Configurable Verification Stimulus Acceleration Method Based on Multicore Processor

Guoteng Pan, Yuxing Tang, Guodong Ou, Li Luo and Qingna Yang[1,*]

[1]*College of Computer, National University of Defense Technology, Changsha 410073,Hunan, China*

Abstract: Functional verification has become a major challenge in the chip design area. To improve the efficiency of verification, it is necessary to choose appropriate verification method and tools. An important aspect of functional verification is RTL verification, simulation-based verification is main method in RTL verification. Based on FT-8 multi-core processor, we developed a configurable test stimulus acceleration method, loading the test stimulus into memory and L2 cache to speed up the processor instructions fetch, which can shorten simulation cycle and simulation time, reduce the verification cost and guaranteed the correctness of design.

Keywords: Acceleration, multicore processor, simulation-based verification, test stimulus.

1. INTRODUCTION

The increasing complexity of chip design is creating many challenges for verification. The success rate of the first chip was only about 30%, the main reason being the insufficient verification. Chip verification requires a lot of resources, accounted for 60% to 80% of the entire design resources [1]. The “Verification wall” has become the technical bottleneck of multi-core processor [2]. To improve the reliability of chip design and shorten the development cycle, selecting suitable and efficient verification tools and methods is essential [3-8].

Simulation-based verification [9] and formal verification [10] are the two main verification techniques and methods for now. The aim of the formal verification is to prove design correctness using mathematical certainty, which divided into three categories: equivalence checking, model checking and theory proven. Equivalence checking [11] is used to prove that the two design models have the same function, which is actually the most widely used formal verification technology, such as Cadence's Conformal and Synopsys's Formality. Model checking [12, 13] is used to prove that a design meets certain attributes, such as Cadence's Formal Checker. Theorem proving requires the user deep understanding of the basic logic and formal proof, which is costly and rarely used. RTL design contains a large number of latches, formal verification often encounter the problem of state space explosion, which is not suitable for large-scale designs. Simulation-based verification is used to discover errors using simulation method, which has good scalability and can be easily applied to large-scale design. However, the speed of the simulation method is not ideal. Intel used 6000 CPUs running simulation for 2 years, to verify Pentium4 processor. The total number of instructions of running is less than that of a real chip to run for 2 min [14, 15]. For multi-core processors, the simulation speed is slower, more prominent verification efficiency.

In order to improve the verification efficiency of multi-core processors, based on our designed FT series processors, we proposed a configurable verification stimulus acceleration method, loading the test stimulus into memory and L2 cache to speed up the processor instructions fetch. Through performance evaluation, the simulation time can be reduced about 15%.

2. RELATED WORK

SUN uses a multi-level method in the UltraSPARC T2 verification which provides suitable method according to different abstract level. Cadence Xtreme hardware emulator is chosen for use in system level verification which is tightly linked to software engine to perform hardware and software co-simulation [16].

The pre-silicon verification of POWER family of processors developed by IBM was divided into module level, component level, chip level and system level. IBM has invested heavily on verification technology for years. They have developed a series of independent tools which include cycle-accurate software simulation MESA, hardware accelerator AWAN, formal tools SIxthSense and RuleBase [17, 18]. A hybrid RAIM (redundant array of independent memory) method was used for the IBM zEnterprise processor verification which contained 80 configurable cores. This verification environment was based on combination of formal verification and random verification, developing configurable stimulus to improve function coverage [19].

In recent years, with rapid development of the independent processors, the domestic study and practice in processor verification is going deeper and deeper. The

*Address correspondence to this author at the College of Computer, National University of Defense Technology, Changsha 410073, Hunan, China; Tel: 0731-84573678; E-mail: gtpan.hn@gmail.com

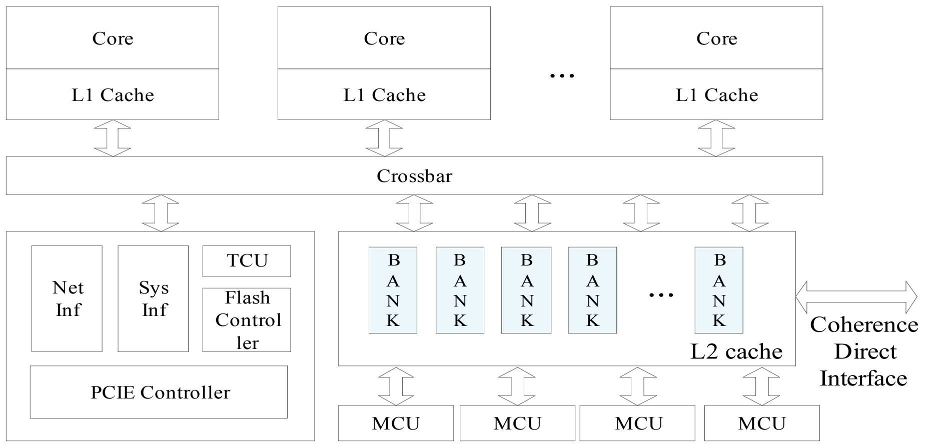

Fig. (1). Architecture of FT-8 processor.

Institute of Computer Technology of Chinese Academy of Science has made full use of both simulation and formal verification method in the verification process of the Loongson 2. Meanwhile, the verification through the FPGA prototype system has helped them to gain a fast running speed, but the debugging is a problem [20].

In the verification process of “ShenWei-1”, National High Performance IC(Shanghai) Design Center established a complete set of verification environment, accomplished real-time automatic checker method by simulating instruction level result and reusable method to generate pseudo random test based on the reference model, Furthermore, they adopted a special floating-point component verification, large-scale test acceleration verification, the FPGA physical prototype verification method and so on [21].

FT family of processors adopts hierarchical verification strategy which is divided into module level, cluster level, chip level and system level. Simulation as the leading method, and also formal verification, FPGA prototype system and hardware acceleration simulation are combined to improve the verification efficiency. Among them, the configurable verification stimulus acceleration method is presented to reduce simulation cycle.

3. ARCHITECTURE OF FT-8 PROCESSOR

The main goal of FT-8 processor is high-throughput computing in large-scale scientific computing applications. The full chip supports 64 hardware threads executing in parallel with Parallel SoC multi-thread architecture, as shown in (Fig. **1**).

FT-8 processor has eight cores, connected with on-chip cache by crossbar. Each processor core has its own integer unit, floating-point unit, data cache, instruction cache and other functional units.

In order to alleviate the increasingly prominent problem of “memory wall” in multi-core processor, FT-8 processor designed a large capacity shared on-chip L2 Cache, shared by multicore, reducing the pressure on the off-chip memory access. The L2 Cache used multi-bank techniques, divided into eight banks, which help high-speed implementation of memory on one hand, on the other hand, help to improve the system memory access bandwidth by concurrent access. Meanwhile, FT-8 implemented on-chip memory controller achieving four concurrent accesses, which designed a optimize memory access scheduling strategy, improved the external memory access bandwidth, reduced memory access latency.

In order to reduce inter-chip interconnect delay and improve the interconnect bandwidth, FT-8 processor designed specialized inter-processor directly connected interface, the interface can be achieved 2-4 way processor directly connected to a tightly coupled shared memory system.

FT-8 processor used system-on-chip design methodology, integrated the PCIE controller and memory controller on the chip, implemented north-bridge and south-bridge functions in traditional processor, further increased the system memory access bandwidth, improved the I/O expansion capabilities, and reduced the complexity of the system design.

4. SIMULATION PLATFORM AND ACCELERATION METHOD

In order to ensure FT-8 processor working properly and to achieve expected performance, we constructed a full-chip level simulation-based RTL verification platform, as shown in (Fig. **2**).

During the simulation, the test stimulus compiled through stimulus loader is loaded into the reference model (RM) and the RTL Design-Under-Test (DUT). Monitor is responsible for the information collection of the DUT and sent to the Checker which determines the function correctness of the design by comparing the key state information between RM and DUT. Memory image is generated by compiling test stimulus and is loaded into memory, accessed by the processor core issuing normal instruction fetch request. Because of relatively time-consuming of memory access, we load the contents of the memory image into L2 Cache before running test. So there will have not more L2 Cache miss in simulation, which can shorten the simulation cycle, reducing simulation time.

L2 Cache loader is implemented using C code, and is called by the simulator through PLI method. The PLI data

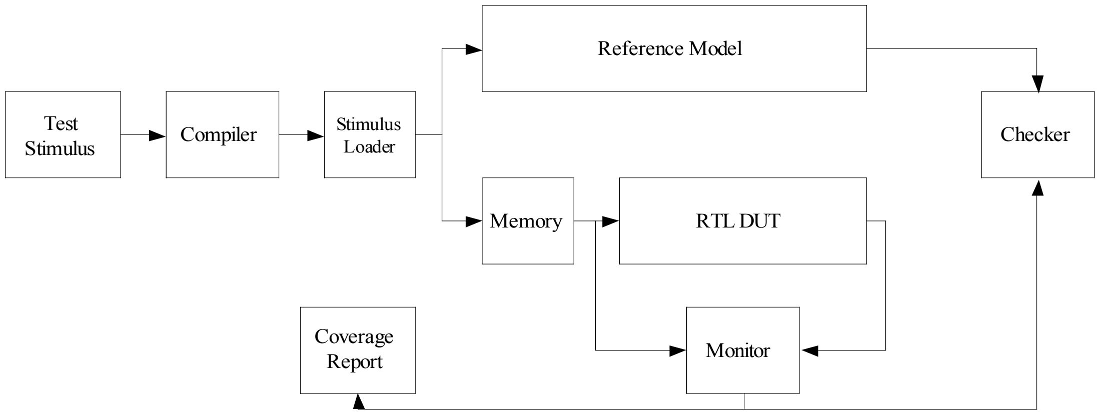

Fig. (2). FT-8 processor full-chip level simulation-based RTL verification platform.

processing flow is divided into three steps: preparing the data structure, preparing data and storing data into the Verilog array. TAG, VUAD and DATA arrays not only are the main three storage structures of L2 Cache, but also the parameters passed between the C code and Verilog design code. PLI code requires getting a pointer to directly read and write the data arrays, so the most critical of PLI function is the correct description of all three kind of storage structure in L2 cache, and passed to the corresponding PLI function.

Data array pointer is read through a function interface, and is saved into appropriate data structure. The main saved data include of the array width of TAG, VUAD and DATA array and the storage array data pointer of them. Function is called to load memory image file into tempdata_s structure before loading date into l2 cache. Tempdata_s is organized as a b-tree, each 64-byte data of the memory image file is inserted into the tree as a b-tree atomic unit. If the address belongs to the low 128GB of 1TB space, the data address is added into the address list. All VUAD bits in L2 Cache are cleared by calling l2load_clear function.

L2load is the main function of L2 cache loader, the input of l2load are the 64-bit address obtained from address linked list and the 64 bytes data obtained from tempdata_s structure by the address. First, the function calculate the L2 Bank number, L2 Cache index and TAG value in accordance with the address and the bank configuration (8Bank/4Bank or 1BANK), and splices into "selected" field in l2load_vars using part of index address of L2 Cache. Then the value of l2load.way is generated using l2load_vars.blackboard by random function. If an available way cannot be found with sixteen consecutive random, the l2load function will exit. After the value (0:15) of l2warm_vars.way is determined, the corresponding flag in blackboard is set to 1, the three-dimensional index of blackboard is [bank] [way] [selected]. Finally, l2load_tag, l2load_vuad l2load_data function are called to store the data of TAG, VUAD and DATA array into verilog data structure.

All PLI codes associated with cache are compiled too file by gcc compiler and a linkable PLI run-time library is generated by tools. For VCS simulator, a .tab file is used to describe the correspondence between the verilog call interface and PLI function. PLI library, tab file, design file list and verification environment file list are VCS command-line input. All of .o files compiled by each module in the design links with PLI library to generate a SIMV execution file.

5. EVALUATION RESULT

In order to evaluate the practical effect of our acceleration method, we select a few representative test stimuli for normal memory access, l2 cache, crossbar, interrupt handling, and memory controller error handling, and perform evaluation under single-and eight-core design configuration.

Table 1. Description of test stimulus.

Stimulus Name	Description
ld_st	normal load and store memory access
l2cache_access	L2 Cache data array access
crossbar	test packet processing of crossbar
interrupt_INT	test interrupt handler
mcu_ecc_err	test ecc error handler of memory controller

During the simulation, we monitor the state information of these tests, get these simulation cycle and simulation time based on single-core and eight-core environment, and then calculate the simulation speed. We found that comparing with the normal method, the simulation cycle and time have significantly reduced, and the simulation speed is also increased to varying degrees using the acceleration method of loading test image into L2 Cache before running test simulation, as shown in (Figs. **3** and **4**).

As can be seen from above two figures, for the single-core environment, in addition to the simulation cycle of l2access test did not change, the decrease in the that of other tests is more than 10%, the simulation time of interrupt handling is reduced by 20%. The simulation speed of interrupt handling increased by 10%, but that of mcu_ecc_err test decreased 3%.

For the eight-core environment, the simulation time of almost all tests reduced more than 15%, and that of ld_st test

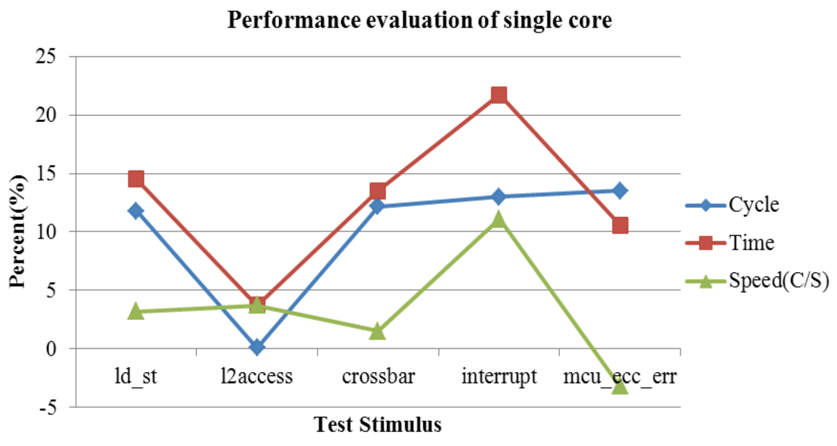

Fig. (3). Performance evaluation of single core.

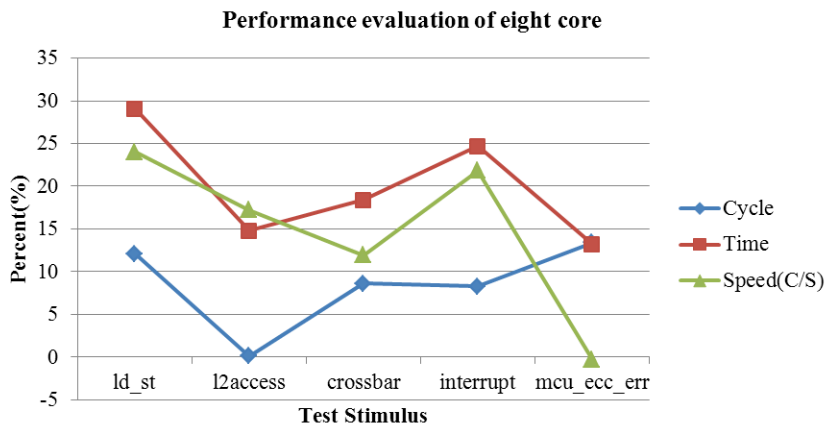

Fig. (4). Performance evaluation of eight core.

even close to 30%. The same with the single-core environment, the simulation cycle of l2 access test essentially unchanged.

6. CURRENT & FUTURE DEVELOPMENTS

With the rapid advancement of IC technology and the increasing complexity of design, the problem of verification efficiency become more and more prominent, how to effectively reduce the simulation cycle and time need carefully considered and to be solved in verification process. Although, we reduced the simulation time to a certain extent by pre-loading the test stimulus into L2 cache, with the development of the FT series processor, how to further improve the verification efficiency of a larger scale design is still a problem need for in-depth study and solve. We will put more strength on the study of coverage-driven test stimuli intelligent generated based on the perspective of the full life-cycle and high-level verification.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

This work is supported by National Natural Science Foundation of China (61202123).

REFERENCES

[1] B. Wile, J. C. Goss, and W. Roesner, "*Comprehensive Functional Verification: The Complete Industry Cycle*", Morgan Kaufmann, San Francisco, 2005.

[2] P. Patra, "On the cusp of a validation wall", *IEEE Des. Test. Comput*., vol.24, no.2, pp. 193-196, 2007.

[3] G. A. Ezer, P. Konas, J. B. Andrews, S. W. Chou, E. M. P. Long and M. A. Evans, "*Method for multiple processor system-on-a-chip hardware and software cogeneration*", U.S. Patent 8639487 B1, March 25, 2003.

[4] Z. Qi, and L. Xinhui, "*Method and device for verifying SoC (system on a chip) chips*", CN Patent 101515301 B, May 4, 2011.

[5] Z. Siliang, "*SOC (system on chip) verifying method*", CN Patent 102567149 A, July 11, 2012.

[6] L. Xinhui, "*Method and system for verifying soc chip*", WO Patent 2013016979 A1, February 7, 2013.

[7] V. Bertacco, and I. Wagner, "*System for High-Efficiency Post-Silicon Verification of a Processor*", U.S. Patent 20110087861 A1, April 14, 2011.

[8] Y. Xiaolang, F. Kewei and Z. Xin, "*Dynamic simulation platform method for embedded processor function verification*", CN Patent 100562879 C, November 25, 2009.

[9] A. S. Kamkin, and M.M. Chupilko, "Survey of modern technologies of simulation-based verification of hardware", *Program comput. Soft.+*, vol. 37, no.3, pp.147-152, 2011.

[10] R. Beers, "Pre-RTL formal verification: an intel experience", in the 45th annual Design Automation Conference, ACM New York, NY, USA, 2008, pp.806-811.

[11] C. Karfa, D. Sarkar, and C. Mandal, "Formal verification of code motion techniques using data-flow-driven equivalence checking", *in proceedings of the IEEE Annual Symposium on VLSI*, IEEE Computer Society Washington, DC, USA, 2010, pp. 428-433.

[12] G. Singh, and S. K. Shukla, "Model Checking Bluespec Specified Hardware Designs", *in proceedings of the 8th International Workshop on Microprocessor Test and Verification*, IEEE Computer Society Washington, DC, USA, 2007, pp. 39-43.

[13] L. Charvat, A. Smrcka, and T. Vojnar, "Automatic Formal Correspondence Checking of ISA and RTL Microprocessor Description", in proceedings of the 13th International Workshop on Microprocessor Test and Verification, IEEE Computer Society Washington, DC, USA, 2012, pp. 6-12.

[14] B. Bentley, "Validating a Modern Microprocessor", *in proceedings of 17th International Conference on Computer Aided Verification*, Springer, Heidelberg, 2005, pp.2-4.

[15] B. Bentley., "Validating the intel pentium 4 microprocessor", *in proceedings of the 38th annual Design Automation Conference, ACM*, NY, USA, 2001, pp.244-248.

[16] D. L. Weaver, *OpenSPARC™ Internals*. Sun Microsystems, Inc. Santa Clara, CA, USA, 2008.

[17] M. Srinivas, B. Sinharoy, R. J. Eickemeyer, R. Raghavan, S. Kunkel, T. Chen, W. Maron, D. Flemming, A. Blanchard, P. Seshadri, J. W. Kellington, A. Mericas, A. E. Petruski, V. R. Indukuru, and S. Reyes, "IBM POWER7 performance modeling, verification, and evaluation", *IBM J. Res Dev*, vol. 55, no.3, pp. 4:1-4:19, 2011.

[18] K. -D. Schubert, W. Roesner, J. M. Ludden, J. Jackson, J. Buchert, V. Paruthi, M. Behm, A. Ziv, J. Schumann, C. Meissner, J. Koesters, J. Hsu, and B. Brock, "Functional verification of the IBM POWER7 microprocessor and POWER7 multiprocessor systems", *IBM J. Res. Dev.*, vol. 55, no.3, pp. 10:1-10:17, 2011.

[19] C. A. Krygowski, E. Almog, D. G. Bair, R. Breil, G. Dittmann, R. M. Gott, W. J. Lewis, A. D. Shah, and B. W. Thompto, "Key advances in the presilicon functional verification of the IBM zEnterprise microprocessor and storage hierarchy", *IBM J. Res. Dev.*, vol. 56, no.1.2, pp. 13:1-13:16, 2012.

[20] Z. Heng, and Shen Haihua., "Function verification of godson-2 processor", *J. Com. Res. and Dev.*, vol. 43, no.6, pp. 974-979, 2006.

[21] H. Yongqin, Z. Ying, J. Pengjin, W. Zhiuong, and C. Cheng, "Functional verification of ShenWei-1 high performance microprocessor", *J. Software*, vol. 20, no.4, pp. 1077-1086, 2009.

Received: July 23, 2014 Revised: August 12, 2014 Accepted: August 17, 2014