UDC 004.318

Siberian Journal of Science and Technology. 2017, Vol. 18, No. 2, P. 307–311

THE SYSTEM OF SOFTWARE-DRIVEN VERIFICATION OF NETWORK IP-CORES IN A REFERENCE SYSTEM-ON-CHIP

A. V. Shahmatov, E. S. Lepeshkina*, V. Kh. Khanov

Reshetnev Siberian State University of Science and Technologies
31, Krasnoyarsky Rabochy Av., Krasnoyarsk, 660037, Russian Federation
*E-mail: klepka1111.93@mail.ru

The article presents the application of network Intellectual Property cores (IP cores) software-driven verification method for network infrastructure devices in the system-on-chip microprocessor (SoC) used as verification environment. The SoC used for verification is a reference system since it consists of previously fully verified and approved IP cores which interact in this system correctly and accurately. Software of a reference system generates test inputs and processes responses to them which are received from a verified device. Conclusions of executed or unexecuted tests are generated on the basis of the expected results. A set of expected results of input action is a reference model of a verified IP core.

General architecture of a verification system of a network device IP core has a form of a classic test loop. The variants of verification architecture given depend on the type of a verified network device: an individual network codec, a network protocol controller or a network switch. The presented architectures show the simplicity of software-driven verification. The test environment naturally results from the reference SoC model and test software developed in such high-level programming language as C/C++.

When the software-driven verification of an IP core takes place in reference SoC environment, the test software consists of two types of tests: directed tests and restricted-random tests. Successive use of both the given types of tests and typical scenarios of network devices interaction which include request-reply packages transmission between network nodes provides high coverage of a verified IP core with test situations. To check fault tolerance function it is supposed to use the scenarios of network devices interaction in conditions of possible faults made by predetermined introducing of errors into packages transmitted over the network connections. Program tests which are developed and proved during the IP core model verification are completely ready to be used in hardware SoC prototype including the given IP core in the programmable logic device.

The presented approach to functional verification was used for IP cores testing in SpaceWire network infrastructure: a fault tolerance codec, a RMAP protocol controller and a routing switch.

Keywords: Intellectual Property cores, functional verification, software-driven verification, reference SoC, verification architecture of intellectual property cores.

Сибирский журнал науки и технологий. 2017. Т. 18, № 2. С. 307–311

СИСТЕМА ПРОГРАММНО-УПРАВЛЯЕМОЙ ВЕРИФИКАЦИИ СЕТЕВЫХ СЛОЖНОФУНКЦИОНАЛЬНЫХ БЛОКОВ В ЭТАЛОННОЙ СИСТЕМЕ НА КРИСТАЛЛЕ

А. В. Шахматов, Е. С. Лепёшкина*, В. Х. Ханов

Сибирский государственный университет науки и технологий имени академика М. Ф. Решетнева
Российская Федерация, 660037, г. Красноярск, просп. им. газ. «Красноярский рабочий», 31
E-mail: klepka1111.93@mail.ru

Представлено применение метода программно-управляемой верификации сложнофункциональных блоков (СФ-блоков) устройств сетевой инфраструктуры в микропроцессорной системе на кристалле (СнК), используемой в качестве среды для проведения верификации. Используемая для верификации СнК является эталонной, так как она состоит из ранее полностью верифицированных и апробированных СФ-блоков, которые гарантированно правильно и безошибочно в ней взаимодействуют. Программное обеспечение эталонной системы вырабатывает тестовые воздействия и обрабатывает реакции на них, получаемые от верифицируемого устройства. Заключения о выполнении или невыполнении тестов вырабатываются исходя из ожидаемых результатов. Совокупность ожидаемых результатов на входные воздействия составляет эталонную модель верифицируемого СФ-блока.

Общая архитектура системы верификации СФ-блока сетевого устройства имеет вид классической тестовой петли. Приведены варианты архитектуры верификации в зависимости от вида верифицируемого сетевого устройства: отдельный сетевой кодек, контроллер сетевого протокола или сетевой коммутатор.

Представленные архитектуры демонстрируют простоту программно-управляемой верификации. Окружение тестирования получается естественным образом из модели эталонной СнК и тестового программного обеспечения, разработанного на высокоуровневых языках программирования, обычно C/C++.

При программно-управляемой верификации СФ-блока в среде эталонной СнК тестовое программное обеспечение состоит из двух видов тестов: направленных и ограниченно случайных. Последовательное использование данных видов тестирования, а также типовых сценариев взаимодействия устройств в сети, заключающихся в посылках серий запросных и ответных пакетов между узлами, обеспечивает высокое покрытие верифицируемого СФ-блока тестовыми ситуациями. Для проверки функций отказоустойчивости предлагается использовать сценарии взаимодействия узлов в условиях возможных сбоев, вносимых путем преднамеренного внесения ошибок в передаваемые через сетевые соединения пакеты. Программные тесты, разработанные и отлаженные в процессе верификации модели СФ-блока, полностью готовы к применению в аппаратном прототипе СнК, включающей данный СФ-блок, в устройстве программируемой логики.

Представленный подход к проведению функциональной верификации был использован для тестирования СФ-блоков для инфраструктуры сети SpaceWire: отказоустойчивого кодека, контроллера протокола RMAP и маршрутизирующего коммутатора.

Ключевые слова: сложнофункциональные блоки, функциональная верификация, программно-управляемая верификация, эталонная система на кристалле, схемы верификации сетевых сложнофункциональных блоков.

Introduction. Functional verification (FV) is intended to serve for the validation of a designed device or an intellectual property core (IP-core) to set functional specifications [1]. The FV significance rises together with the increase of designed devices complexity. The FV complexity also increases reaching 70 % of all development work [2]. Usually FV is performed on models of a designed device and then models are converted into a hardware prototype which becomes a base for creating of a target digital device.

FV is divided into 2 large groups of methods: methods based on dynamic modeling or simulation (Simulation-Based Verification, SBV) of a model of Register-Transfer Level (RTL) [1; 2]; and Formal Functional Verification (FFV) methods based on the representation of a verified IP-core as other models that are not RTL, which are called formal and used for logical fault finding [3]. SBV is a traditional approach still having high priority because a target device is synthesized from RTL-model created by Hardware Description Language (HDL).

FFV methods actively developed in recent years are additional and serve to avoid faults at early project stages, to detect faults not identified by SBV-methods; to carry out formal provability of the errorless operation of a designed device. There are also so-called hybrid methods using the combinations of different approaches. Hereinafter the SBV-methods are used in the paper.

In general terms SBV-method consists of creating the test environment where RTL-model of Device Under Test (DUT) is simulated. The test environment generates test inputs (stimuli) on the model in the form of different sequences of input signal sets and analyses responses, or model reactions, that is, changes in its state and its output values. The action described is provided by special programs – RTL-simulators.

At the present moment the simulation of complex digital devices such as microprocessors or devices connected to their internal bus is based on the Transaction Level Modeling (TLM) concept [1; 2]. In this case the stimuli are not specially-generated input signal sets but high-level interaction with DUT (bus exchange operations and input/output interface exchange, processor instructions, and network packages) converted into signals. DUT reactions to transactions are also converted into high-level data saved as response tracks. The given test environment creating simplifies the FV.

For test environment creating one can use both programming languages such as C/C++ or assembler and VHDL. The latter way belongs to autonomous IP-core verification and is inherent in all RTL-simulators. A new stage of its development is presented by such well-known methodologies as Open Verification Methodology (OVM) and Universal Verification Methodology (UVM) [4; 5]. To reduce the time spent on developing the test environment and creating stimuli of any complexity they use such an object-oriented system designing language as System Verilog and UVM class library.

Traditional programming languages are usually used for IP-core verification within an entire system. For example, a well-known approach for processor verification is Instruction Set Simulation (ISS) [6]. The stimuli are program tests of verifiable processor instruction executing and result monitoring. ISS is more flexible: it is used not only for system verification but also for autonomous one on condition of corresponding preparation as well as for hardware and software co-verification.

Such approach is generally called Software-Driven Verification (SDV) [7]. In special literature one can come across another name – Processor-Driven Verification (PVD) [8]. The approach consists of IP-core RTL-model connection to the full-function microprocessor system (MPS) model, for example system-on-chip (SoC), by the standard bus and interface and later verification by MPS program tests.

This paper presents the SoC system for software-driven IP-core verification, where IP-cores are interface devices.

The software-driven verification system. The overall architecture of IP-core verification system for network devices has the form of a classic test loop, when the interaction of a reference device and a tested one is checked under the control of an operating device which in this case is a SoC processor with required test software. A codec test loop is first performed in the model and for FPGA prototyping it is replaced by cable connection.

Depending on the type of network device to be verified – a network codec, a network protocol controller

or a network switch - the architecture of the verification system is modified.

Fig. 1 shows the architecture of network codec IP-core verification system (DUT on fig. 1). All the other SoC devices – a processor, an on-chip bus, and other devices not shown in the figure for simplicity – are reference ones in this case. It is supposed that reference IP-cores were previously fully verified and approved; correctness of their functioning is guaranteed; and they can be fully trusted. The given IP-cores can be self-developed, purchased as commercial products or obtained from information resources as open products, but in any case their correctness is guaranteed. The entire SoC system consisting of reference IP-cores which are guaranteed to correctly and unerringly interact in it is a reference system. In a reference system only one separate IP-core is verified at a time.

The scheme presented in fig. 1 is intended for verification of a new IP-core of a network codec, for example, its fault-tolerant version, by a reference IP-core of a network codec. The test inputs generated by test software are fed to a verified IP-core network codec from two directions: from an on-chip bus and a codec reference IP-core.

Verified IP-core responses to inputs impacts are also processed by test software. The conclusion about executed or unexecuted tests is based on the expected results. The cumulative expected result is a DUT reference model realized in a reference SoC.

The architecture of a verification system of a network protocol controller IP-core is presented in fig. 2. A network controller includes a reference network codec. The logic of a network controller is to be verified. The architecture of network switch verification is built the same way (fig. 3). The network switch has *n* reference network codecs equal to the number of switch ports. A separate network connection using the *n* network codecs of the reference SoC is created for every switch port. Only the network switch logic should be verified.

Presented architectures illustrate the simplicity of software-driven verification. Test environment naturally results from the reference SoC model and test software developed in a high-level programming language, usually C/C++.

Test software. According to general verification methodology, if software-driven IP-core verification takes place in SoC reference environment then the test software consists of two test types: a directed test and a restricted random test.

Directed software tests are designed manually. They are used for initial testing of a verified IP-core basic elements and its basic function as well as for checking of hard to formalize and rare events [6]. Besides, directed software tests allow recreating and checking the typical scenarios of network device interaction which consists of request-reply packages transmission between network nodes.

To check fault tolerance function the network device interaction scenarios can be practised in response to possible faults injected by predetermined fault introducing into packages transmitted over the network connections.

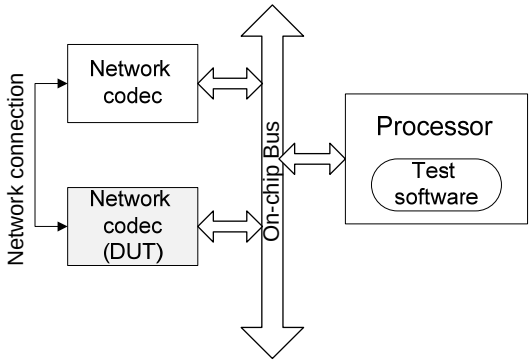

Fig. 1. The architecture of a network codec IP-core verification system

Рис. 1. Архитектура системы верификации СФ-блока сетевого кодека

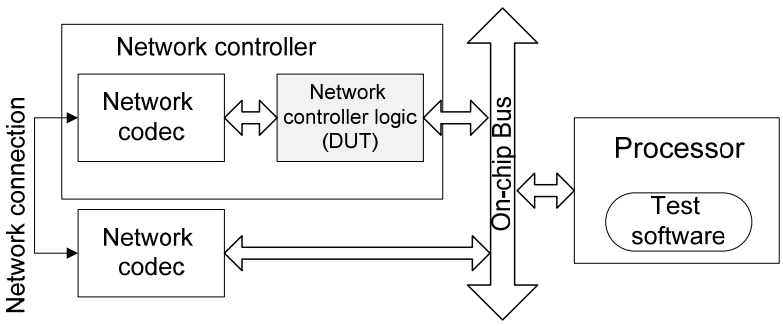

Fig. 2. The architecture of a verification system of a network protocol controller IP-core

Рис. 2. Архитектура системы верификации СФ-блока контроллера сетевого протокола

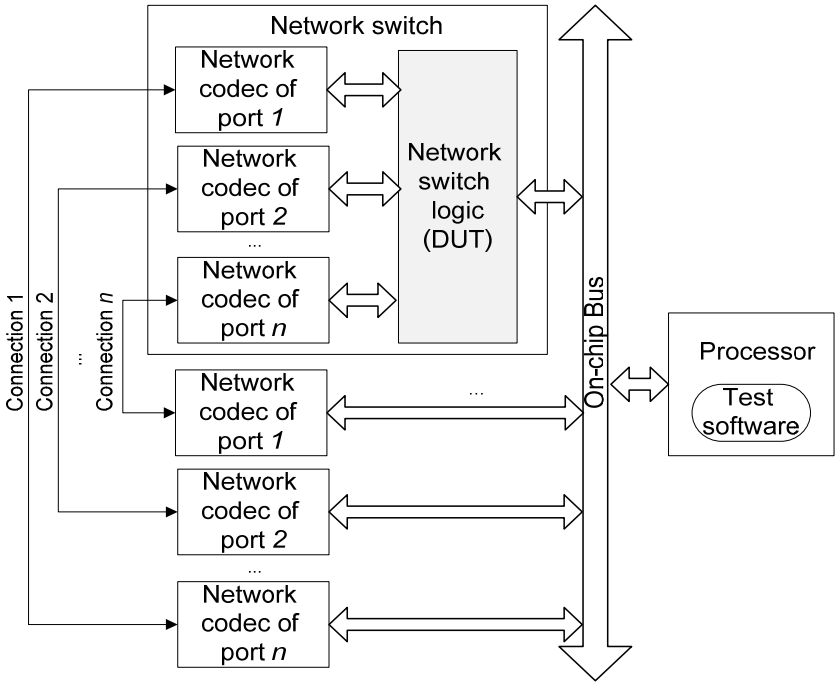

Fig. 3. The architecture of a verification system of a network switch IP-core

Рис. 3. Архитектура системы верификации СФ-блока сетевого коммутатора

The major part of detailed verification is performed by restricted random verification based on the object-oriented programming capability. The method of restricted random (stochastic) testing is that the test sequence is generated automatically according to a set template with parameterized pseudorandom selection of test body instructions and their arguments. The method key benefit is the opportunity to fully automate tests generation and start-up processes, as well as the comparison of tests results which is necessary for mass testing of complex IP-cores [9].

The advantage of the software-driven IP-core verification by reference SoC is naturalness and simplicity. There is no need to study specialized methodologies, language aids and verification libraries, and then with their help to create required verification environment and test inputs. In this case the test environment results naturally from a priori available reference SoC model created by a traditional designing language (VHDL, Verilog), and tests are written in a high-level programming language C/C++, studied in the course of preparation for all engineering degrees at technical universities. The hardware and software co-verification task is also naturally solved because the program tests developed for IP-core verification become the basis for operational software of a given IP-core in a given SoC [10].

The restriction of this methodology usage is the absence of a reference SoC. Such a situation is possible when a principally new SoC with a new processor and a new on-chip bus is developed. But the situation is not typical for the major part of IP-core development and testing processes.

Verification results. Software-driven verification method for a reference SoC and presented verification system architectures are used to verify IP-cores of network infrastructure SpaceWire [11]. The reference SoC as well as the other IP-cores except SpaceWire reference codec are realized on the basis of LEON3 processor core from the open library GR-LIB of Gobham Gaisler [12]. The open codec SpaceWire Light is used as a SpaceWire reference codec [13]. SoC created on the basis of GR-LIB IP-core base can serve as a reference one. Gobham Gaisler Company has a good reputation, its concepts have piloting capacity and are realized both in FPGA and in ASIC.

The architecture presented in fig. 1 is used for SpaceWire fault-tolerant codec verification; the one presented in fig. 2 – for RMAP protocol controller verification [14], and the one presented in fig. 3 – for a routing switch of SpaceWire network.

Verification process is performed in two stages: the first stage – verification on a model and the second one – verification by FPGA prototype. Testing by FPGA implementation allows both detecting errors that were not detected during the model verification, and making sure of IP-core functional specification values for a specific FPGA [15]. In this process the test software which was used for model verification is fully used for FPGA prototype verification which is an important advantage of software-driven verification in a reference SoC. Further the developed test software can be used for target VLSI testing, including a reference SoC and a verified IP-core.

Conclusion. The software-driven IP-core verification method with a reference SoC provides flexibility and variety of possible ways of creating test software; it does not create disparity between verification and synthesis; it provides the SoC hardware and software co-verification; it provides software test portability when tests are created during the verification process of an IP-core model in a computational model on a SoC FPGA-prototype including this IP-core.

Software-driven verification architecture with a reference SoC realizes the classic test loop regardless of a verified network device type. Developed modifications of software-driven verification system with a reference SoC are used for IP-core testing in SpaceWire network: a fault-tolerant codec, a RMAP protocol controller and a routing switch.

References

1. Meyer A. Principles of Functional Verification. *Newnes*, 2003, 217 p.

2. Bergeron J. Writing Testbenches: Functional Verification of HDL Models. *Kluwer Academic Publishers*, 2000. 354 p.

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

4. Mentor Graphics UVM/OVM Documentation (verification methodology cookbook). *Mentor Graphics Corporation*, 2011, 166 p.

5. Universal Verification Methodology (UVM) 1.1 Class Reference. Available at:www.accellera.org/community/uvm (accessed: 13.01.2017).

6. Kamkin A., Kotsynyak A. *Sredstva funktsional'noy verifikatsii mikroprotsessorov* [Trudy Instituta sistemnogo programmirovaniya RAN]. 2014, Vol. 26, iss. 1, P. 149-200 (In Russ.).

7. Processor-DrivenVerification, Cadence. Available at: www.cadence.com/content/cadence-www/global/en_US/home/tools/system-design-and-verification/software-driven-verification.html (accessed: 10.02.2017).

8. Processor-Driven Verification, Mentor Graphics. Available at: www.mentor.com/products/fv/verification-horizons/processor-driven-verification (accessed: 10.02.2017).

9. Gribkov I., Zakharov A., Kol'tsov P. *Razvitie sistemy stokhasticheskogo testirovaniya mikroprotsessorov INTEG* [Programmnye produkty i sistemy]. 2010, No. 2, P. 14-23 (In Russ.).

10. Shakhmatov A. *Sistema na kristalle v kachestve testovogo okruzheniya dlya verifikatsii slozhnofunktsional'nykh blokov*. [V mat. XX Mezhdunar. nauchn. konf. Reshetnevskie chteniya]. Krasnoyarsk, 2016, Part 1, P. 357–358. (In Russ.).

11. *ECSSE-ST-50-12C SpaceWire – Links, nodes, routers and networks* [European Cooperation for Space Standardization (ECSS)]. 2008, 129 p.

12. LEON3 Processor. Available at: www.gaisler.com/index.php/ products/processors/leon3 (accessed: 10.02.2017).

13. J. Rantwijk, SpaceWire Light v20110709. Available at: www.opencores.org/project, spacewire_light (accessed: 10.02.2017).

14. *ECSSE-ST-50-52C SpaceWire – Remote memory access protocol* [European Cooperation for Space Standardization (ECSS)]. 2010, 109 p.

15. Shakhmatov A., Khanov V., Chekmarev S. *A functional verification system of IP-blocks in network protocols* [12TH International Conference On Actual Problems Of Electronic Instrument Engineering Proceedings (APEIE-2014)]. Novosibirsk, NSTU, 2014, Vol. 1, P. 247–251.

Библиографические ссылки

1. Meyer A. Principles of Functional Verification. Newnes, 2003. 217 p.

2. Bergeron J. Writing Testbenches: Functional Verification of HDL Models. Kluwer Academic Publishers, 2000. 354 p.

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

4. Mentor Graphics UVM/OVM Documentation (verification methodology cookbook). 2011. 166 p.

5. Universal Verification Methodology (UVM) 1.1 Class Reference [Электронный ресурс]. URL: http://www.accellera.org/community/uvm (дата обращения: 13.01.2017).

6. Средства функциональной верификации микропроцессоров / А. С. Камкин [и др.] // Труды Института системного программирования РАН. 2014. Т. 26, вып. 1. С. 149–200.

7. Processor-Driven Verification, Cadence [Электронный ресурс]. URL: https://www.cadence.com/content/cadence-www/global/en_US/home/tools/system-design-and-verification/software-driven-verification.html (дата обращения: 10.02.2017).

8. Processor-Driven Verification, Mentor Graphics [Электронный ресурс]. URL: https://www.mentor.com/products/fv/verificationhorizons/processor-driven-verification (дата обращения: 10.02.2017).

9. Развитие системы стохастического тестирования микропроцессоров INTEG / И. В. Грибков [и др.] // Программные продукты и системы. 2010. № 2. С. 14–23.

10. Шахматов А. В. Система на кристалле в качестве тестового окружения для верификации сложнофункциональных блоков // Решетневские чтения : материалы XX Междунар. науч. конф. / СибГАУ. Красноярск, 2016. Ч. 1. С. 357–358.

11. ECSSE-ST-50-12C SpaceWire – Links, nodes, routers and networks. European Cooperation for Space Standardization (ECSS), 2008. 129 p.

12. LEON3 Processor [Электронный ресурс]. URL: http://www.gaisler.com/index.php/products/processors/leon3 (дата обращения: 10.02.2017).

13. Joris van Rantwijk, SpaceWire Light v20110709 [Электронный ресурс]. URL: http://opencores.org/project, spacewire_light, (дата обращения: 10.02.2017).

14. ECSSE-ST-50-52C SpaceWire – Remote memory access protocol. European Cooperation for Space Standardization (ECSS), 2010. 109 p.

15. Shakhmatov A. V., Khanov V. Kh., Chekmarev S. A. A functional verification system of IP-blocks in network protocols // 12th International Conference on Actual Problems of Electronic Instrument Engineering Proceedings (APEIE-2014). Novosibirsk : NSTU, 2014. Vol. 1. P. 247–251.