International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:9, No:3, 2015

Formal Verification of Cache System Using a Novel Cache Memory Model

Guowei Hou, Lixin Yu, Wei Zhuang, Hui Qin, Xue Yang

Abstract**—**Formal verification is proposed to ensure the correctness of the design and make functional verification more efficient. As cache plays a vital role in the design of System on Chip (SoC), and cache with Memory Management Unit (MMU) and cache memory unit makes the state space too large for simulation to verify, then a formal verification is presented for such system design. In the paper, a formal model checking verification flow is suggested and a new cache memory model which is called "exhaustive search model" is proposed. Instead of using large size ram to denote the whole cache memory, exhaustive search model employs just two cache blocks. For cache system contains data cache (Dcache) and instruction cache (Icache), Dcache memory model and Icache memory model are established separately using the same mechanism. At last, the novel model is employed to the verification of a cache which is module of a custom-built SoC system that has been applied in practical, and the result shows that the cache system is verified correctly using the exhaustive search model, and it makes the verification much more manageable and flexible.

Keywords**—**Cache system, formal verification, novel model, System on Chip (SoC).

I. Introduction

WITH increasing design complexity, verification becomes an important aspect of the design flow. In the meantime it has been observed that up to 80% of the overall design costs are due to verification. This is one of the reasons why formal verification method has been proposed as alternative to typical simulation method, since it cannot guarantee sufficient coverage of the design.

Formal verification has gained a lot of attention, because it allows proving the correctness of a circuit. And it ensures 100% functional correctness. Besides being more comprehensive, formal verification methods have also shown to be more cost effective in many cases, while testbench creation - usually a time consuming and error prone task - becomes superfluous [1].

Block-level verification is essential when working with SoC designs. Cache is one of the most important parts of SoC blocks. So cache verification becomes more and more significant. One of the most primary function challenges in cache design is the management of the integrity of the data transferred between cache and main memory. This problem is termed as cache coherence [2], [3]. As our cache control unit has an Icache used to control instruction cache, a Dcache used to control data cache, an Acache used to manage the communication between cache and Advanced High Performance Bus (AHB), and a MMU whose function is to convert the virtual address into the physical address [4]. The cache operations consist of read and write byte, half word, word, double word, burst read model, etc. Thus the verification of cache coherence is very complex. Simulation verification depends on test benches or vectors for verification and the speed relies on the size of design [5]. On the contrary, formal verification with model checking technology analyzes the RTL structure of the design and characterizes its internal nature. Each assertion violation discovered by model checking is reported with the counter-example, which uncovers functional errors that would have been missed using traditional verification methodologies.

In this paper, a formal verification of cache which is a module of a custom-built SoC system is presented using the exhaustive search cache memory model. Most of the models that have been used in cache formal verification apply large size rams to model the whole cache memory. However, this paper chooses four cache blocks as representations. Two of them are used to represent Icache memory, while the other two represent Dcache memory. This exhaustive search model reduces the capacity of cache memory model and makes the formal verification of cache system much more convenient.

The rest of the paper is organized as follows. Section II provides an overview of some verification techniques of cache including simulation and formal methods. Section III analyzes the limitation of model checking and the difficulties of cache formal verification. Section IV presents the flow of formal model checking and proposes the exhaustive search model of cache memory. Section V offers concluding remarks.

II. Research Actuality

There are many verification techniques, although the area of formal verification of cache is extremely significant, it is relatively new compared with simulation method. To our best knowledge, there has been some work published in this area [6]. However, the area of validation of cache has been a relatively more popular area of research. In this section, simulation and formal verification based on model checking are reviewed.

A. Simulation Verification

Simulation techniques are in common use. Engineer abstract the model from the specification of circuit, and then add external stimuli or data into this model, finally they estimate

Guowei Hou is with the Beijing Microelectronics Tech. Institution (BMTI), Beijing, China (phone: 18811721249; e-mail: maxhou@163.com).

Lixin Yu is with Beijing Microelectronics Tech. Institution (BMTI), Beijing, China. He is now with the Department of SoC (e-mail: 745845791@qq.com).

Wei Zhuang, Hui Qin, and Xue Yang are with the SoC Department, Beijing Microelectronics Tech. Institution (BMTI), Beijing, China.

whether the design obtain the design goal by observing the simulation result.

Directed test cases which are hand generated carefully by verification engineers become ineffective because of their sheer size and complexity. Then random and pseudo-random test generation techniques [7] are becoming popular. However, these techniques encounter some trouble owing to the issue of coverage.

All in all, simulation verification's advantage are - it is a classical verification, and now is in common use [8]. While its disadvantages are - it is an incompleteness method, and is a time consuming and nonflexible one to verify cache design.

B. Formal Verification

In design verification, it is difficult to detect bugs that depend on specific sequences of events. Detecting obscure bugs early on and the exhaustive nature of formal verification have been the main driving forces toward using formal verification techniques. Formal verification use mathematics to validate that the RTL design is consonant with design intention specification. Formal verification methods do not require test benches or vectors. They theoretically promise a very fast verification time and 100% coverage. The main idea of formal hardware verification is to prove the functional correctness of a design instead of simulating some vectors. Different techniques have been proposed for the proof process. Most of them work in the Boolean domain. There into, model checking [9] is one of the most useful formal verification methods.

Model checking perform an exhaustive checking – that is one that explores all possibilities and either confirms that a design responds correctly, or provides an error trace showing where the design responds incorrectly. A model checker is a software package that accepts as inputs a circuit model and a set of properties [10]. Formal model checking exploits formal mathematical techniques to verify behavioral properties of a design. The model checker evaluates the properties against the model. If the model satisfies the properties, it confirms that the model satisfies the property; otherwise, it will point out the error trace related to the properties.

Formal model checking is well suited to complex control structures, such as cache control unit. And it does not require any testbenches or vectors. The properties to be verified are specified in the form of queries. When the tool finds an error, it generates a complete trace from initial state to the state where the specified property failed.

III. Formal Model Checking Limitation and Difficulties of Cache Verification

Formal model checking provides an exhaustive check of the design properties and can be performed after link checking RTL code. This process enables early identification of error conditions that are not obvious candidates for deterministic simulation. However, it also has some limitations.

A. Formal Model Checking Limitations

Formal model checkers available in the industry have capacity restrictions. The correctness and integrity that model checking verification can achieve depends on the completeness and definitions of the properties that the users supply.

Formal model checking is effective for verifying control-intensive designs, but not for data path-intensive designs. Designs with data paths often have very large and deep state spaces, and the state space explosion will be exploded.

Model checking is not yet widely used as a verification tool because earlier model checkers are difficult to run. The property specifications are not intuitive, and designers have to learn special-purpose specification languages [11].

B. Difficulties of Cache Formal Verification

This paper applies formal model checking method to a cache system verification work, in the process, some difficulties are encountered.

First, the most important function that needs to be validated is cache coherence. As the cache consists of Dcache, Icache and Acache three main parts, it may be not easy to verify the whole cache system at the very start.

Second, to confirm that all functions are correct; this paper considers the communications between cache and cache memory, cache and AHB, cache and Integer Unit (IU). So in the process of verification, three monitors need to be established to represent the real units.

Third, the most important and difficult problem of the cache formal verification is the model of cache memory. As can be seen, the real cache memory is a very large ram, so most of the cache memory models that have been applied in cache verification are very complex.

To overcome these three difficulties, model checking method using exhaustive research cache memory model is presented.

IV. Model Checking Method and Exhaustive Research Cache Memory Model

A. Formal Model Checking Flow

Formal model checking requires the RTL code of the design. Most model checkers support Verilog description of the design. Fig. 1 shows the flow of model checking methodology.

1. Extract Properties

The properties that are useful to verification are Boolean expressions extracted from an informal description in the design's functional specification, design documents and informal communication with the hardware designer.

2. Partition the Design

Due to the problem of state space explosion, sometimes it is necessary to partition the design into smaller parts and verify each part separately. Because most designs use a hierarchical design methodology for model checking purposes, the same division been used by designer can be used here. For each part to be verified independently, it is important to discuss with designers to determine properties.

3. Model the Environment

The design must communicate with peripheral units in practice, so all possible interactions of the design with its environment must be calculated during the verification process [12]. This means engineers should model the environment by specified some constraints of design environment.

4. Verify Properties and Debug the RTL Design

After all steps above, verification tool will use its formal engine to validate the assertions (properties) in terms of the assumptions (constraints). If the verification of a property fails in a model checker, the tool will indicate a counterexample that is the shortest trace from an initial state to the failed state. The tool will also show the incorrectly assigned signals and other signals that are part of the property being verified which will help to backtrack through the source code and locate the error.

Then, the error information should be analyzed carefully. If the cause is due to an inaccurate abstraction of the environment, engineers should review the queries which consist of a set of properties and constraints. In contrast, if the cause of error is due to a design error, then designers should fix the bug.

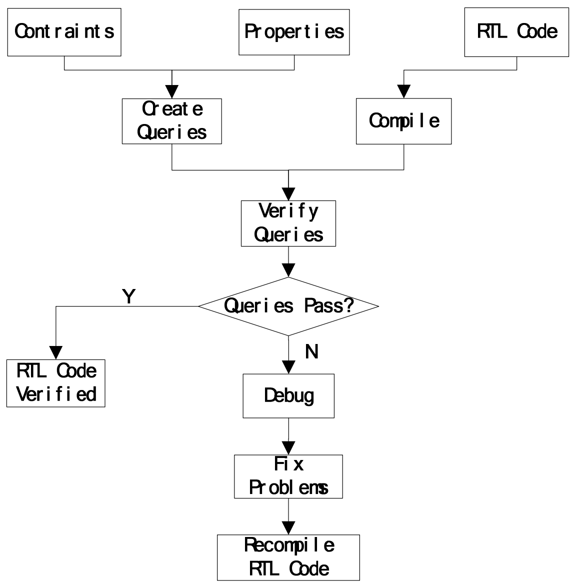


Fig. 1 Model Checking Methodology Flow

B. Model of Cache Memory

Cache memory is a very important and complex part of the cache environment according to the following two reasons. First, data needed by IU, tag and index that are used to identify different address, and valid which is used to indicate whether the data in cache memory is available are all saved in cache memory, so the correctness of data operation has important relevant with cache memory. Second, due to the big size of data cache and instruction cache data, the capacities of cache memory is too large and this will make cache formal verification too complex. To deal with this problem, simplified and exhaustive data cache memory and instruction cache memory model are presented with the help formal technology.

Instead of using large size rams to model the whole cache memory, this paper chooses four cache blocks as representations. Two of them are used to represent instruction cache memory, while the other two represent data cache memory.

1. Establish Dcache and Icache Memory Model

Dcache memory is modeled by D_RAM[1:0][33:0] and D_RAM_FIRST[1:0][33:0]. As data cache is two-set-associative, D_RAM[0][33:0] and D_RAM[1][33:0] represent set0 and set1 respectively. The model consists three parts: TAG, INDEX and VALID. TAG is used to save the address information which is used to indicate whether the word is requested. INDEX points out which cache line is to read, while valid is used to check the availability of words in the corresponding cache line.

Once the first dcache miss appear, D_RAM_FIRST and D_RAM will both be updated. From then on, every time the same dcache line misses, D_RAM will be updated and D_RAM_FIRST[1:0][3:0] will be used to save the valid value of former D_RAM while D_RAM_FIRST[1:0][33:4] keep immobile. Take set0 and valid[0] as an example. To validate the correctness of valid updating operation, the value of D_RAM_FIRST[0][3:0] and D_RAM[0][3:0] are compared when D_RAM_FIRST[0][33:6] amounts to D_RAM_FIRST[0][33:6]. If D_RAM_FIRST[0][3:1] is equal to D_RAM[0][3:1] and D_RAM[0][0] is high, then valid updating operation has been proved correctly.

Instruction cache memory model is established with the similar method of data cache memory.

2. Significance of Cache Memory Model

As the cache that is verified is a cache control unit without cache memory in practice, thus a monitor to model the memory which is necessary for cache verification should be made. The model has three functions in the paper. First, some bits of D_RAM/I_RAM are used to judge whether the cache miss or hit. Second, in cache miss case, some bits of the model act as the conditions to estimate whether the cache miss appears in the same cache line. Third, the valid value of model is due to validate the correctness of cache valid updating operation just as presented in Section IV.*B*.1.

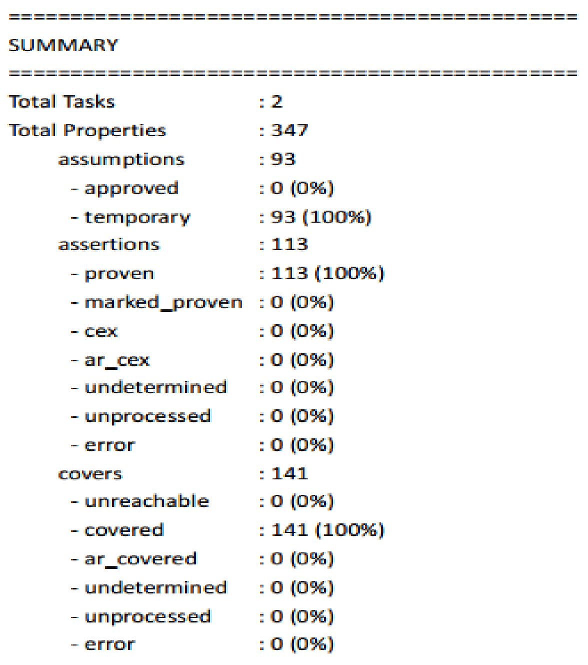
3. Feasibility and Predominance of the Exhaustive Search Model

This method to model memory is acceptant because formal verification applies non-determinism, which means all the possible values will be applied at Design under Testing (DUT) inputs. Thus the checker needs to watch only a part of the whole events, streams and so on. Once "a part" thoroughly picks any possibility, the checker covers all.

Due to the formal technique, the capacity problems of all cache memory models are greatly alleviated. The complexity of the cache verification has also been shrunk.

C. Result

The paper verifies a cache system using the exhaustive search cache memory model. Fig. 2 shows the verification result. As can be seen, 347 properties which consists 93 assumptions, 113 assertions and 141 covers have been considered in the verification process. All assumptions are environment constraints which are used to restrict the inputs of the device under verification to reasonable values. Assertions have two functions. Those who correspond to the constraints are used to monitor whether the constraints are violated, while others are used to validate cache functions like data integrity. In Fig. 2, 113 assertions are all proven and the covers are covered 100%. That means the cache is verified correctly.



```
===============================================
SUMMARY
===============================================
Total Tasks               : 2
Total Properties          : 347
      assumptions         : 93
       - approved         : 0 (0%)
       - temporary        : 93 (100%)
      assertions          : 113
       - proven           : 113 (100%)
       - marked_proven    : 0 (0%)
       - cex              : 0 (0%)
       - ar_cex           : 0 (0%)
       - undetermined     : 0 (0%)
       - unprocessed      : 0 (0%)
       - error            : 0 (0%)
      covers              : 141
       - unreachable      : 0 (0%)
       - covered          : 141 (100%)
       - ar_covered       : 0 (0%)
       - undetermined     : 0 (0%)
       - unprocessed      : 0 (0%)
       - error            : 0 (0%)
```

Fig. 2 Result of cache formal model checking

In the verification process, the exhaustive search cache memory model is used. It proves that the cache system RTL consistent with design intention specification. The model reduces the capacity of cache memory monitor and makes the formal verification much more straightforward and convenient.

V. Conclusion and Future Work

The main contribution of this paper is the emphasis on the importance of formal model checking for cache system verification. Applying formal methods, some bugs that are difficult to be detected by typical simulation verification can be detected. With the help of formal technique, the exhaustive search model of cache memory which makes the cache verification more manageable is presented. The paper also shows the importance of extracting appropriate properties from the specification documents and converting them into accurate assertions, assumptions and coverage.

Model checking is now powerful enough to be widely used in SoC block-level verification especially in cache verification. More research should be done about formal verification methods, and more work should be done to optimize the new model of cache memory.

References

[1] R. Drechsler, Advanced Formal Verification (M), Kluwer Academic Publishers, 2004.
[2] S. Srinivasan, P. S. Chhabra, P. K. Jaini, A. Aziz and L. John, “Formal Verification of a Snoop-based Cache Coherence Protocol using Symbolic Model Checking,” International Conference on VLSI Design, pp. 288-293, Jan. 1999.
[3] P. Dhakad, A. Katariya and A. Arya, “Performance Verification for Cache Memory of Multicore Processor,” International Conference on Computational Intelligence and Communication Network, pp. 622-627, Nov. 2010.
[4] LEON2 Processor User’s Manual: Version 1. 0. 30, Jul. 2005.
[5] D. L. Dill, A. J. Drexler, A. J. Hu and C. H. Yang, “Protocol Verification as a Hardware Design Aid,” VLSI in Computers and Processors, pp. 522-525, Oct. 1992.
[6] E. T. Schubert, “Formal Verification of an MMU and MMU Cache,” 3rd NASA Symposium on VLSI Design, vol. 4, 1991.
[7] T. J. Li, J. M. Zhang and S. K. Li, “An FPGA-based Random Functional Verification Method for Cache,” IEEE Eighth International Conference on Networking, Architecture and Storage, pp. 277-281, Jul. 2013.
[8] M. Tomasevic and V. Milutinovic, “A Simulation Study of Snoop Cache Coherence Protocols,” Hawaii International Conference on System Sciences, pp. 427-436, Jan. 1992.
[9] W. Grumberg and D. E. Long, “Model Checking and Modular Verification,” ACM Transaction on Programming Languages and Systems, pp. 843-871, 1991.
[10] A. Miczo, Digital Logic Testing and Simulation, 2nd edition, published by John Wiley & Sons, Inc., Hoboken, New Jersey, 2003.
[11] O. Rashinkar, P. Paterson and L. Singh, System-on-a-chip Verification Methodology and Techniques, Kluwer Academic Publishers, 2002.
[12] B. H. Bao, J. Bormann, M. Wedler, D. Stoffel and W. Kunz, “Formal Plausibility Checks for Environment Constraints,” Forum on Specification and Design Languages, pp. 13-18, Sep. 2012.