoint areas that require additional verification effort.

The coverage metrics of the various test cases are monitored. It is found that the test case targeting the EXU for a particular block gives maximum coverage for its corresponding module. For example, shift.s gives more than 90% coverage for the shift block but very poor coverage for the rest of the blocks. Cumulative targeting of all test cases is conducted to improve overall coverage and it is observed that all the blocks could be covered with very high percentage. Also the overall coverage of EXU could be improved.

7. Coverage results

Code coverage provides information on how thoroughly a design has been exercised during simulation. They are used to evaluate the effectiveness of directed tests and to guide the generation of new tests. The shift.s testcase covering the EXU block is shown below. The green block indicates high coverage percentage, red indicates the lowest percentage and yellow indicates average coverage. Blocks which are not touched by the diag are left blank.

Contents of cmp_top.iop.sparc0.exu

Instance	Module	Total	Total C	Total C(%)
exu	sparc_exu			
alu	sparc_exu_alu	256	256	100.00
bypass	sparc_exu_byp	656	641	97.71
div	sparc_exu_div	634	64	10.09
ecc	sparc_exu_ecc	128	83	64.84
ecl	sparc_exu_ecl	534	290	54.31
irf	bw_r_irf			
rml	sparc_exu_rml	345	153	44.35
shft	sparc_exu_shft	207	206	99.52

Fig 4.6: Coverage results of shift.s targeting the EXU block

The merged coverage results are shown below. It is observed that all the blocks hit maximum coverage percentage.

Contents of cmp_top.iop.sparc0.exu

Instance	Module	Total	Total C	Total C(%)
exu	sparc_exu			
alu	sparc_exu_alu	256	256	100.00
bypass	sparc_exu_byp	656	656	100.00
div	sparc_exu_div	634	634	100.00
ecc	sparc_exu_ecc	128	86	67.19
ecl	sparc_exu_ecl	534	320	59.93
irf	bw_r_irf			
rml	sparc_exu_rml	345	153	44.35
shft	sparc_exu_shft	207	207	100.00

Fig 4.7: Merged Coverage results of diags targeting the EXU block

8. Conclusion and Future Scope

Verification is a process that is never truly complete. We understand that designs are error-prone and so, the objective of verification is to detect the errors. Yet, no one can really prove that the design is error-free. Verification can only show the presence of errors, not their absence. The difficulty of the design verification problem is compounded by the current processor design flow. The white box approach gives a better understanding and easiness for the verification engineers to develop testcases. In most design cycles, a design's verifiability is not explicitly considered at an early stage, when decisions are most influential, because that initial focus is exclusively on improving the design on more traditional metrics like performance, power, and area. It is thus possible for the resulting design to be very difficult to verify in the end, specifically because its verifiability was not given higher priority in the beginning. Hence verifiability should be viewed as a critical design constraint, together with other established metrics, like performance and power, from the initial stages of design. This approach will make designs more easily and thoroughly verifiable, which would both decrease the resources invested in the verification step and lead to more

robust designs.

The future scope of the project includes optimizing the hardware for 4 threads, 16 TLBs, with SPU unit to download on FPGA. Targeting bugs on the LSU and trap units can give a deeper insight of the processor memory and troubleshooting mechanism. Several on board applications can be run by building large systems by linking many CPUs on FPGA boards.

References

[1] David A.Patterson and John L.Hennessy "Computer Organization and architecture",Third edition, The Morgan Kaufmann Series in Computerarchitecture and Design, 2008.

[2] Samir Palnitkar, "Verilog HDL-A guide to Digital Design and Synthesis", First Edition, Pearson Education Asia, ISBN 81-7808-489-9, 2001.

[3] David L.Weaver, "OpenSPARC Internals, Sun Microsystems, First Edition, ISBN 978-0-557-01974-8, 2008.

[4] Darryl Gove "Solaris Application Programming", Prentice Hall, First Edition, ISBN 978-0-13-813455-6, 2008.

[5] Babu Turumella, et al. "Design Verification of a Super-scalar RlSC Processor" Twenty-Fifth International Symposium on Fault -Tolerant Computing, p.0472, IEEE, 1995

[6] Sangeetha Sudhakrishnan, Liying Su, and Jose Renau, Dept. of Computer Engineering,University of California."Processor Verification with hwBugHunt", 9th International Symposium on Quality Electronic Design, IEEE 2008.

[7] Kypros Constantinides, Onur Mutlu, Todd Austin, "Software based Online Design Bug Detection: RTL Analysis, Flexible Mechanisms, and Evaluation, Evaluation", Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-40), December 2007.

[8] D. Van Campenhout, et.al. "High-Level Design Verification of Microprocessors via Error Modeling" ACM Transactions on Design Automation of Electronic Systems, Vol. 3, No. 4, Pages 581–599, October 1998.

[9] Jacob A. Abraham, Pradip Bose, Depts. of ECE and CS, University of Texas,Austin. "Performance and Functional Verification of Microprocessors", VLSI Design, Thirteenth International Conference, Volume, Issue, Page(s):58 – 63, 2000.

[10] S. Mehta, et al. "Verification of the UltraSPARC Microprocessor" 40th IEEE Computer Society International Conference (COMPCON'95) p. 452, 1995.

[11] P. J. Tan, et. al. "Testing of UltraSPARC T1 Microprocessor and its challenges" IEEE International Test Conference, Pages 1-10, 2006 .

[12] OpenSPARC™ T1 Microarchitecture Specification " Copyright © 2006 Sun Microsystems, Inc.

[13] "OpenSPARC™ T1 Processor Design and Verification User's Guide" Copyright 2008, Sun Microsystems, Inc.

[14] David L. Weaver and Tom Germond "The SPARC Architecture Manual", Version 9. SPARC International, Inc. Santa Clara, California.

[15] OpenSPARC™ T1 Processor Datasheet Copyright 2006, Sun Microsystems, Inc.

[16] "UltraSPARC Processor Emulation Verification: Getting HW/SW right the first time "DesignCon 2007. Jai Kumar, Sun Microsystems.

[17] Ana Sonia Leon,et.al. "A Power-Efficient High-Throughput 32-Thread SPARC Processor", IEEE Journal Of Solid-State Circuits, Vol. 42, No. 1, January 2007.

[18] Shrenik Mehta, David Weaver, Jhy-Chun Wang, Ashley Saulsbury, Partha Tirumalai, Raj Prakash "Innovating with OpenSPARC" ASPLOS XII 2006 San Jose, CA Oct. 21, 2006.

[19] "OpenSPARCT1 FPGA Implementation" Microelectronics group, Sun Microsystems inc.

[20] The basics of constructing FPGA by Gina R. Smith, CEO, Owner Brown-Smith Research and Development Laboratory Inc.

[21] VCS® / VCS® MX Coverage Metrics User Guide by Synopsys.

[22] VirSim User Guide by SynopsisVirtex-5 FPGA User Guide, www.xilinx.com.

[23] www.opensparc.net

Author's Profile

Brijmohan .K has received his Masters degree in Biomedical Instrumentation in the year 2008 from Indian Institute of Technology, Bombay. At present he is working as an Assistant Professor at Government Engineering College, Kozhikode. His areas of interests are Power Electronics, Biomedical Engineering, Embedded Systems and Process Control Instrumentation.

Lekha Pankaj has received her Bachelor of Engineering degree in Electronics and Communication Engineering from MES College of Engineering, Kuttipuram in the year 1998. She has completed her M.Tech with the specialization of Electronic Design and Technology in National Institute of Technology, Calicut. At present she is working as Associate Professor in AWH Engineering College, Calicut. Her area of interest includes Embedded systems, Testing and Verification, Wireless Mobile Communication, Microprocessors and Microcontrollers.

Nutan Hegde She has completed her M.Tech with the specialization of Electronic Design and Technology in National Institute of Technology, Calicut. At present she is working as Assistant Professor in KMCT College of Engineering for Women, Calicut. Her area of interest includes Embedded systems, Control Systems, Digital System Design.