DOI: 10.15514/ISPRAS-2019-31(3)-6

Test environment for verification of multi-processor memory subsystem unit

D.A. Lebedev, ORCID: 0000-0002-9244-4949 <lebedev_d@mcst.ru>
M.V. Petrochenkov, ORCID: 0000-0001-7384-9732 <petroch_m@mcst.ru>
MCST, 1, Nagatinskaya st., Moscow, 117105, Russia

Abstract. State of the art microprocessor systems usually include complex hierarchy of a cache memory. Coherence protocols are used to maintain memory consistency. An implementation of memory subsystem in HDL (hardware description language) is complex and error-prone task. Ensuring the correct functioning of the memory subsystem is one of the cornerstones of a modern microprocessor systems development. Functional verification is used for this purpose. In this paper, we present some approaches for verification of memory subsystem units of multi-core microprocessors. We describe characteristics of memory subsystems that need to be taken into account in the process of verification. General structure of test environment for stand-alone verification of memory subsystem units is presented. Classification of checking model types and their advantages and disadvantages are described. The approach of construction of a standalone verification environment using Universal Verification Methodology (UVM) is presented in the paper. Restrictions that should be taken into account when verifying memory subsystem unit are listed. The generation stimulus algorithm stages are presented. Method of using “hints” from design under verification to eliminate nondeterminism is used in the implementation of checking module. We review several other techniques for checking the correctness of memory subsystem units, which can be useful at different stages of project development. A case study of applying the suggested approaches for verification of Home Memory Unit of microprocessors with Elbrus architecture is presented. Classification of detected and corrected errors in different submodules of verified device is provided. Further plan of the test system enhancement is presented.

Keywords: multicore microprocessors; cache memory; coherence protocols; test system; model-based verification; stand-alone verification.

For citation: Lebedev D., Petrochenkov M. Test environment for verification of multi-processor memory subsystem unit. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 3, 2019. pp. 67-76. DOI: 10.15514/ISPRAS-2019-31(3)-6

Тестовое окружение для верификации блока подсистемы памяти многопроцессорной системы

Д.А. Лебедев <lebedev_d@mcst.ru>
М.В. Петроченков <petroch_m@mcst.ru>
АО «МЦСТ», 117105, Россия, г. Москва, ул. Нагатинская, д. 1, стр.23

Аннотация. Современные микропроцессорные системы обычно включают сложную иерархию кэш-памяти. Протоколы когерентности используются для поддержания согласованности памяти. Реализация подсистемы памяти на языке описания аппаратуры является сложной и подверженной ошибкам задачей. Обеспечение корректного функционирования подсистемы памяти, является одной из важнейших задач в процессе разработки современных микропроцессорных систем. Для этого используется функциональная верификация. В данной работе представлены некоторые подходы к верификации блоков подсистем памяти многоядерных микропроцессоров. Описаны характеристики подсистем памяти, которые необходимо учитывать в процессе верификации. Представлена общая структура тестовой системы для автономной верификации блоков подсистемы памяти. Приведена

классификация типов проверяющих моделей, их преимущества и недостатки. В статье представлен подход к построению автономного окружения для верификации с использованием универсальной методологии верификации (UVM). Перечислены ограничения, которые следует учитывать при проверке блоков подсистемы памяти. Представлен алгоритм генерации входных стимулов. Для устранения неопределенности текущего состояния верифицируемого устройства в проверяющем модуле используется метод анализа «подсказок». Рассмотрен ряд других методов проверки корректности блоков подсистемы памяти, которые могут быть полезны на различных этапах разработки проекта. Представлен пример применения предложенных подходов к верификации блока HMU микропроцессоров с архитектурой Эльбрус. Приведена классификация обнаруженных и исправленных ошибок в различных подмодулях верифицируемого устройства. Представлен дальнейший план совершенствования тестовой системы.

Ключевые слова: многоядерные микропроцессоры; кэш память; протоколы когерентности; тестовая система; верификация на основе моделей; автономная верификация

Для цитирования: Лебедев Д.А., Петроченков М.В. Тестовое окружение для верификации блока подсистемы памяти многопроцессорной системы. Труды ИСП РАН, том 31, вып. 3, 2019 г., стр. 67-76 (на английском языке). DOI: 10.15514/ISPRAS-2019-31(3)-6

1. Introduction

With the development of microprocessor technology and growth of the number of computational cores and CPUs in systems processor performance increases rapidly. Unfortunately, the speed of memory access is not growing as fast as the speed of the processor [1]. Thus, one of the biggest bottleneck elements become the memory subsystem. To level the difference in speed, designers of microprocessor systems implement a complex memory subsystem that includes cache hierarchy. State of the art microprocessor systems usually include 3-4 levels of cache memory. This approach is able to reduce the number of accesses to main memory, and, therefore, reduce memory access instructions average execution time.

In the multicore systems if multiple cores are simultaneously allowed to contain copies of a single memory location, the problem of maintaining memory consistency arises. A mechanism must exist to ensure that all copies remain consistent when the contents of that memory location are modified. Coherence protocols support such mechanism. Usually we have higher-level caches shared between cores and lower-level caches served by a single core. Complex systems that combine several multi-core processors may also have cache memory to speed up other processors' access to their memory.

A large number of processors and processor cores and complexity of system data exchange organization makes coherence protocol very complicated. An implementation of cache coherence protocol is a complex and error-prone task. Errors of this kind are critical and difficult to detect on system-level verification. Thus, a memory subsystem and implementation of coherence protocols in HDL (Hardware Description Languages) models must be thoroughly verified [2].

There are two main methods for verification of memory subsystem: a simulation-based verification and formal verification [3]. Formal verification is used to mathematically prove the correctness of a DUV (Device Under Verification) model with respect to its specification. It is widely known that main advantage of formal methods is their exhaustiveness. Many works are devoted to this method [4-6]. Disadvantages of these methods are complexity of development and high specification requirements. Simulation-based methods are not exhaustive, but they can be applied at earlier stages of development and they are much simpler.

Verification of a memory subsystem, as a part of whole microprocessor, can be provided by means of system verification [7]. However, it is essential to mention that some of the components of a memory subsystem are invisible from the point of view of a testing program and it is hard to recreate necessary conditions for verification with proper quality. To overcome these drawbacks, a stand-alone verification of the memory subsystem is usually used.

There are a number of methods to implement a standalone functional verification of a memory subsystem. One of them is C++TESK Testing ToolKit created in ISP RAS [8]. It is an open-source

C++ based toolkit intended for automated functional testing of RTL (HDL) models of digital hardware (in Verilog and VHDL). The tool included a library of C++ classes and macros that define facilities for all parts of a verification environment. Some of disadvantages of this tool are high complexity of the application and needs documentation and checking reference model high accuracy.

Another tool name is Alone-env created in the MCST. The Alone-env provides a wrapper-class over Verilog description of the verified module. The Alone-env too has some disadvantages: the lack of collecting coverage means, high requirements for the checking reference model and the inability to reuse the test system.

Nowadays the most widespread verification methodology is Universal Verification Methodology (UVM). This is a standard verification methodology from the Accellera Systems [9]. UVM designed to enable creation of robust and reusable testbenches and their components. UVM is a class library helps to bring much automation to the SystemVerilog language. Disadvantages of UVM is learning curve is very high for new users and it takes a lot of code to create basic UVM testbench classes. Nevertheless, our team already have a number of test systems, basic classes and libraries written and debugged. Therefore, we choose UVM for developing the stand-alone verification environment of memory subsystem modules.

The rest of the paper is organized as follows. Section 2 reviews the existing techniques for standalone verification of the memory subsystem. Section 3 describes a case study suggests an approach to the problem of developing test system. Section 4 describes additional used approaches. Section 5 reveals results and Section 6 concludes the paper.

2. Standalone verification methods of memory subsystem

In a stand-alone verification we implement test system that allows to select a single part of the whole system and examine its behavior in the test environment that behaves in a way similar to the “real” system. Correct mechanisms of interaction with DUV are defined in its specification. One of the main advantages is that it is easier to explore edge and corner cases in the verified module.

When verifying a part of the memory subsystem with included cache, we need to take into account some features while developing the test system:

- it consists of cache lines that are fixed size blocks used to transfer data between two nodes of the system;
- logic to locate and transfer requested data;
- cache line also hold service information;
- may be several requesters which work with different cache lines
- if two or more requesters want to refer to the same cache line such request have to be serialized and completed in the same order as they received;
- controller support some of implementations of a coherence protocol;
- due to the limited amount of a memory, one of the data eviction algorithms is implemented.

Test environment (or testbench) for verifying the memory subsystem usually includes:

- generator of input stimulus;
- checker of collected reactions correctness;
- module collecting coverage information.

Generator of input stimuli is responsible not only for primary requests that perform operations with memory, it also collects reactions from verified device and generate answers from test environment - secondary requests. Generalized scheme of test environment shown on Fig. 1. Generation of stimuli can be simplified by using TLM [10] (Transaction Level Modeling) to communicate with DUV. TLM allows focusing more on the functionality of the data transmission and less on its actual implementation.

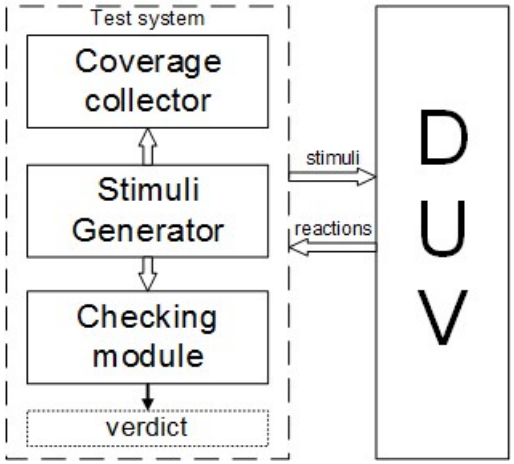

Fig 1. Generalized scheme of test environment

If the verified device has a complex structure and many states, the easiest way to check correctness of reactions is building the separate checking module. Checking module is based on the external to the test environment reference model usually written in high-level language (C, C++ or some specific languages for verification of hardware, such as SystemVerilog, SystemC or «e»). All requests and reactions from the verified device sent to the checking module where then made a conclusion about the correctness of the behavior.

The reference models could be divided into three types: cycle-accurate, discrete-event with time accounting and event models [11]. First two of them require a very accurate specification. It is hard to support design changes that happen very often on the first steps of the development. Furthermore, the similarity of the implementations of the model and the DUV can lead to duplication of errors. To check correctness of memory subsystem, it is reasonable to use event models because they require less time to develop and more flexible for changes of the design. Data interchange of the test system with reference model occurs instantly by calling appropriate functions. For some devices, there are several correct scenarios of the operation for the same input stimuli. We call those devices non-deterministic. There are two methods allowing using behavioral event models for verification of these devices [12].

The first method is dynamic refinement of transaction level model. A general approach is as follows. When a reference model gets a request and there are several possible ways to react to the request, the model creates additional instances and executes the requests in each of them. Then the models are waiting for the reactions from the device under verification. The reaction contents service information (such as a response type, a direction of sending, etc) which helps to exclude impossible states. Absence of suitable state for reaction signals about an error. The sign of a successful completion is comparison all the reactions of the DUV and removal of all unnecessary states. The complexity of this approach is that the number of possible states potentially grows exponentially with a number of stimuli. However, this method can be implemented efficiently for memory subsystem units because all requests to a single cache line are serialized and requests to different cache lines are independent.

Second method is identification of a single correct state using hints from the verified device or a "gray box" method. This method replaces usual “black box” method. When we cannot predict the “real” sequence of interactions, we access inner interfaces of the verified device. Information from these interfaces have to be transferred to the test environment and helps to determine a single

possible execution scenario and eliminate nondeterminism. This method imposes additional requirements to the device specification, but, as a result, it is quite simple to implement.

Coverage collection module extract information of functional code coverage. This information is used to identify unimplemented test cases and helps to improve stimuli generation by adding new test scenarios.

3. Using gray box approach for verification of home memory unit

Home Memory Unit (HMU) is a part of memory subsystem of 16-core "Elbrus" microprocessor responsible for the coherent and non-coherent access to the RAM from different requesters. HMU contains a global directory (MOSI protocol of coherence), which monitors the requests of other processors to its memory and a DMA directory which is a full copy of the DMA caches of all processors (supports the extended coherence protocol MOI). Total volume of the directory in HMU is 2.5 MB, size of entry of a cache line is 80 B, number of banks – 128, bank associativity – 16. Main functions of HMU include:

- serialization of all requests to RAM;
- reduction of coherence traffic and access time to RAM;
- support of interprocessor coherence.

Test system for stand-alone verification of the coherence protocol implementation and other functionality of HMU based on UVM. UVM helps to generate pseudo-random constrained input requests to cover possible states of the verified device.

We have to note some restrictions for generation primary, secondary stimuli and answers. The first of them is limited amount of space in input buffers. Due to process of verification, it is important not to lose some data. When generating random system settings, it is necessary to take into account that some setup combinations may be incorrect and lead to errors. There are several types of requesters in the test system. Each of them has special identifier and a set of possible operation codes. The specific implementation of the coherence protocol also imposes restrictions on the used operation codes. Sending an inappropriate operation code may result in undefined results. Address generation is also a non-trivial task. The address have to fit the interleaving conditions. In addition, each requester have to wait for the completion of previous request when working with same cache line.

Stimuli generation is divided into several stages:

1. randomization of device configuration registers. This allows to switch different ways for handling requests and determine request routing;
2. creating list of addresses for current configuration of device with respect to routing setup;
3. choosing random requester and cache line address;
4. checking cache line availability and presence of resources needed for request transfer;
5. choosing random operation code constrained by the current state of cache line;
6. sending primary request, collecting reactions from the device under verification, sending secondary requests;
7. collecting all of necessary reactions and completion of current request;
8. transferring transaction information to checking module.

To simplify handling of requests and reactions we create models of each used cache line. Model of cache line is an object that stores information about primary request, collected reactions, data and some auxiliary functions. For generation of correct requests we created an associative memory storing current states for each cache line. The choice of the next request type is made according to the limitations imposed by the coherence protocol and the current state of the cache line. For example, one of these rules is there cannot be two requesters in a modified (M) state for a single cache line. Another feature, which was necessary to pay attention, is that one address tag corresponds

to two-neighbor cache lines information. This mechanism allows increasing the ratio of the directory coverage (the ratio of the cache memory covered by the directory to the cache memory of the directory).

As noted before, there are two ways of building checking module. The choice of «gray box» method is determined by following sources of a nondeterminism inherent to HMU:

- HMU contains two input queues of primary requests what means exponential growth of possible device states size (2n+m, where n, m – number of requests inside input queues);
- cache eviction algorithm in the global directory.

HMU has two cache memories responsible for different functions of a memory subsystem: the global and DMA directories. The global directory has information only about data belonging to the own processor and used by other processors. Along with that there is no information about presence of this cache line in cache of own processor. Such information located in the L3 cache. DMA writes are also coherent requests. For a fast and correct handling of DMA writes sent by DMA controllers the special DMA cache directory is present inside HMU. This cache directory supports extended coherence protocol MOI with substates. Its main function is processor notification about cache lines captured by DMA controllers and storing their states.

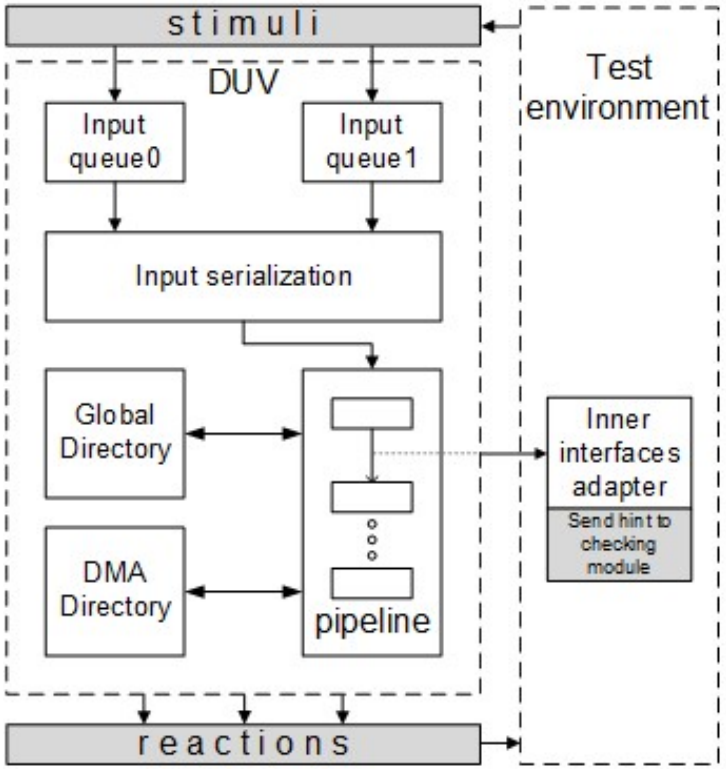

Fig 2. Simplified version of the test environment using the "gray box" method.

The device under verification connected with other parts of the system by means of network-on-chip and has two input channels for primary requests. All generated primary requests are sent to the DUV and the checking module simultaneously. The checking module (implemented in C++)

receives requests and reactions from the verified device by means of DPI (Directed Programming Interface). Using of the DPI is necessary to match the types and classes of the test environment written in SystemVerilog hardware description language with the reference model interface functions. Inside checking module, all requests are received into two queues. Requests to the same cache line can get into the different queues. It is impossible to predict which of these requests will be handled first. Getting a sequence hint from the device under test eliminates the nondeterminism of the current state. Stable and well-described inner interfaces to the point where all request are serialized, made it possible to build simple checking module in the short time. In a similar way, an access to the eviction mechanism interface was obtained. In addition, the checking module and its behavior model may be modified if it will be needed in the future projects. Structure of the test environment with proposed “gray box” method shown in fig. 2.

*4. Additional verification methods*Management of transaction flow

To check if the verified device operates correctly, it is necessary to achieve multiple edge and corner cases. This involves filling all input and output primary and secondary requests queues, delaying some necessary types of answers and blocking of transactions from some modules [13]. HMU supports the credit-exchange mechanism, which indicates the devices ability to accept certain type of requests. We added the special configuration module that randomizes time delays of sending requests and credits. Management of delays setup allows to create different test scenarios with overflowing requests and responses buffers. This mechanism helps to detect livelocks and deadlocks. These types of system behavior are hard to implement during system testing.

4.2 Applying assertions

SystemVerilog Assertions (SVA) is an important subset of SystemVerilog [14]. The assertions are used to specify the behavior of the system. The assertions work as follows: we add some piece of verification code to the test system that monitors a design implementation for compliance with the specifications. In addition, the assertions can be used to flag that input stimuli do not conform to assumed requirements. In the beginning of the project, it may help to find more bugs and locate them faster.

In HMU verification process the assertions are used for checking for uncertain and unconnected states of signals. Usage of coherence protocols in the DUV involves certain restrictions on the stimuli generation and the state of the cache lines for different requesters. Thus, additional function of the assertions, which was used in the test system, is detection of the discrepancy between coherence protocol specification and generated requests types in the certain cache lines states. The disadvantage of this approach is the limitation of the properties of the verified device that can be checked by assertions.

4.3 On-the-fly ECC errors insertion

ECC bits are stored together with the state of the cache line. Special submodules of HMU encode the data written to the RAM and decode data read from RAM. Using ECC bits allows to detect single, double, parity errors and to correct single errors. This mechanism is a source of potential errors in the device. The special module with flexible configuration was developed to insert single and double errors. This module is managed by the test system. Frequency and type of error insertion can be regulated. Detecting and correcting ECC errors additionally loaded computing logic of verified device.

5. Results

The approaches described in this paper were applied for standalone verification of Home Memory Unit of 16-core and 2-core with 6 integrated graphic boosters microprocessors with “Elbrus” architecture.

There are some difference in operating with memory subsystem in the microprocessors. The 16-core microprocessor’s HMU has a global directory and DMA directory, sends coherent requests with accordance to the state in the global directory, and collects short coherent answers and coherent answers with data for write operations. For read operations, requester (DMA or L3 cache) collects all the answers.

The 2-core microprocessor does not have a global directory in HMU but includes DMA directory. HMU provides inter-core coherence. Coherence requests are sent broadcast to the cores and DMA. HMU also collects all the answers for write operations and for read operations only when requester is not DMA. Integrated graphic boosters are not snooped.

Due to the specificity of the test system construction, some part of MC controller (the MC adaptor) was also added to the verified system. Generator of responses from MC controller with randomized setups was also developed.

In the process of the standalone verification of the Home Memory Unit we found 28 errors that have not been found by other means of verification. All errors were corrected. The distribution of the number of bugs in different subsystems of the HMU are presented in Table 1. Code and functional coverage was carried out and 94% coverage was extracted. Total result indicates about effectiveness of the proposed methods of standalone verification.

Table 1. Types of found bugs and its quantity

Type of bugs	**Number of bugs**
Coherence protocol implementation	21
Configuration registers	2
Parity checker	1
Performance improvement	2
MC adaptor	3
Total:	28

6. Conclusion and directions for future work

Memory subsystem is one of the most important parts of microprocessors. Its parts that support coherence protocols are especially complicated and error-prone. Verification of these types of devices is time-consuming and labor-intensive work. The stand-alone verification designed to simplify this task. The approaches mentioned in this paper can be applied for stand-alone verification memory subsystem parts regardless of their implementation.

The proposed approaches have been applied in the verification of the Home Memory Unit as a part of multi-core microprocessor memory subsystem with “Elbrus” architecture developed by "MCST". Test environment and test scenarios made it possible to detect and correct a number of logical errors that were not detected by other verification methods.

In the future, it is planned to adapt the test environment for the forthcoming projects and possible changes in coherence protocols.

References / Список литературы

[1]. Hennessy J.L., Patterson D.A. Computer Architecture: A Quantitative Approach. Fifth Edition. Morgan Kaufmann, 2012. 857 p.

[2]. A. Kamkin, M. Petrochenkov. A Model-Based Approach to Design Test Oracles for Memory Subsystems of Multicore Microprocessors. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 3, 2015, pp. 149-160. DOI: 10.15514/ISPRAS-2015-27(3)-11.

[3]. W.K. Lam. Hardware Design Verification: Simulation and Formal Method-Based Approaches. Prentice Hall, 2005, 624 p.

[4]. Burenkov V.S. A Technique for Parameterized Verification of Cache Coherence Protocols. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 231-246. DOI: 10.15514/ISPRAS-2017-29(4)-15.

[5]. Ivanov Lubomir and Nunna R. Modeling and verification of cache coherence protocols. In Proc of the 2001 IEEE International Symposium on Circuits and Systems, vol. 5, 2001, pp. 129-132. DOI: 10.1109/ISCAS.2001.922002.

[6]. P.A. Abdulla, M.F. Atig, Z. Ganjeiy, A. Reziney, and Y. Zhu, Verification of cache coherence protocols wrt. trace filters. In Proc. of the 15th Conference on Formal Methods in Computer-Aided Design, pp. 9-16.

[7]. I.A. Stotland, V.N. Kutsevol, A.N. Meshkov. Problems of functional verification of Elbrus microprocessor L2-cache. Issues of radio electronics, ser. EVT, no. 1, 2015, pp. 76-84 (in Russian) / Стотланд И.А., Куцевол В.Н., Мешков А.Н. Проблемы функциональной верификации кэш-памяти второго уровня микропроцессоров с архитектурой «Эльбрус». Вопросы радиоэлектроники, сер. ЭВТ, 2015, no. 1, стр. 76-84.

[8]. C++TESK Testing ToolKit review. Available at: https://forge.ispras.ru/projects/cpptesk-toolkit, accessed 12.06.2019.

[9]. Standard Universal Verification Methodology. Available at: http://accellera.org/downloads/standards/uvm, accessed 12.06.2019.

[10]. Kamkin A., Chupilko M. A TLM-based approach to functional verification of hardware components at different abstraction levels. In Proc. of the 12th Latin-American Test Workshop (LATW), 2011, pp. 1-6.

[11]. Averill M. Law, W. David Kelton. Simulation Modelling and Analysis. McGraw-Hill Education, 3rd edition, 2000, 784 p.

[12]. Petrochenkov M., Stotland I., Mushtakov R. Approaches to Stand-alone Verification of Multicore Multiprocessor Cores. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 3, 2016, pp. 161-172. DOI: 10.15514/ISPRAS-2016-28(3)-10.

[13]. Lebedev D.A., Stotland I.A. Construction of validation modules based on reference functional models in a standalone verification of communication subsystem. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 183-194. DOI: 10.15514/ISPRAS-2018-30(3)-13.

[14]. 1800-2017 - IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language. Available at: https://standards.ieee.org/standard/1800-2017.html, accessed 22.06.2019.

Информация об авторах / Information about authors

Дмитрий Алексеевич ЛЕБЕДЕВ работает в АО МЦСТ, Москва, Россия. Он защитил диплом специалиста в области электроники в 2014 г. в НИЯУ МИФИ. Область его исследовательских интересов включает исследование методов верификации контроллеров связи, систем прерываний, устройств подсистемы памяти с поддержкой протоколов когерентности.

Dmitry Alexeevitch LEBEDEV works in AO MCST, Moscow, Russia. He earned a specialist in electronics diploma in 2014 at MEPhI. His area of research interests includes the verification methods study of communication system controllers, interrupts systems and memory subsystems devices with support of protocols of coherence.

Михаил Владимирович ПЕТРОЧЕНКОВ работает в АО МЦСТ, Москва, Россия. Он получил степень магистра в 2012 г. МФТИ. Область его научных интересов включает исследование методов верификации устройств подсистемы памяти и контроллеров связи.

Mikhail Vladimirovich PETROCHENKOV works in AO MCST, Moscow, Russia. He received his master's degree in 2012 from MIPT. Area of his scientific interests includes the verification methods study of memory subsystemsdevices and communication controllers.