DOI 10.31114/2078-7707-2019-2-12-17

Multicore Processor Models Verification in the Early Stages

N.A. Grevtsev[1,2], P.A. Chibisov[1]

[1]Scientific Research Institute of System Analysis (SRISA RAS), chibisov@cs.niisi.ras.ru

[2]MIPT, ngrevcev.cs.niisi.ras.ru

***Abstract* — In the paper the early stage of verification technology for multicore processor models testing is proposed. We demonstrate the applicability of the single core verification method extension where creating new multicore test generator is not required. The solution scheme deals with the adaptation method of some available single core stochastic testing approaches to a fully functional multicore testing tool.**

The proposed technique has been successfully applied to test RTL model of dual-core microprocessor with SMP developed in SRISA. The discussed approach was initially considered to be the first stage of RTL model testing, but the possibilities of the approach are also of interest for testing the model at the later stages of its design and functional maturity.

The testing process begins by creating simple random tests that check the MOESI coherence protocol. A new advanced random testing method based on the usage of proposed interleaved memory structures is developed to increase the probability of finding rare and hard to detect bugs in the memory subsystem.

The great advantage of the proposed memory allocation structure is that both cores gain access to the same cache-line simultaneously for read and write. Despite this accessibility, the methodology avoids losses of data consistency due to cores access to different bytes. Furthermore, each common area that is allowed to write to more than one core will contain deterministic values at every time.

The proposed approach aims to detect failures in cache coherence protocol, memory subsystem and memory buffers. Also, this testing system helps to find machine state errors that lead to deadlock.

***Keywords* — functional verification, multicore, stochastic testing, pseudorandom tests generation, memory subsystem, SMP, MOESI, cache coherence, pre-silicon verification.**

I. Introduction

The verification strategy of the Register Transfer Level (RTL) model of a multi-core microprocessor, considered in this paper, is to carry out the most complete testing of the project at the early stages of the design cycle using ready-made testing tools for single-core microprocessors. The purpose of the system approach is to verify the inter-core interactions between the individual blocks of the memory subsystem and the blocks themselves on a system-wide basis directly in the process of developing the model. It should be emphasized that the generator of single-core test programs does not require significant corrections and improvements.

Multi-core architecture of the microprocessor with symmetric memory access (SMP) allows to use resources of additional processor cores for parallelization of resource-intensive calculations or simultaneous multiple tasks performance. A common system controller in such a system provides access to RAM for each microprocessor core, as well as the connection of peripheral devices. An example of a block diagram of a multi-core system on a chip with SMP memory access architecture is shown in figure 1.

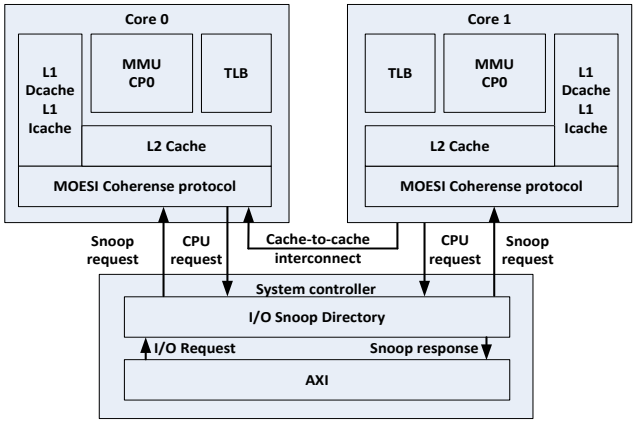

Fig. 1. Multicore SMP architecture

The data coherence problem can arise when two or more compute cores access the same data at the same time if one or more of these accesses are written. Thus, the most current data may be in the cache memory of one of the cores and will not be available to other devices in the system. This problem of data integrity preservation in multi-core microprocessors is solved with coherence protocol, according to which data synchronization between processor cores and peripherals is provided.

In spite of the fact that it is recommended to avoid data conflicts while writing concurrent programs, we consider it essential to create random test cases with forced shared data areas between threads intentionally to test the interaction between cores in a shorter period of time. We have the opportunity to apply well-established single core verification techniques and solutions for fully functional multicore models testing by template customization of user-defined memory allocation.

The article continues studies [1], [2] of microprocessor models stochastic verification. The following sections of the article describe the proposed new approach to early verification stages of multi-core microprocessor RTL models. We put emphasis on the early design stage, as well as the applicability of single-threaded testing tools.

II. Special Aspects of Memory Subsystem Verification

When designing new microprocessors, the study of new microarchitectural solutions effects can be required. For this purpose, a number of experimental models, which evaluate the effectiveness of solutions on a set of key criteria, such as the complexity of the implementation in HDL-code, time costs for implementation, performance, size of the occupied area on the chip, power consumption are created. Different approaches to the development require iterative experimental models creation to assess the performance and productivity of decisions.

With each new change, the performance of the model must be verified in a limited time. For example, the development process has required an assessment of the applicability of one of two options (inclusive and exclusive) for interaction between first-level and second-level cache memory. An inclusive organization assumes information duplication in the L1 and L2 caches, while exclusive cache memory assumes uniqueness of information in the L1 and L2 caches. To assess the optimality of such architectural decisions a test system is needed, which includes architecture certification and performance tests.

The article proposes a route of multi-core microprocessor models testing on the early stages. It allows to obtain a full-fledged multi-core testing tool by adapting existing single-core testing tools. Directed stochastic testing is one of the models testing methods at the system level. In conditions of extremely limited time frames and the lack of human resources, it is necessary to test the developed RTL model of a multi-core microprocessor without creating new testing tools aimed solely at coherence protocol verification.

The memory subsystem of modern multi-core microprocessors consists of multiple components on a single chip: MMU, TLB, cache memory of all levels and their controllers, data coherence support mechanisms, system controller, data prefetch and reordering buffers. The presence of many cores increases combinatorial complexity of the memory subsystem validation.

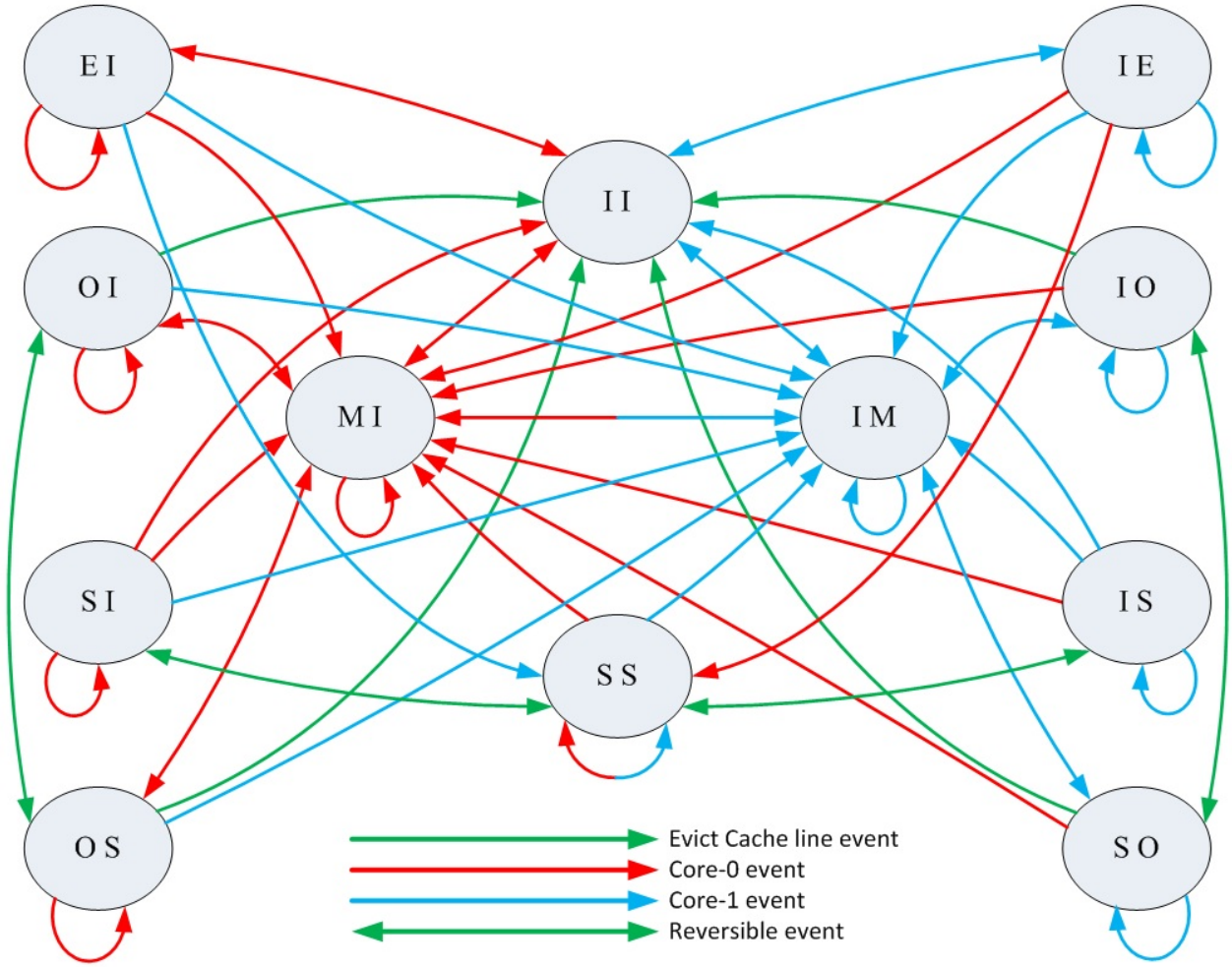

Fig. 2. State transitions in MOESI cache coherence protocol, projected on two cores basis

The cache coherence protocol describes the state of one cache line relative to a similar line in another core and does not affect operations on other cache lines. The state of the cache line is the state of the corresponding cache controller. All states can be divided into stable and transient. The stable states are defined by a subset of the states: Modified, Owned, Exclusive, Shared, Invalid (MOESI). Transitions from one stable state to another in modern cache coherence protocols do not occur instantaneously, but through transient states [3].

High complexity of the cache coherence protocol verification problems is due to the fact that combinatorial brute force search quickly leads to a state explosion. In [4], it is said that the coherence protocol can be verified separately according to its specification using the model verification method (model-checking). Meanwhile, our task is to check not only the design and implementation of the selected cache coherence protocol but the implementation of the entire memory subsystem in conjunction with other microarchitectural improvements mentioned above.

Figure 2 shows all possible transitions from the load-store requests, for a cache block in MOESI cache coherence protocol, projected on two cores bases. However, there are many other memory subsystem blocks which are used in the real system.

One of the methods of testing models at the system level is the directed stochastic testing method, which is of high effectiveness [2].

III. Applicability of existing tools

Before developing a random test generator for testing the memory subsystem of multi-core systems, it is proposed to modify the existing testing tools and create separate tests for each core, and then run them in parallel.

Meanwhile, the responsibility for the correct memory allocation between the cores lies on the developer of the template.

When building a template, the memory distribution between the cores is taken into account. Memory access instructions are randomly selected within specified limits. An example of a stochastic testing route for dual-core system verification is shown in figure 3.

The advantages of the technique are:

- possibility to start testing immediately without spending resources on developing a special generator,
- scalability: it is easy to extend the approach for any number of cores (2-16),
- ability to accurately define test scenarios with different degrees of freedom,
- absence of necessity to modify the generation process for different architectural solutions.

Nevertheless, this method has some limitations: it is necessary to control memory distribution between the cores by the template developer and control thread synchronization. Also, the proposed method is aimed at memory subsystem verification and does not affect the atomic operations testing.

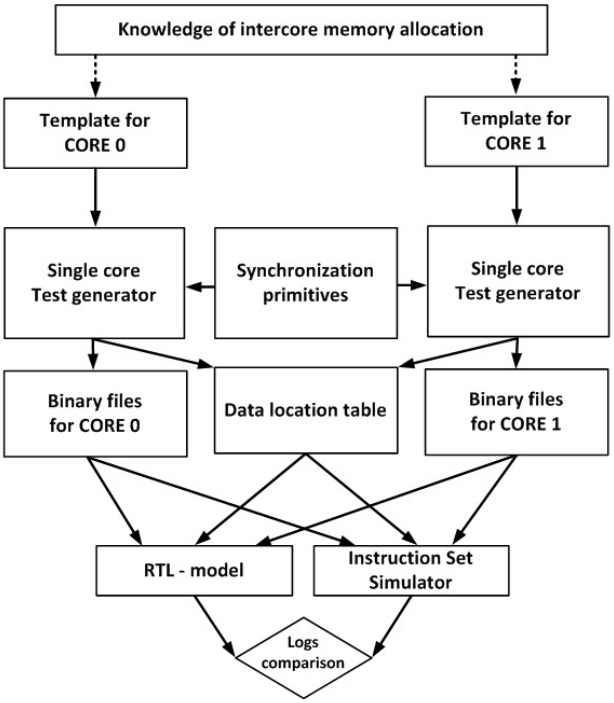

Fig. 3. Single core generator application diagram for multicore verification

IV. Test generation process for multicore microprocessors

This chapter describes the main stages of multi-core memory subsystems verification, corresponding to different levels of functional development of RTL-model maturity.

A. Hand-written tests aimed at cache protocol

The testing process begins (starts) with the creation simple (hand-written) tests in assembly language aimed at checking the MOESI cache coherence protocol. These tests check all transitions between MOESI states separately (figure 2), formalize and verify the multicore synchronization algorithms that will be used for further testing. With the help of such tests, the mechanism of the test correctness checking execution is debugged. For this purpose, the system of ISS (Instruction Set Simulator) and RTL-model logs comparison is used.

B. Hand-written tests with self-check

The root idea is that modification of independent variables sharing the same cache line by different threads (false sharing) results in computational performance degradation. Even one byte modification leads to the whole cache line replacement; therefore there may be situations when parallel threads of user application are unintentionally updating independent variables which are located in the same cache line, interchangeably. In such circumstances, a cache line update on one core leads to cache line eviction on the other core (cores) [5].

In publications [6] and [7], false sharing detection methods are investigated. Scalability for parallel threads of execution in an SMP system when there are many data conflicts related to the same cache line usage is also analyzed. Finally, it is recommended to avoid data conflicts described above while writing concurrent programs.

However, it is reasonable to create test cases with forced shared data areas between threads intentionally to test the interaction between cores. Threads compete for shared cache line iteratively in such test cases. The result of the same application, although implemented with another chosen data assignment pattern among threads (without false sharing), is accepted as a reference result of these tests.

The simplest example of a hand-written test based on false sharing with internal self-checks is arrays addition, a classical problem of software engineering (figure 4). Elements of the arrays are added in a variety of ways in order to obtain reference data for self-check. Let us consider an example of two ways of array addition.

Approach 1. No false sharing. Correct data assignment with relation to memory allocation. Each core has access to assigned cache lines during the test, for example, core 0 works with even-numbered cache lines, while core 1 works with odd-numbered cache lines only.

Approach 2. False sharing. Each core has access to different data in the same cache lines resulting in continuous data transfer.

The arrays obtained in a variety of ways have to be compared at the end of the test. Mismatch of the values shows the evidence of data loss during the test. Also, this can indicate that some data have not been updated and are now irrelevant.

C. Pseudorandom test generation

A stochastic verification method based on random tests generation being guided by a given processor instructions template is widely used for test coverage increase [2]. The method helps to find exceedingly rare and transient bugs such as bugs arising as a result of several instructions interaction in a processor pipeline, and as a result of multiple simultaneous cache memory requests.

The following sections of this chapter will address each of the three stages of random testing which were developed for RTL-model (pre-silicon) testing on different levels of the project maturity.

D. Separated cache lines

Microprocessor cores can be tested jointly on condition that intercore data transfer is reduced to a minimum in the initial phase of multicore RTL-model verification. In this case, the templates for the test generator should be arranged in such a way as to avoid a data collision, i.e., each core uses its own private memory regions. Address mapping in all levels of a cache memory is unique for every core. A physical memory description assigned in the generator configuration is given below.

Name	Lower	Upper
data1_core0,	0x0020000,	0x00207FF,
data2_core0,	0x0030000,	0x00307FF,
data1_core1,	0x0020800,	0x0020FFF,
data2_core1,	0x0030800,	0x0030FFF.

Data areas for core 0 and core 1 occupy different parts of L1 and L2 cache memories and do not intersect in the above examples.

This approximation was done in order to prevent accesses to the same bytes and even to the same cache line from read and write operations which are being issued by different cores. Otherwise, no assurance can be given that accesses to the shared memory region (or shared cache line) during RTL model simulation happen in the same order as during reference ISS (instruction set simulator) run. This situation of uncertainty is called a race condition.

A race condition here occurs when unordered memory accesses from different cores lead to unpredictable order of write operations, and, moreover, even if memory areas are chosen independently for each core, data from such areas can still hit in different ways of the same cache line in a set associative cache. In both cases, this results in impossibility of exhaustive comparison between RTL model simulator and ISS behaviors [8].

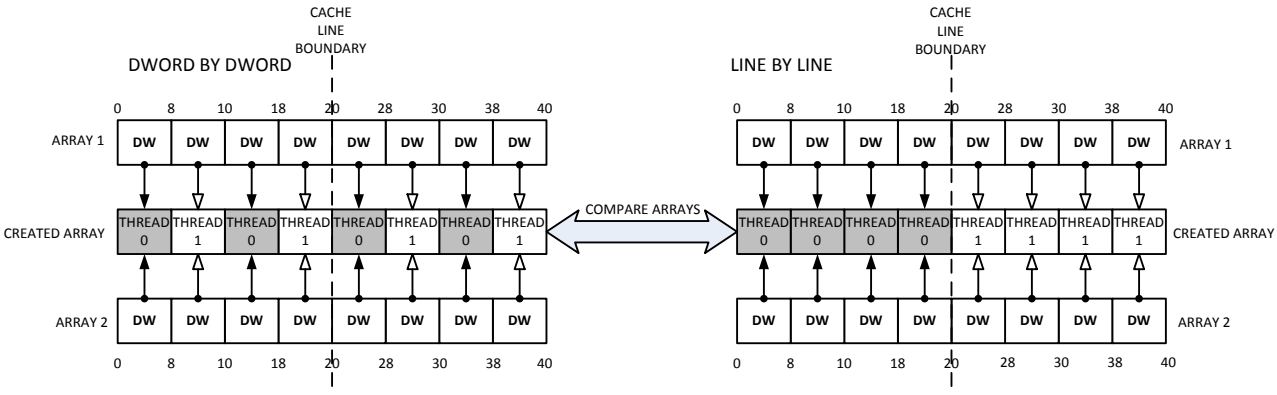

Fig. 4. Arrays addition example. Elements are added in two ways in order to obtain reference data for self-check

E. Cross-rerun

In the case of isolated cache lines, at the end of the test, all used lines go into a random modified state. This state can be used as a starting point to build a more complicated test. In the second half of the task, the test code which has just been executed on core 0 is passed to execution to the core 1 and vice versa. This ensures that each requested cache line will be in changed (edited) state in the cache memory of the other core. This technique makes it possible to improve testing mechanisms efficiency of inter-core interactions (temporal relationships between the cores and the cache memory of the cores) due to a greater variety of test situations. The structure of the cross-run test with examples of MOESI state transitions is shown in figure 5.

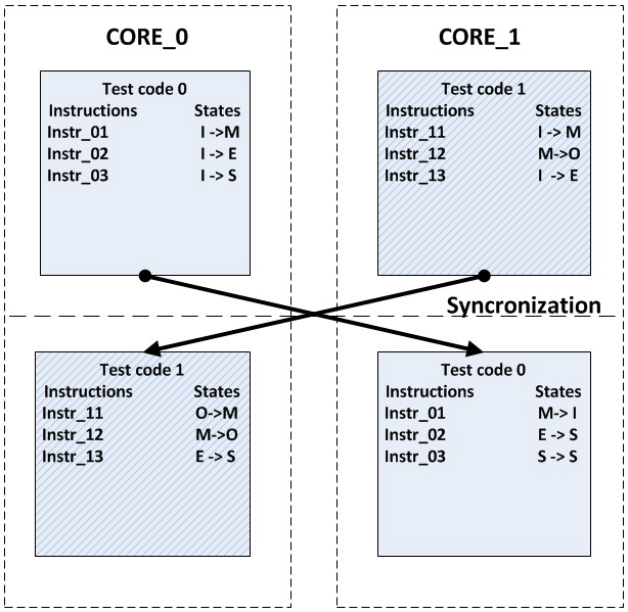

Fig.5. Cross-rerun test structure

The first half of the test at this stage can give the same level of MOESI protocol state machine coverage as the complete tests from the previous stage. However, after the interchange of program code (the second half of the test), the reachable state space can be expanded by the MOESI transitions induced by requests from another core.

F. Interleaved memory structure

A whole class of potential errors related to the simultaneous access of both cores to the same memory area is overlooked because of the approximations made in the previous paragraphs. One possible way to solve this issue is to use interleaved memory structure.

In order to ensure the completeness of the cache coherence protocol verification, it is necessary to implement the possibility of simultaneous access to one cache line by both cores in the test system. For this purpose, a periodic mask with a cell size smaller than the cache line size is superimposed to the selected memory area. In this case, one core gets access to all odd elements, and the other one — to even.

An important consequence of this organization of data in memory is the fact that both cores have access to the same cache line at the same time and they are able to write and read, without violating the integrity of the data (accesses occur in different bytes).

Every shared area, to which more than one core is allowed to write, will contain deterministic values because of the described data separation. Thus, all data in that area are predictable at any time.

Special read-only and write-only areas are created to test the remaining uncovered group of situations involving simultaneous (within multiple clock cycles) accesses to the same address in memory by both cores. The load and store operations are being executed uncontrolled within these areas due to the asynchronous behavior of the memory access threads, working with the shared data (race condition situation). Such requests should not be checked since they are aimed solely at finding pipeline stalls (deadlock).

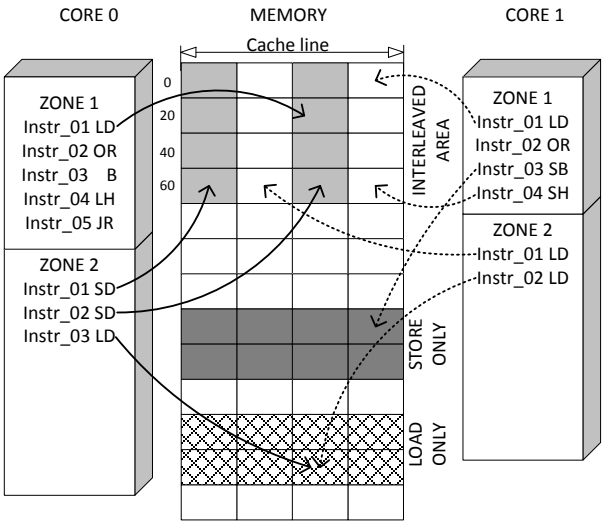

Fig. 6. Memory allocation for interleaved memory structure

In the example above (figure 6), both cores share memory space, but one core will have access only to even double words, and the other will have access to odd ones. Thus, tests can cover a much larger number of difficult to achieve situations which potentially can lead to errors in the memory subsystem. An example of address allocation in test templates might look like this:

Valid addresses for core 0: XXXX0-XXXX7,

Valid addresses for core 1: XXXX8-XXXXF.

When using the proposed memory allocation model, it is necessary to exclude cache-memory log files from comparison with corresponding ISS logs because it is impossible to guarantee the same sequence of requests. At the same time, the obviously correct memory state and general-purpose registers makes it possible to find errors in the RTL model with a high probability.

All mentioned above methods are simultaneously applied in the upgraded test system, as well as the interleaved regions partitioning techniques can be changed several times during test execution.

V. Tests quality analysis

A functional coverage metric was defined to evaluate the quality of the created pseudo-random tests. It is based on the state space, built on a combination of test situations, which are set by chosen events. The type of operation, hit or miss in the cache memory of any level, the replacement of modified line in the cache, the types of MOESI state machine transitions and multicore architectural features are examples of the events being used to direct test situations.

Examination of microarchitecture events coverage, data coverage, instruction set coverage and data coherency coverage is required to ensure proper validation [9].

With the help of heuristic analysis, the concept of test coverage is introduced and reachable maximum value is determined as corresponding to 100% of the test coverage.

Functional coverage metric introducing allows to quantify the degree of testing works completion, as well as to assess the quality of the tests created by the generator.

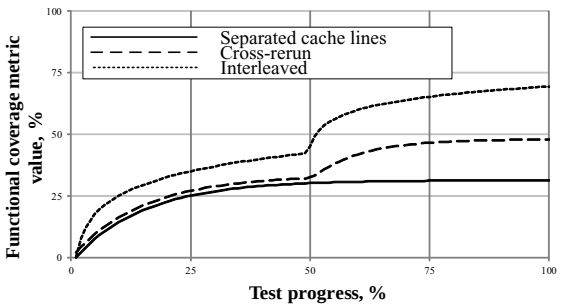

Fig. 7. Graphical representation for functional coverage

Figure 7 shows the growth of functional coverage during test execution for the three test construction techniques described above. A sharp increase in the middle of the tests (50% on the abscissa axis) is due to the fact that at this point the cross-rerun stage begins.

VI. Conclusion

The discussed approach was successfully applied to the verification of the RTL model of dual-core microprocessor with SMP developed in SRISA. The method made it possible to find the majority of memory consistency bugs and pipeline stalls.

The capabilities of the proposed techniques are proven to allow the directed random tests to cover highly sophisticated scenarios that would be very hard to generate if the generator is configured to produce only random instruction sequences.

Additionally, the approach was initially considered to be the first stage of RTL model verification, but the possibilities of the approach are also of interest for testing the model at the later stages of its design and functional maturity.

The testing process begins with creating simple random tests that check the MOESI coherence protocol. The advanced random testing method based on the usage of proposed interleaved memory structures is developed to increase the probability of finding rare and hard to detect bugs in the memory subsystem.

References

[1] KHisambeev I.SH., CHibisov P.A. Ob odnom metode postroeniya metrik funktsional'nogo pokrytiya v testirovanii mikroprotsessorov (On a method for constructing functional coverage metrics in microprocessor testing) // Problemy razrabotki perspektivnykh mikro- i nanoehlektronnykh sistem. Sbornik trudov/M.: IPPM RAN, 2014. S. 63-68 (in Russian).

[2] Grevtsev N.A., KHisambeev I.SH., CHibisov P.A. Issledovanie sposobov povysheniya ehffektivnosti stokhasticheskogo testirovaniya modelej mikroprotsessorov (Methods to improve efficiency of microprocessor model stochastic tests) // Problemy razrabotki perspektivnykh mikro- i nanoehlektronnykh sistem (MES'2016) (in Russian).

[3] Sorin, D. A Primer on Memory Consistency and Cache Coherence / Morgan & Claypool, 2012. - 210 p.

[4] Kamkin A.S., Burenkov V.S. Metod masshtabiruemoj verifikacii PROMELA-modelej protokolov kogerentnosti ke`sh-pamyati (A method for scalable verification of PROMELA models of cache coherence protocols) // Problemy` razrabotki perspektivny`x mikro- i nanoe`lektronny`x sistem (MES). 2016. №2. S. 54-60 (in Russian).

[5] Velesevich E. A. Obnaruzhenie i ocenka kolichestva promaxov kogerentnosti na osnove veroyatnostnoj modeli (Number of coherence misses detection based on the probabilistic model), Trudy` ISP RAN, 27:4 (2015), 39-48. (in Russian).

[6] T. Liu et al., "PREDATOR: Predictive false sharing detection", PPoPP 2014.

[7] Tongping Liu , Xu Liu, Cheetah: detecting false sharing efficiently and effectively, Proceedings of the 2016 International Symposium on Code Generation and Optimization, March 12-18, 2016.

[8] Smirnov A.V., CHibisov P.A. Generator testov dlya proverki kogerentnosti kehsh-pamyatej mnogoyadernyh mikroprocessorov (ristretto) (Random test generator for multicore microprocessor cache coherence verification (ristretto)) // Problemy razrabotki perspektivnyh mikro- i nanoehlektronnyh sistem. 2018. Vypusk 2. S. 31-38. (in Russian).

[9] Satish Kumar Sadasivam, Sangram Alapati, Varun Mallikarjunan: Test Generation Approach for Post-Silicon Validation of High End Microprocessor. DSD 2012: 830-836.