Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2015, Article ID 915409, 10 pages
http://dx.doi.org/10.1155/2015/915409

Research Article

Hybrid Model: An Efficient Symmetric Multiprocessor Reference Model

Shupeng Wang,[1] Kai Huang,[1] Tianyi Xie,[2] and Xiaolang Yan[2]

[1]*Department of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China*
[2]*Institute of VLSI Design, Zhejiang University, Hangzhou 310027, China*

Correspondence should be addressed to Kai Huang; huangk@vlsi.zju.edu.cn

Received 23 September 2014; Revised 26 January 2015; Accepted 15 March 2015

Academic Editor: Marco Platzner

Functional verification has become one of the main bottlenecks in the cost-effective design of embedded systems, particularly for symmetric multiprocessors. It is estimated that verification in its entirety accounts for up to 60% of design resources, including duration, computer resources, and total personnel. Simulation-based verification is a long-standing approach used to locate design errors in the symmetric multiprocessor verification. The greatest challenge of simulation-based verification is the creation of the reference model of the symmetric multiprocessor. In this paper, we propose an efficient symmetric multiprocessor reference model, Hybrid Model, written with SystemC. SystemC can provide a high-level simulation environment and is faster than the traditional hardware description languages. Hybrid Model has been implemented in an efficient 32-bit symmetric multiprocessor verification. Experimental results show our proposed model is a fast, accurate, and efficient symmetric multiprocessor reference model and it is able to help designers to locate design errors easily and accurately.

1. Introduction

Recently, the symmetric multiprocessor (SMP) has become a leading trend in the development of advanced embedded systems. Meanwhile, with the rapid improvement of the hardware manufacturing technologies and the help of computer-aided design (CAD) tools, SMP systems become more and more powerful and complex. As a result, the design verification of SMP systems takes up a large part of the total design period. The verification method directly determines the efficiency of SMP system verification and even the whole design cycle.

A variety of techniques have been deployed to efficiently and effectively detect design errors in SMP systems. These techniques can be divided into three categories: formal verification, simulation-based verification, and hardware emulation [1–3]. Various formal verification methodologies with the relevant environment setup have been proposed and used [4–9]. Formal verification, such as model checking and theorem proving, takes advantage of mathematical methods to judge whether the behavior of the design follows the rules instituted by designers. With the increasing size of system design, the space needed by formal verification is beyond the ability of tools and the process of formal verification is slow. As a result, the formal verification is not appropriate in large-scale system verification, such as the SMP system verification. Hardware emulation maps a gate level model of the design onto Field-Programmable Gate Array (FPGA) on the emulation system. It is much faster than the simulation-based verification. The main disadvantage of the hardware emulation is that it is difficult to debug when an error takes place. Simulation-based verification [10–14] is the most used method to verify the function of the SMP systems. It generates instruction sequences that are then fed in parallel to the design under test (DUT) and its reference model. Any discrepancy between the two models indicates a design error. Simulation-based verification is able to locate the errors easily and rapidly, and it is not limited by the size of system. As a result, it is widely used in SMP system verification.

The major drawback of the mainstream simulation-based approach is the difficulty of creating an efficient reference model of the DUT in a short time. The success of simulation-based verification depends on the accuracy and the quality of the reference model in use. An efficient and accurate

reference model is able to help designers locate errors easily and quickly. Many researchers have already proposed various reference models of the processor at presilicon. During the simulation-based verification, most processors regard the simulator as the reference model. These simulators are normally obtained from earlier stage in processor development, in which simulators are used for performance evaluation under benchmark [15]. Some of these simulators cannot support SMP verification, such as SimpleScalar [16]. Some other simulators, such as MARSS [17] and PTLsim [18], can be implemented to verify SMP systems. However, these simulators are usually timing-accurate; it is time-consuming for design verification by using these simulators to act as the reference model. In addition, the verification of these simulators themselves is often very complicated due to their architectural complexity [19]. As these models are usually timing-accurate, they are called timing-accurate models (TMs). The other type of reference model is Instruction Set Simulator (ISS) that is function-accurate. ISS only cares about the system function and its architecture is simple. These simulators are relatively easy to ensure due to their simpler architectures. This enables them to be used as reference model in the functional verification of the single-core processors. However, as they have no ability to sequence the out-of-order load/store transactions among CPUs perfectly, they cannot be used to verify the SMP system efficiently. As these models are function-accurate, they are called function-accurate models (FMs). It is difficult to test the function of the SMP system by using the timing-accurate models and function-accurate models efficiently. Such difficulties prompt us to create an efficient SMP reference model that is called Hybrid Model (HM). This model is simpler and faster than the timing-accurate model and more accurate than the function-accurate model. SystemC can be very effective in describing the system architecture and functionality to support high-level simulation. So SystemC can be used to obtain the efficient HM. When the reference model has been created, tests are fed in parallel to the DUT and its reference model to check design correctness.

In a simulation process, function coverage analysis is needed to check and show the quality of testing. It helps the verification team to check whether the function points that they want to simulate are covered during the testing phase. Sometime, some direct tests written by hands are needed with the help of function coverage analysis to cover the missing cases. The function coverage analysis is usually achieved from the RTL (Register Transfer Level) code and indicated by one signal or a set of signals. As the verification team is unfamiliar with the RTL code, it is difficult for them to observe the function points in RTL code, especially if the signals needed by the function points do not exist in the RTL code and the verification team has to turn to the designers for help. It is necessary for the designers to add these signals that are useless to the system function. In this way, the function coverage analysis needs the interaction of the verification team and the designers, so it is error-prone. However, the verification team is familiar with the reference model that is created by them. So if they achieve the function coverage analysis from the reference model rather than from the RTL code, the function coverage result can be more accurate. And the direct tests are able to be written by the verification team more effectively.

The main contribution of our work is that an efficient SMP reference model is proposed. It is written with SystemC. Acting as the SMP reference model, HM is simpler and faster than TM and more accurate than FM. The second contribution is that we define a timing sequence called Dependent Timing Sequence (DTS). The function of DTS is the timing interface between two models. The final contribution is that the function coverage analysis is able to be obtained from HM. In this way, the verification team can achieve more accurate coverage result quickly. Then the direct tests can be written by them more effectively.

2. Hybrid Model

As shown in Figure 1, the Hybrid Model (HM) consists of CPU Pipeline Model (CPM) and Cache Coherence Model (CCM). A common SMP consists of CPU pipelines, Load Store Units (LSUs), caches, and the interconnection between CPUs. The interconnection is responsible for maintaining the cache coherence between CPUs. The reference model of CPU pipeline is the function-accurate CPM. As the interconnection, LSU, and cache are related to load/store transactions, they are called Load Store Module (LSM). LSM is closely related to cache coherence and its reference model is the timing-accurate CCM. CPM and CCM are connected through DTS. The whole SMP system can be verified efficiently with the cooperation of CPM and CCM.

In the validation process, when a test case is stressed on the SMP system and HM simultaneously, the SMP system executes and HM simulates the instructions in this test case one by one. For each single instruction, the CPU pipeline executes it and the execution results of the CPU pipeline are obtained. If this instruction is a load/store instruction, the CPU pipeline needs to send this instruction to LSM. Then LSM executes this instruction and the execution results of the LSM are obtained. In this way, the execution results of the whole SMP system are obtained. On the HM side, first CPM simulates this instruction and the simulation results of CPU pipeline are achieved. If this instruction is a load/store instruction, CPM has to pipe its timing stream to CCM via DTS accordingly. The timing stream makes CCM begin to simulate and the simulation results of LSM are achieved by CCM. In this way, the simulation results of the whole SMP system are achieved. At this time, the tool will compare the execution results with the simulation results to check the correctness. Once any discrepancy occurs, the tool stops the simulation immediately. Then the tool will collect the information of this instruction such as its execution results and simulation results for the verification team. It is convenient for the verification team to locate errors with the help of these messages.

2.1. CPU Pipeline Model. An important part of HM is CPU Pipeline Model (CPM) that is function-accurate. It can be used to act as the reference model of CPU pipeline. CPM

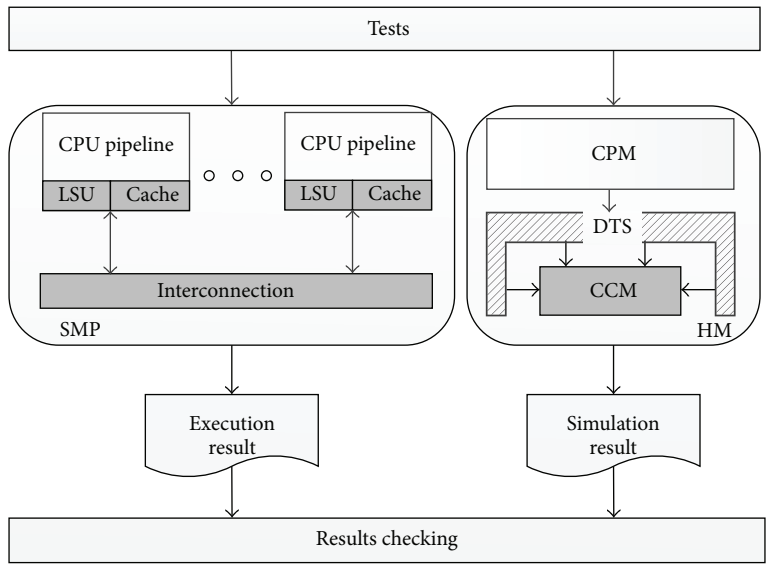

Figure 1: Efficient symmetric multiprocessor verification.

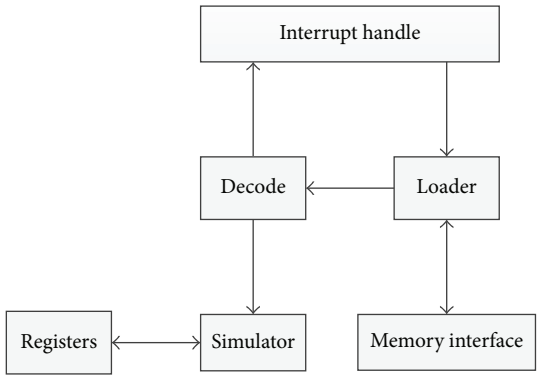

Figure 2: Block diagram of CPM.

only cares about the function of CPU pipeline rather than its timing information. As shown in Figure 2, three important modules of CPM are Loader, Decode, and Simulator. When CPM receives a test case needed to be simulated, Loader fetches the instructions in this test case one by one from memory according to the program counter (PC) first. Then Decode is responsible for decoding and interpreting these instructions. The Simulator is implemented with nonpipeline and it simulates these instructions directly. No matter whether instructions in the SMP system are in-order executed or out-of-order executed, they are retired one by one sequentially. As a result, the simulation results achieved by the nonpipeline Simulator directly are the same as the execution results obtained by the processor after going through complex CPU pipeline. For the instructions that are not load/store transactions, there is no need for them to be sent to LSM, as they are irrelevant to the cache coherence. For these transactions, all the simulation results of them can be achieved by CPM and the simulation is over after updating the value of registers. For load/store transactions, they need not only to go through CPU pipeline but also be sent to LSM. CPM is responsible for piping the timing stream of load/store instructions to CCM via DTS when it has finished its simulation of these instructions. The timing stream makes CCM begin to simulate. The simulation process of a load/store instruction is finished when CPM gets the response from CCM and the value of registers is updated. If an interrupt is found in this process, CPM needs to jump to interrupt handler.

The simulation results of CPU pipeline can be obtained rapidly, including much key information of the SMP system, for example, PC, the value of registers, and the state of the target processor. The tool compares these simulation results achieved by CPM with the execution results obtained by DUT. And any discrepancy indicates an error of the DUT. If no discrepancy occurs and the simulating instruction is not a load/store instruction, the simulation of this instruction is finished successfully. If this instruction is a load/store instruction, CPM has to send the complete timing information of this instruction to CCM via DTS. If an error occurs, the simulation will be stopped at once and the simulation results and the execution results are obtained directly to help the verification team to locate and fix this error.

2.2. Cache Coherence Model. The other important part of HM is Cache Coherence Model (CCM) that is timing-accurate. CCM is the reference model of LSM. As CCM is timing-accurate, it needs to care about the details of LSM. However, only the details that have an effect on the function points that the verification team wants to simulate are considerable. The function points are defined manually by the verification team, and they are the combination of the characteristics

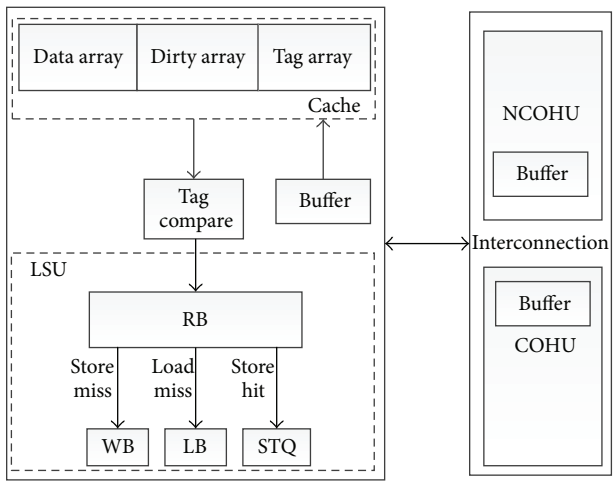

Figure 3: Block diagram of hardware. RB is used to preserve load/store transactions and maintain their order. WB keeps store miss transactions, LB is responsible for preserving load miss transactions, and STQ keeps store hit transactions. COHU maintains cache coherence between cores and NCOHU deals with the transactions unrelated to cache coherence.

of the DUT and a series of events that must be verified. In the application, these events are analyzed by observing the signals and states of the DUT. When the verification team has finished listing these events, they would serialize the events that have close relationship and outline their features. Finally the events that have close data relationship are put in one process according to the serialized events and the relationship of data structure between these events. As a result, these processes can be implemented with SystemC and run in parallel. And the processes communicate with each other by FIFO.

Figure 3 shows the common block diagram of the interconnection, cache, and LSU of the SMP system. The load/store transactions are first preserved in the Request Buffer (RB) in LSU. Then these transactions are sent to cache to decide whether the cache lines they want to access are located in cache. Further, they are sent to the appropriate buffers to wait for the chance to access the interconnection. The store miss transactions are sent to Write Buffer (WB), the load miss transactions are sent to Load Buffer (LB), and the store hit transactions are sent to Store Queue (STQ). Then they are sent to the interconnection when they have obtained the permission. The Coherence Unit (COHU) would maintain cache coherence between cores and handle the transactions related to cache coherence. The address domains of these transactions are cacheable and shareable. On the contrary, the function of Noncoherence Unit (NCOHU) is to deal with the transactions unrelated to cache coherence. The address domains of these transactions are other domains. The framework of the LSM is so complex; it is difficult and time-consuming for CCM to be created the same as the hardware. Some unnecessary hardware architectures can be abstracted due to the relationship between the hardware architectures and the function points the verification team wants to simulate. If the abstraction of some hardware architectures has no effect on the function points and the accuracy, these hardware architectures can be removed in CCM. When the number of cores in the multiprocessor system has been changed, the designers would modify some details of the interconnection according to the specification.

As the main memory has a lower load/store speed, buffers are utilized in the NCOHU to save load/store transactions unrelated to cache coherence. However, it is fast to access software memory. As a result, there is no need to create buffers for memory access in CCM. And sometimes more than one transaction attempts to access cache, whereas cache is a one-port element. So buffers are needed to save the outstanding requests to cache. However, CCM can accept and execute all the requests simultaneously, so no buffer is needed to save these transactions to cache in CCM. The abstraction of these buffers not only has an effect on the function, but also can reduce the implementation time of CCM. However, some hardware architectures cannot be abstracted; even any discrepancy between the hardware and CCM may cause fatal functional mistakes.

The interconnection usually works faster than CPU; some of the transactions related to cache coherence need to be saved in COHU. The order of these transactions is maintained by COHU in order to achieve accurate execution results. CCM has to deal with these load/store transactions in the same way with hardware to obtain the right simulation results. Figure 4 shows the different simulation results caused by the different orders of store transactions. A certain cache line is located in both CPU0 and CPU1 cache. At cycle A, CPU0 and CPU1 send store requests to the interconnection simultaneously. As shown in Figure 4(a), as the arbitration result of these two store transactions is that CPU0 could execute the store transaction before CPU1, the store transaction of CPU0 is accepted by the interconnection at cycle A; however, the store transaction of CPU1 is not accepted which is indicated by the symbol $*$. Then the store transaction of CPU1 is accepted by the interconnection at cycle B. At cycle C, the cache line in CPU1 cache is invalidated by the interconnection and the state of the store transaction of CPU1 is modified from store hit to store miss. At cycle D, the interconnection accepts the load transaction of CPU2, and the data CPU2 loads is 2. On the other hand, as shown in Figure 4(b), if the arbitration result of these two store transactions is that CPU1 could execute the store transaction before CPU0, the data CPU2 loads would be 1 at cycle D. The data CPU2 gets highly depends on the arbitration of these two store transactions of CPU0 and CPU1. Different execution orders lead to different results; hence, CCM has to achieve timing-accurate for these transactions to avoid errors.

Figure 5 shows the block diagram of CCM. The function of NCOHU is the same as that of SMP. But there are no buffers in NCOHU of CCM. The COHU of CCM is the same as that of SMP, not only their functions but also timing. No buffer is needed for cache in CCM. When the number of cores in the multiprocessor system has been changed, the verification team would modify some details of CCM according to the hardware changes made by the designers. Hence, HM can go to perform well even when the number of cores increases to hundreds.

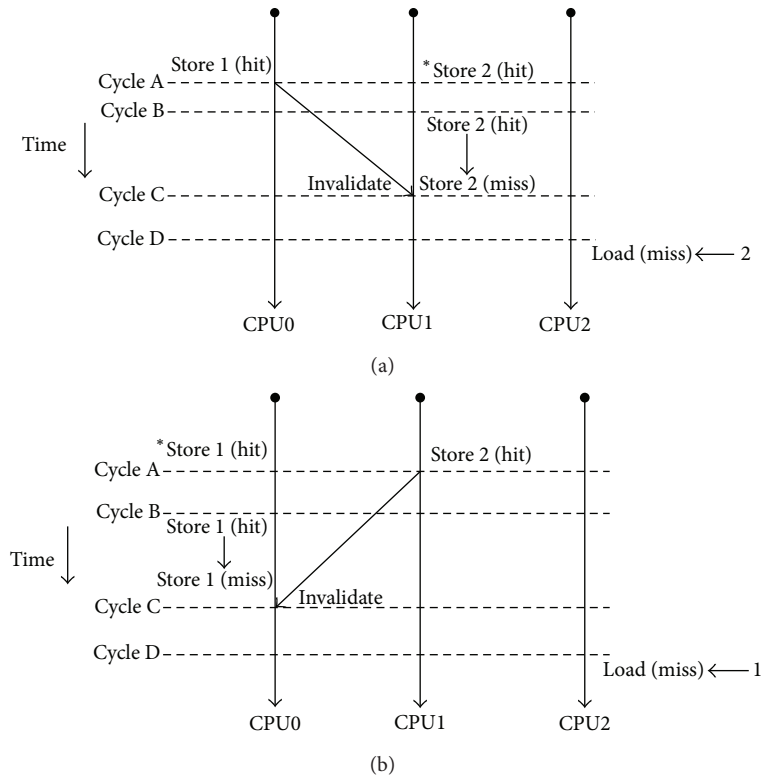

Figure 4: The execution result depends on the arbitration. (a) CPU0 executes before CPU1. (b) CPU1 executes before CPU0.

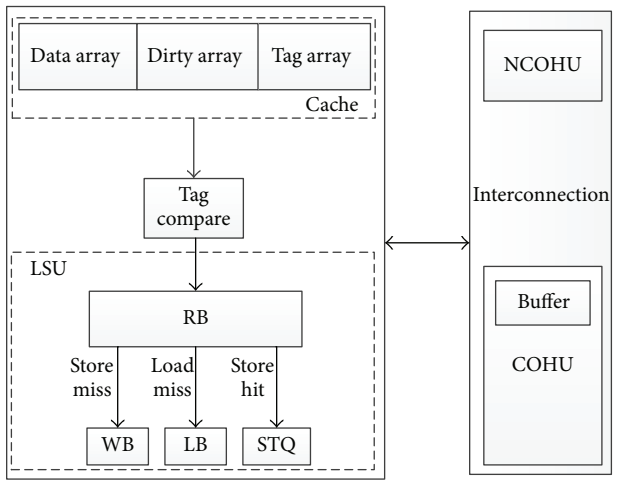

Figure 5: Block diagram of CCM.

2.3. Dependent Timing Sequence. Dependent Timing Sequence (DTS) is the timing interface between CPM and CCM. For every single instruction, CPM simulation and CPU pipeline execution proceed simultaneously. The tool compares the simulation results with execution results all the time. If no error is found in CPU pipeline and the simulating instruction is a load/store transaction, CPM is responsible for delivering the timing information of this transaction to DTS. CPM is aware of all the timing information of this transaction except for the cycle number whose function is to notify CCM when to begin its simulation. However, CPM can find this information from the execution results of hardware. In this way, the complete timing sequence of this transaction can be obtained and piped to DTS by CPM. DTS includes all the timing information CCM needs. Then CCM reads the timing information from DTS and begins its simulation. Figure 6 shows the timing information in a simulation process. Transaction type indicates the type of this transaction. Transaction size indicates the byte amount in this transaction. Data means the data CPU stores and x indicates that this transaction is a load transaction. Coherence indicates whether this transaction relates to cache coherence or not. As shown in Figure 6, at cycle number 21, CPU0 stores 1 into address 0x1fff_fee8, and CPU1 stores 2 into the same address. If these two store transactions are both store hit transactions, the condition is similar to what is shown in Figure 4.

As different kinds of CCMs may need different timing information, the information in DTS should be adjusted to meet the timing requirements of CCM.

2.4. Function Coverage Analysis. As HM is written by verification team and only includes the considerable function points, it is fast to obtain the function coverage report. Moreover, the isolation between system design and verification

Cycle number	CPU ID	Transaction type	Transaction size	Address	Data	Coherence
Tail → 112589	2	1	1	0x2020_001c	0	1
○ ○ ○						
156	3	0	3	0x3020_0008	x	0
21	1	1	4	0x1fff_fee8	2	1
21	0	1	4	0x1fff_fee8	1	1
Head → 5	0	0	4	0x1fff_fff0	x	1

Figure 6: Timing information in DTS.

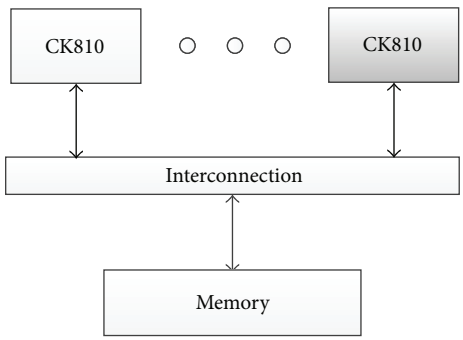

Figure 7: Architecture of CK810MP.

due to the proposed function coverage analysis approach can avoid many unnecessary errors in function coverage report and make the analysis more accurate.

3. Experimental Results

3.1. Verification Platform. We selected the CK810MP of Hangzhou C-SKY Microsystems Co., Ltd., to evaluate the feasibility of HM. As shown in Figure 7, CK810MP system consists of several modified CK810 processors, interconnection, and memory. CK810 is a high-performance 32-bit embedded processor based on CSKY v2 instruction set and its LSU is modified to support cache coherence according to the specification. A number of CK810 processors are connected by a bus-based interconnection that is responsible for maintaining cache coherence and dealing with requests to memory. The data channel and instruction channel are separate to increase bandwidth. Finally, an efficient SMP, CK810MP, is obtained with the addition of memory. We made extensive experiments with a CK810 quad-processor system, as the quad-processor is the mainstream of the embedded systems currently, such as mobile phones and personal computers. In addition, the quad-processor can meet the performance requirement of most of embedded applications, and it is a good tradeoff between performance and power. We chose SystemC to act as our program language and created a timing-accurate model (TM), a function-accurate model (FM), and a Hybrid Model (HM) to act as the reference models of CK810 quad-processor. As FM is only interested in the design function and easy to be created, it took more than 20 days to complete this model. It took almost 6 months to achieve the TM, as it cares about the majority of details of the target CK810 quad-processor. As the HM pays attention to a part of the details of the target processor, it took almost a month to obtain HM. To compare our proposed model with state-of-the-art simulation models, we selected GEM5 [20], which is a popular open-source timing-accurate multiprocessor simulator, to act as the reference model of the target CK810 quad-processor. GEM5 simulator supports a wide range of processor instruction set architectures (ISA), such as Alpha, ARM, MIPS, PowerPC, and x86. However, the GEM5 simulator cannot support the CSKY v2 instruction set. The CSKY v2 instruction set is much less complex than ARM. Moreover, the similar instructions can be found in ARM instruction set for most of the instructions of the CSKY v2 instruction set. Hence, we can use CSKY-to-ARM instruction translation to make GEM5 support the CSKY v2 instruction set and act as the reference model of CK810MP system.

Figure 8 shows the verification platform of CK810MP. DMA (Directly Memory Access) is able to help improve the system performance effectively. TLB (Translation Lookaside Buffer) translates virtual addresses to physical addresses. Each test was generated by a test generator based on random selection from more than 20 types of instructions, such as math, logic, load, store, and jump supported by the CK810 core. The generated tests were stressed on CK810MP system and its four reference models, respectively. The function coverage analysis was performed to direct the verification effort. We obtained four comparison results by comparing execution results of CK810MP system with the simulation results of these four reference models. According to these four comparison results, errors of CK810MP were discovered.

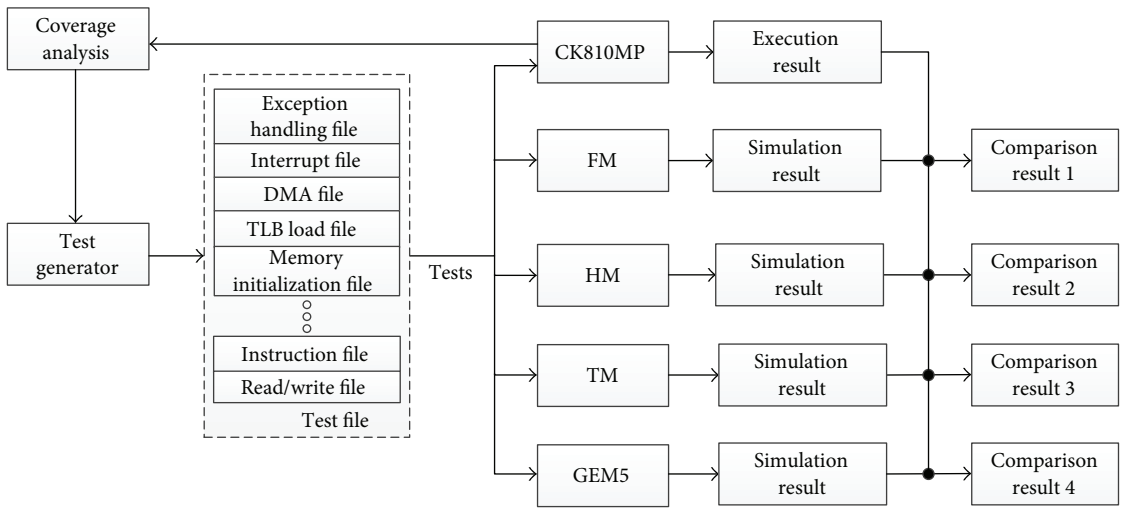

Figure 8: Verification platform.

3.2. Simulation Speed. The test generator generated 4000 tests each with 100 instructions, including the boot sequence used to initialize the CK810 core. In the first experiment, we compared the simulation speeds of these four models of CK810MP. To obtain the differential results, these 4000 tests were divided into 10 test groups randomly and each test group has various numbers of tests. The numbers of tests included by these 10 groups gradually increased from the first one to the tenth one. Then these test groups were fed to the reference models of CK810 quad-processor system, respectively, to compare their simulation speeds. Figure 9 shows the average simulation time of these four reference models stressed by these test groups. As shown in Figure 9, the simulation speeds of TM and GEM5 are similar, and they are the slowest in these four reference models as they are both timing-accurate. The simulation speed of FM is about 600 times those of TM and GEM5, and it is the fastest in these four reference models. The simulation speed of HM is about 30 times those of TM and GEM5. In comparison to FM, HM is slower, but it has a much better performance than TM and GEM5 in speed.

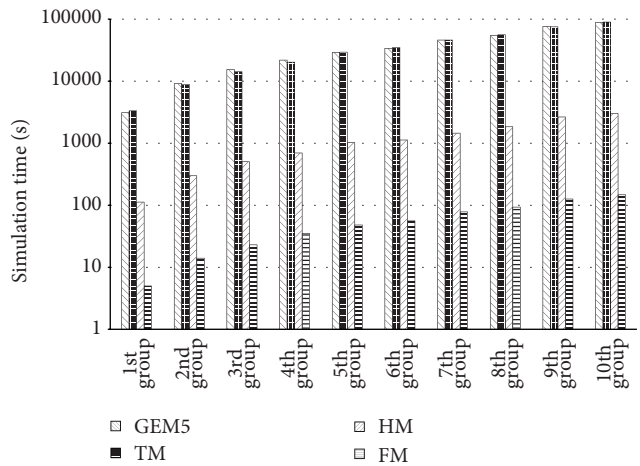

Figure 9: Simulation speed.

Further, we focused on the functional design of CPU pipeline in HM (denoted as CP-FM) and the timing-accurate model of CPU pipeline in TM (denoted as CP-TM) to explain why HM has obvious speed advantages comparing with TM. The test groups were fed to CP-FM and CP-TM to compare their simulation speed. Figure 10 shows the comparison of simulation speeds of CP-FM and CP-TM. The simulation speed of CP-FM is about 720 times as that of CP-TM. This means the speed advantage of HM comes from the functional model of the CPU pipeline.

3.3. Accuracy. In the second experiment, we compared the accuracy of these four models indicated by the number of errors found by them. The 4000 tests in the simulation environment were divided into 10 test groups each with 400 tests randomly. Then these test groups were stressed on CK810MP system and its four reference models, respectively. Figure 11 shows the number of the errors found by these 10 test groups and accumulated errors found by these four reference models. As shown in Figure 11, the abilities of TM and HM to find errors are similar and stronger than those of GEM5 and FM. The accumulated errors found by HM are about 1.5 times as many as those found by GEM5. And the accumulated errors found by HM are about four times as many as those found by FM. The ability of FM to find errors is the weakest in these four reference models. As the GEM5 simulator is developed specifically to evaluate the performance of embedded systems, its details could not be the same as the details of the CK810 quad-processor system. Therefore, the accumulated errors found by GEM5 are much less than those found by TM and HM.

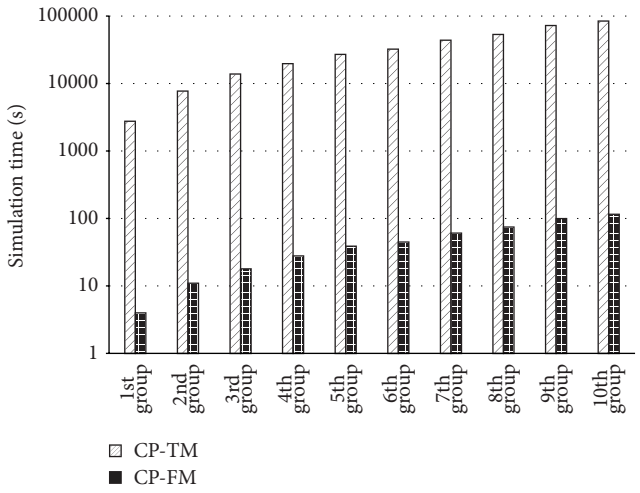

Figure 10: Comparison of simulation speeds of the function-accurate model and timing-accurate model of CPU pipeline.

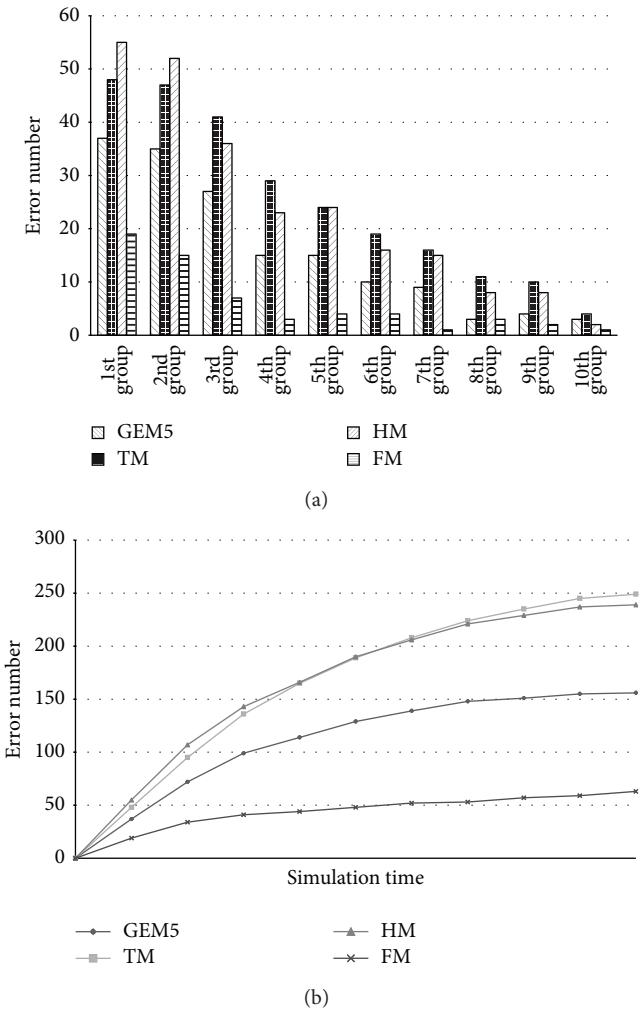

Figure 11: (a) Error number found by test groups; (b) accumulated errors.

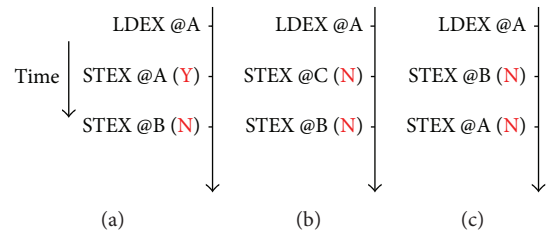

Figure 12: An example of exclusive transactions. (a) The correct implementation. (b) The wrong execution result caused by a design error. (c) The wrong simulation result caused by the timing inconsistency.

As soon as these four reference models' writing is finished, they are put into operation in the CK810 quad-processor verification. However, here these models are not exactly the correct golden models defined by the specification, especially the TM. The CPU pipeline of the CK810 quad-processor is a complex dual-emission superscalar 10-stage pipeline; hence some inconsistency between TM and the correct timing-accurate model is unavoidable at the beginning of simulation. The elimination of the inconsistency needs to take a lot of time. Before the TM becomes a correct timing-accurate model, it may obtain wrong simulation results because of some timing inconsistency, whereas the processor achieves the wrong execution results caused by a design error. If the wrong simulation results and the wrong results are the same, unfortunately, TM would take the attitude that the hardware is infallible. Figure 12 shows a simple example, where the results of store exclusive transactions are shown in brackets in red. Y indicates that the store exclusive transaction succeeds, while N shows that the store exclusive transaction fails. Figure 12(a) shows the correct implementation of three exclusive transactions, consisting of a load exclusive transaction and two store exclusive transactions. The first store exclusive transaction is executed successfully, as the exclusive transaction before this store exclusive transaction is a load exclusive transaction and they have the same address. The second store exclusive transaction fails. However, the address of the first store exclusive transaction is modified by a design error in CPU pipeline from address A to address C, as shown in Figure 12(b). As a result, the first store exclusive transaction fails. At the same time, as shown in Figure 12(c), TM inverts the order of two store exclusive transactions and these two store exclusive transactions both fail. In this way, TM cannot find this design error of CPU pipeline. However, HM is able to discover this design error as it can simulate these three exclusive transactions in the right order and obtain the right simulation results as shown in Figure 12(a). Hence, the design errors found by the HM are more than those found by the TM when the first two test groups are simulated.

As the simulation goes on, these models are all modified by the verification team to become the correct golden models gradually. At this time, if some timing errors of CPU pipeline do not influence the function of the CK810 quad-processor, the TM can discover these timing errors but the HM cannot. As an example, the interval between the load

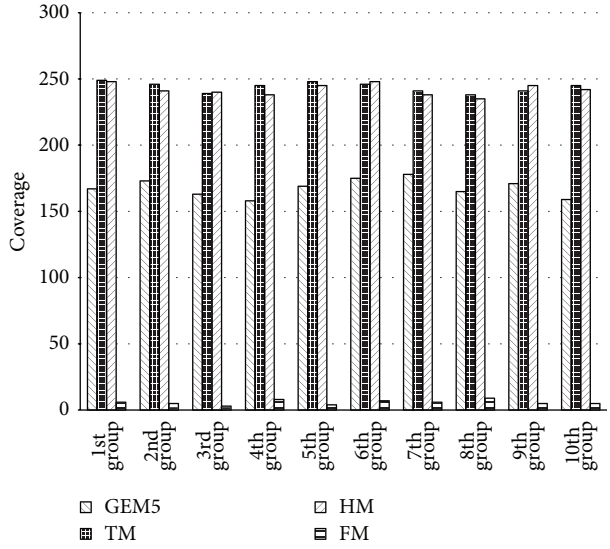

Figure 13: Coverage of function points.

exclusive transaction and the first store exclusive transaction in Figure 12 should be 10 cycles according to plan. However, the store exclusive transaction executed 2 cycles in advance because of the inappropriate change of a request pointer. This store exclusive transaction still succeeds. HM cannot discover this timing error but TM can, as the interval between these two exclusive transactions is ten cycles in TM. As a result, the design errors found by the TM are more than those found by the HM when the last eight test groups are simulated. And the accumulated errors found by the TM are more than those found by the HM at the end of simulation. However, the design errors that HM cannot discover have no effect on the function of processor and most of them can be discovered with the help of assertion checkers.

To compare the accuracy of four reference models further, we analyzed the coverage of function points that we want to simulate. Figure 13 shows the coverage of function points in these four reference models. The interconnection, cache, and LSU have 253 function points. HM and TM are capable of covering all the function points basically and the GEM5 simulator can cover partial function points. However, FM can only cover a few of function points.

Further, we focused on CP-FM and CP-TM to compare their accuracy and explain why HM has obvious speed advantages comparing with TM, while maintaining similar accuracy, by using the test groups used in Figure 11. Figure 14 shows the design errors found by these 10 test groups and the comparison of the accumulated errors found by CP-FM and CP-TM. As shown in Figure 14, 60 to 70 percent of design errors of the CK810 quad-processor are in the CPU pipeline, and the abilities of CP-FM and CP-TM to find errors are similar. The accumulated errors found by CP-TM are a little more than those found by CP-FM, as CP-TM can find the timing errors of CPU pipeline but CP-FM cannot. However, these errors are not functional errors and most of them can be discovered by assertion checkers. The experimental results in Figures 10 and 14 show that the function-accurate model

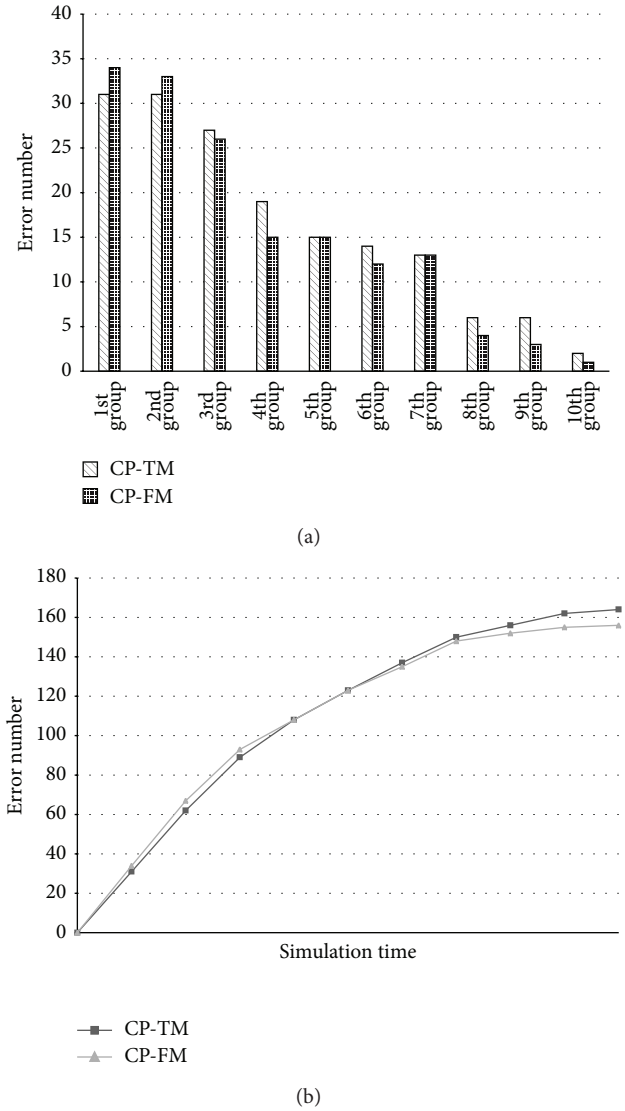

Figure 14: (a) Error number found by test groups; (b) accumulated errors found by the function-accurate model and timing-accurate model of CPU pipeline.

of the CPU pipeline is much faster than the timing-accurate model of the CPU pipeline, while the accumulated errors found by them are similar. This means the advantages of HM come from the functional design of the CPU pipeline model.

4. Conclusion

An accurate and efficient symmetric multiprocessor reference model is proposed in this paper. The function coverage analysis is able to be achieved from it to help the verification team to write direct tests more accurately. This reference model has been implemented for a 32-bit symmetric multiprocessor verification. The experimental results show that the number of errors found by our proposed model is about 4 times that found by a function-accurate model. Our proposed model has a better performance in finding errors

than the function-accurate model. The simulation speed of our proposed model is about 30 times as high as that of a timing-accurate model in the same condition. In comparison to the timing-accurate model, our proposed model is easier to create and faster, whereas their abilities to find errors are similar. The advantages of the proposed model come from the functional design of the CPU pipeline model. With the help of our proposed model, the verification team can locate design errors more quickly and verify the interconnection more efficiently. The time for symmetric multiprocessor verification can be shortened obviously with our proposed model.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

The authors would like to thank the members of the multiprocessor project team at Hangzhou C-SKY Microsystems Co., Ltd., especially Ke Wang, Xiaomeng Zhang, Teng Hu, and Xiaofei Jin, for their cooperation and help in this work.

References

[1] Y. Chen and V. Dinavahi, "Digital hardware emulation of universal machine and universal line models for real-time electromagnetic transient simulation," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 2, pp. 1300–1309, 2012.

[2] M. Peker, H. Altun, and F. Karakaya, "Hardware emulation of HOG and AMDF based scale and rotation invariant robust shape detection," in *Proceedings of the 1st International Conference on Engineering and Technology (ICET '12)*, pp. 1–5, Cairo, Egypt, October 2012.

[3] F. Ren and Y. R. Zheng, "Hardware emulation of wideband correlated multiple-input multiple-output fading channels," *Journal of Signal Processing Systems*, vol. 66, no. 3, pp. 273–284, 2012.

[4] R. Alur, "Formal verification of hybrid systems," in *Proceedings of the International Conference on Embedded Software (EMSOFT '11)*, pp. 273–278, Taipei, Taiwan, October 2011.

[5] M. Amrani, L. Lúcio, G. Selim et al., "A tridimensional approach for studying the formal verification of model transformations," in *Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation (ICST '12)*, pp. 921–928, IEEE, Montreal, Canada, April 2012.

[6] S. F. Siegel and T. K. Zirkel, "Automatic formal verification of MPI-based parallel programs," in *Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming (PPoPP '11)*, pp. 309–310, San Antonio, Tex, USA, February 2011.

[7] M. L. Bolton, E. J. Bass, and R. I. Siminiceanu, "Using formal verification to evaluate human-automation interaction: a review," *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans*, vol. 43, no. 3, pp. 488–503, 2013.

[8] R. Zhou, R. Min, Q. Yi, C. Li, and Y. Sheng, "Formal verification of fault-tolerant and recovery mechanisms for safe node sequence protocol," in *Proceedings of the 28th IEEE International Conference on Advanced Information Networking and Applications (AINA '14)*, pp. 813–820, Victoria, Canada, May 2014.

[9] H. Chockler, A. Ivrii, A. Matsliah, S. Moran, and Z. Nevo, "Incremental formal verification of hardware," in *Proceedings of the Formal Methods in Computer-Aided Design (FMCAD '11)*, pp. 135–143, Austin, Tex, USA, October 2011.

[10] E. Guralnik, M. Aharoni, A. J. Birnbaum, and A. Koyfman, "Simulation-based verification of floating-point division," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 60, no. 2, pp. 176–188, 2011.

[11] S. Seidel, U. Donath, and J. Haufe, "Approach to a simulation-based verification environment for material handling systems," in *Proceedings of the IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA '12)*, pp. 1–4, Krakow, Poland, September 2012.

[12] A. Braun, O. Bringmann, D. Lettnin, and W. Rosenstiel, "Simulation-based verification of the MOST netinterface specification revision 3.0," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '10)*, pp. 538–543, Leuven, Belgium, March 2010.

[13] A. S. Kamkin and M. M. Chupilko, "Survey of modern technologies of simulation-based verification of hardware," *Programming and Computer Software*, vol. 37, no. 3, pp. 147–152, 2011.

[14] E. Clarke, A. Donzé, and A. Legay, "On simulation-based probabilistic model checking of mixed-analog circuits," *Formal Methods in System Design*, vol. 36, no. 2, pp. 97–113, 2010.

[15] J. An, X. Fan, and S. Zhang, "A fast virtual device framework for improving RTL verification efficiency," in *Proceedings of the IEEE 3rd International Conference on Communication Software and Networks (ICCSN '11)*, pp. 73–75, IEEE, Xi'an, China, May 2011.

[16] T. Austin, E. Larson, and D. Ernest, "SimpleScalar: an infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[17] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: a full system simulator for multicore x86 CPUs," in *Proceedings of the 48th ACM/EDAC/IEEE Design Automation Conference (DAC '11)*, pp. 1050–1055, New York, NY, USA, June 2011.

[18] M. T. Yourst, "PTLsim: a cycle accurate full system x86-64 microarchitectural simulator," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS '07)*, pp. 23–34, San Jose, Calif, USA, April 2007.

[19] B. Glamm and D. J. Lilja, "Automatic verification of instruction set simulation using synchronized state comparison," in *Proceedings of the 34th Annual Simulation Symposium*, pp. 72–77, Seattle, Wash, USA, April 2001.

[20] N. Binkert, B. Beckmann, G. Black et al., "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.