DOI: 10.15514/ISPRAS-2019-31(3)-5

Approach to test program development for multilevel verification

P.V. Frolov, ORCID: 0000-0002-9810-2210 <Pavel.V.Frolov@mcst.ru>
INEUM, 24, Vavilova st., Moscow, 119334, Russia
MCST, 1, Nagatinskaya st., Moscow, 117105, Russia

Abstract. Development of system-on-chips or network-on-chips requires verification of standalone units (peripherals and commutators) and a system as a whole. An approach to test development for verification of programmable standalone units is presented. The tests are written in C++ using a specific API to program the device-under-test (DUT) and the test environment. The API functions are implemented in the standard environment library; the specific implementation depends on the test environment structure: a standalone device, a device as a part of controllers block or a device as a part of the whole SoC. For system-level verification the test program is translated for execution on a general-purpose core of the verified SoC as well as the standard environment library. The testbench for unit-level verification consists of the environment library and the test linked to the testbench as a PLI-application, an adapter for the DUT-system bus interface and, possibly, a specific imitator of an external device. Different devices with one programming interface can be tested by the same test program even if they have different bus interfaces; different bus interfaces require different adapters to be implemented. The presented approach gives an opportunity to use the same test program both for standalone and for system-level verification (as an integration test). The implementation of the presented approach and its application to verification of microprocessors of the Elbrus family are described.

Keywords: hardware verification; simulation-based verification; test system; standalone verification; system-level verification

For citation: Frolov P.V. Approach to test program development for multilevel verification. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 3, 2019. pp. 59-66. DOI: 10.15514/ISPRAS-2019-31(3)-5

Разработка универсальных тестовых программ для автономной и системной логической верификации программируемых контроллеров

П.В. Фролов, ORCID: 0000-0002-9810-2210 <Pavel.V.Frolov@mcst.ru>
ПАО «ИНЭУМ им. И.С.Брука», 119334, Россия, г. Москва, ул. Вавилова, д. 24
АО «МЦСТ», 117105, Россия, г. Москва, ул. Нагатинская, д. 1, стр.23

Аннотация. При разработке систем-на-кристалле необходимо проводить верификацию как отдельных подмодулей (контроллеров периферийных интерфейсов и коммутаторов), так и системы в целом. В статье представлен подход к разработке тестов для верификации программируемых контроллеров. Тесты разрабатываются на языке программирования C++; программирование тестируемого устройства и тестового окружения осуществляется с помощью специального программного интерфейса. Функции этого программного интерфейса реализуются в стандартной библиотеке тестового окружения; реализация зависит от структуры тестового окружения: в качестве моделируемого устройства может выступать только тестируемый контроллер, контроллер в составе блока контроллеров, или контроллер в составе полной системы-на-кристалле. Для верификации системного уровня библиотека и тестовая программа компилируются для исполнения на одном из вычислительных ядер системы-на-кристалле. При автономной верификации тестовая программа и библиотека окружения формируют программный

модуль, взаимодействующий с симулятором RTL-описания с помощью стандартного интерфейса PLI; библиотечные функции взаимодействуют с моделируемым устройством через специальный адаптер системного интерфейса; кроме того, в тестовое окружение может быть включен имитатор внешнего устройства. При таком устройстве тестового окружения одна и та же тестовая программа может проверять устройства с одним программным интерфейсом, но разными системными интерфейсами; необходимо только реализовать соответствующие адаптеры. Представленный подход позволяет запускать тестовую программу как автономный тест, так и в качестве теста интеграции на верифицируемой системе-на-кристалле. В статье описаны реализация представленного подхода и его применение в маршруте верификации микропроцессоров семейства Эльбрус.

Ключевые слова: логическая верификация аппаратуры, верификация на основе моделирования, тестовая система, автономная верификация, системная верификация

Для цитирования: Фролов П.В. Разработка универсальных тестовых программ для автономной и системной логической верификации программируемых контроллеров. Труды ИСП РАН, том 31, вып. 3, 2019 г., стр. 59-66 (на английском языке). DOI: 10.15514/ISPRAS-2019-31(3)-5

1. Introduction

Typical test scenarios for programmable standalone units (peripherals and commutators) are based on estimated work patterns of the designed chip operating. Such test scenarios are an indispensable part of a standalone verification testplan. They also must be included in a device integration test suite for system-level verification to check considered device interaction with other units.

This paper describes an approach to test development for verification of programmable standalone units which allows using the same test both for standalone and system-level verification. The presented approach also enables tests run in different *execution environments* (via an RTL simulator, an FPGA-based prototype or a manufactured chip).

The rest of paper is organized as follows. Section 2 reviews the existing techniques considering the same tests reuse for different execution environments. Section 3 introduces the structure of the framework for test development, implementing presented approach. Section 4 describes API provided by the framework for tests use. Sections 5 and 6 present test transformation for system-level and standalone verification respectively. In Section 7, results are presented and in Section 8, possible/planned future work is mentioned.

2. Related work

The main target of the presented approach is to reduce verification effort through the unit-level tests reuse for system-level simulation.

Review works on SoC verification suppose high level of the verification components reuse [1][2], but there is not much information about practical approaches for the test programs reuse. The problem of the stimulus reuse for different execution targets and environments is targeted by The Portable Test and Stimulus Standard (PSS) [3], but this standard provides only language for a test intent description [4].

Typical approach to unit-level verification is transaction-based verification, implemented, for example, in UVM (Universal Verification Methodoly) standard [5]. Such tests are written in SystemVerilog and commonly use constraint-random stimuli generation, implemented via external tools (RTL-simulator, for example). The reuse of such a test for system-level verification requires its additional adaptation. For example, the work [6] describes an approach which allows to get a system-level test based on the unit-level one for the separate IP-block (GPU) of the heterogeneous SoC. A trace of DUT interactions with the testbench is logged during unit-level simulation and then is compiled into assembly, ready for execution on the CPU at SoC level. The approach copes with register polling through the test driver library instumentation but DUT interrupts handling isn't described.

3. Test development framework

In the presented approach, a test is written in C++, so it can be translated to different host CPU architectures:

- to a PLI-application [7] (PLI is for Program Language Interface) interacting with a simulator modeling the RTL description of the standalone unit (or the block of controllers including this unit);
- for system-level execution on one of the general-purpose cores of the verified SoC.

The system-level test runs without an operating system and this restricts usage of standard C++/C library: no explicit usage of externally linked functions is allowed. Instead, the test development framework provides a common standard API for different test execution environments. The API is described in header files as a list of C++ function prototypes. For every supported test execution environment the framework provides a corresponding *environment library* implementing these functions.

Advantages of C++ as a test implementation language mainly address system-level test execution.

Firstly, C++ allows to transfer some calculations to the compilation stage via ***contexpr*** specifier (since C++11 [8] version of language standard). Secondly, C++-templates allow wrap of specific assembly instructions into ***inline*** functions to avoid function call overhead while preserving test portability. Besides, parts of device drivers or BIOS, commonly written in C, can be relatively simply ported for test use and vice-versa.

4. Environment library API

A typical programmable controller implements three kinds of interaction with a system: it provides access to the internal registers and memory for configuration (PIO, programmed input/output), can initiate DMA-transactions (Direct Memory Access) to the system memory and send interrupt messages. Thus the environment library API must provide means to perform, control and observe these interactions.

The API contains a description of typical operations:

- access to the registers and the internal memory of the device under test,
- system memory handling operations (allocation, pattern filling, data comparison),
- device interrupts handling,
- address translation for DMA-transactions programming,
- simulated time measuring and timeout setup,
- debug test output,
- other auxiliary procedures.

5. System-level verification

For system-level verification the test program is translated for execution on a general-purpose core of the verified SoC as well as the standard environment library. The framework also provides a bootstrap program for basic system initialization required for the test to run. The test and the library are linked into a single executable image (the system-level test). To run the test the execution environment places this image in the memory of the SoC (DRAM and/or NVRAM) and transfers control to the entry point of the environment library, which in turn calls the test function. After the test execution the environment library handles the exit code and provides diagnostic information (fig. 1).

The framework allows executing the same unit test with different system settings, providing comprehensive unit integration check. System settings programming is performed by the bootstrap part of the environment library; their values are described in additional files and are transmitted to

the system-level test either via compilation macro definitions or as object files with initialized C-structures during linkage.

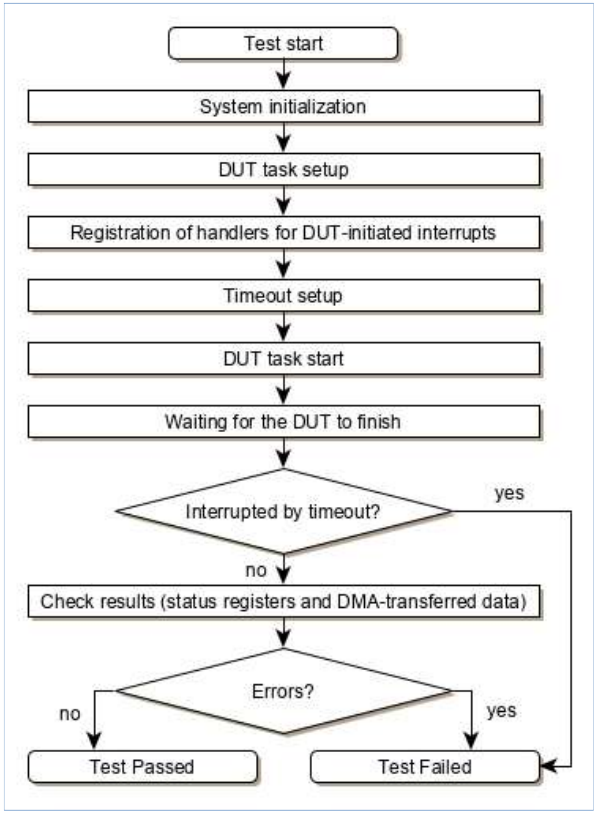

Fig. 1. The framework for system-level verification

Examples of system settings to vary range from separate bits in different control registers of the verified SoC to modes which require additional nontrivial setup. For example, DMA-transactions from the tested device can work directly with system physical addresses or can be additionally redirected via the IOMMU (Input/Output Memory Management Unit).

The environment library implements the API with functions executed in super-user mode. Read/write access to the device registers is implemented with load/store instructions with specific attributes (memory type specifiers). In microprocessors of the Elbrus family registers of external programmable devices are placed within PCI-address spaces: memory, I/O and PCI-configuration space. The test defines a target device address in a PCI-configuration space and allocates necessary address ranges in PCI I/O or memory spaces via appropriate API functions.

The environment library provides a simple heap manager without deallocation implementation. The test program allocates data arrays in the heap for use as RAM regions accessed from the tested device by DMA-transactions.

Virtual addresses for DMA-transactions are written to the device registers and/or to descriptor tables in RAM. In the simplest case the virtual address is equal to the physical address: so-called transparent translation, but DMA-transactions from the tested device can be redirected via the IOMMU, so the environment library provides functions for IOMMU configuration and in-test

address translation functions. The test uses that functions for getting virtual addresses from physical ones, which are returned from the heap allocation-function.

The environment library implements functions for the system interrupt controller configuration and test-defined interrupt handling. The test configures interrupts to be sent by the tested unit and registers callback functions handling those interrupts. During the test execution the environment library catches interrupts from the device and calls registered handlers.

Simulated time measuring is implemented via reading of the clock-counting register or programming local timer to send interrupts in defined time intervals.

The system-level test can be compiled for different execution environments: a functional model, a simulated RTL-description of the tested SoC, an FPGA-based emulator or a manufactured chip. The target execution environment determines the bootstrap procedure and the debug print support linked to the test.

The functional model allows fast execution with high observability (instruction execution trace, units programming trace), so it is used for the test and the environment library debug.

6. Unit-level verification

The structure of the unit-level testbench is presented on fig. 2. The testbench consists of the environment library and the test linked to the testbench as a PLI-application, an adapter for the DUT-system bus interface and, possibly, a specific imitator of an external device.

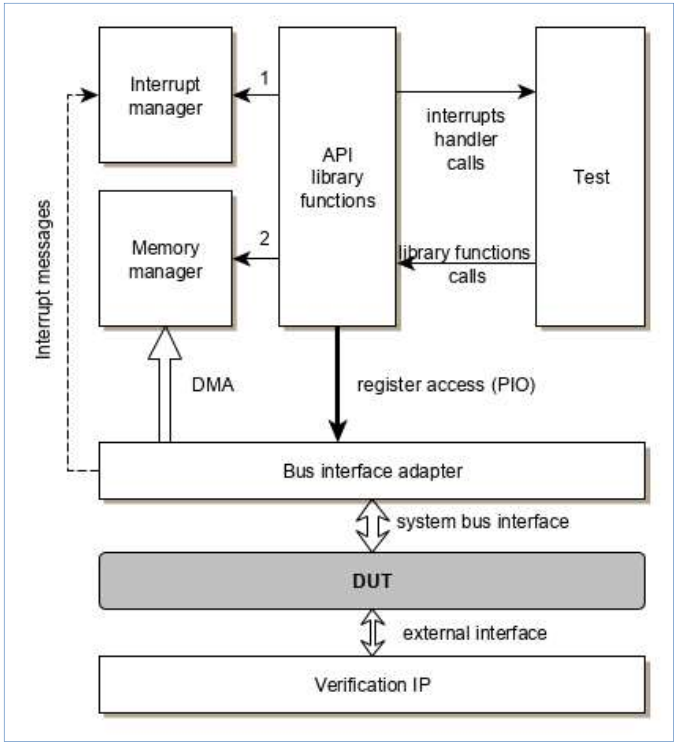

Fig. 2. The structure of the unit-level testbench

The interface of the device-under-test which connects it to the rest of the SoC requires an appropriate adapter for interaction with the testbench. It provides an interface-specific implementation for DUT registers access operations and redirects DUT-initiated transactions to the environment library.

There are two separate address spaces in the test: "internal" for direct access from the test and "external" for DMA-transactions. Memory manager returns to the test pointers with "internal" addresses for memory allocation requests and all library functions for on-core memory processing work with "internal" addresses. Addresses to be targeted by DMA-transactions are wrapped by translation functions that convert internal pointers to external ones and record this translation. DMA-requests are transferred by the adapter to the memory manager that checks DMA destination addresses against previously recorded translations. If there is an appropriate record of translation, the memory manager writes data from DMA-transactions or reads it for return to the adapter. Otherwise an error is detected.

Interrupt messages issued by the device are registered within the environment library; when the test calls library functions, pending interrupts are handled and a user-defined callback is executed.

Simulated time measuring is implemented by means of functions DPI-exported from the part of the library written in SystemVerilog.

Different devices with one programming interface can be tested by the same test program even if they have different bus interfaces; different bus interfaces require different adapters to be implemented. The tested controller can be connected to the adapter not directly, but through the root commutator of the block of controllers including the unit in consideration (fig. 3).

That variant of the DUT allows verification of interaction between system commutator and the tested controller (intermediate-level verification). Test scenarios with simultaneous work of several controllers can be implemented.

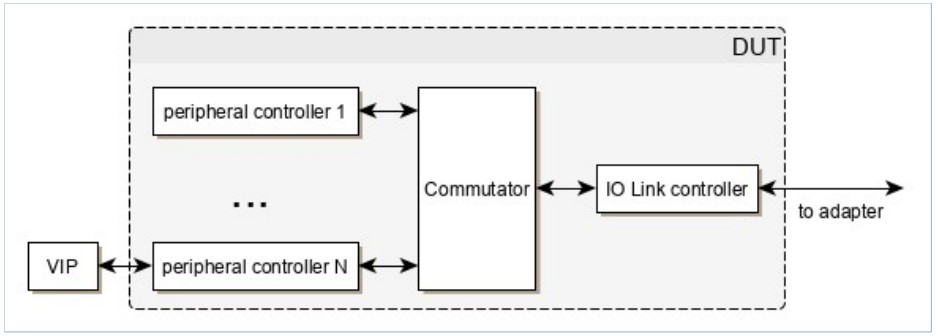

Fig. 2. Indirect connection through the root commutator

7. Results and use experience

The described approach to test development has been applied to verification of peripheral interfaces controllers of standalone southbridge ASICs developed in MCST [9], such as HD Audio, SATA, USB 2.0, PCI and PCI-e bridges, and multiple low-speed controllers. Now it is used for verification of embedded IOHubs being developed for a new generation of the Elbrus microprocessors. Standalone and embedded southbridges have different in-house interfaces to transfer packets based on PCI Express transaction layer packets [10], therefore different adapters have been implemented in order to reuse the same set of tests.

MCST designs computing systems based on CPU of Elbrus and SPARC instruction set architectures, thus the environment library for system-level tests is implemented for both architectures and for different microprocessor models (starting from Elbrus-4C [13] for Elbrus-based microprocessors and R-1000 [14] for SPARC-based ones).

The typical test development and use flow consists of the following subsequent stages:

- system-level build for functional model execution and test logic debug;
- unit-level build for standalone unit verification;
- unit-level build for verification of the unit as a part of the southbridge;
- system-level build for test execution on full system-on-chip (RTL or FPGA-based prototype [11]).

The system-level environment library supports simultaneous execution of several tests for different controllers on multi-core systems. Tests are executed on different cores; shared resources are distributed between tests based on static planning [12].

8. Future work

The described approach for unit-level verification was implemented mainly for southbridge controllers of MCST projects. The future work is supposed to embrace adaptation of system-level tests for north-bridge integrated graphics cores to unit-level verification. It requires further development of internal system bus interface adapters for different target CPU models.

There is also an endeavor to use already developed system-level tests for verification of hardware I/O virtualization support in new microprocessors of Elbrus family. The test program is supposed to run as a simple guest OS while the environment library functions are executed in hypervisor mode. Different modes of virtual I/O support are to be implemented: emulation mode and direct device assignment.

References / Список литературы

[1]. Anil Deshpande. Verification of IP-Core Based SoC's. In Proc. of the 9th International Symposium on Quality Electronic Design, 2008, pp.433–436.

[2]. G. Mosensoson. Practical approaches to SoC verification. In Proc. of the DATE User Forum, 2002.

[3]. The Portable Test and Stimulus Standard. Available at: https://www.accellera.org/downloads/standards/portable-stimulus, accessed: 11-May-2019.

[4]. Bryon Moyer. Portable Stimulus Intent. Accellera's New Standard Goes to Early Adopters. EEJournal, July 31, 2017. Available at: https://www.eejournal.com/article/portable-stimulus-intent, accessed: 11-May-2019.

[5]. Standard Universal Verification Methodology. Available at: http://accellera.org/downloads/standards/uvm, accessed: 11-May-2019.

[6]. Narendra Kamat. IP Testing for Heterogeneous SOCs. In Proc. of the 14th International Workshop on Microprocessor Test and Verification, 2013, pp. 58–61.

[7]. IEEE Standard for SystemVerilog. IEEE Std 1800-2009.

[8]. ISO International Standard ISO/IEC 14882:2011(E) – Programming Language C++.

[9]. A.K. Kim, M.S. Mikhailov, V.M. Fel'dman. IO-subsystem for «MCST-4R» and «ELBRUS-S» SOCs based on peripheral interfaces controller IC. Issues of Radio Electronics, series EVT, no. 3, 2012 (in Russian) / А.К.Ким, М.С.Михайлов, В.М.Фельдман. Подсистема ввода-вывода для систем на кристалле «МЦСТ-4R» и «Эльбрус-S» на основе микросхемы контроллера периферийных интерфейсов. Вопросы Радиоэлектроники, серия ЭВТ, вып. 3, 2012.

[10]. Petrochenkov M. V., Mushtakov R. E., Stotland I. A. Verification of 10 Gigabit Ethernet controllers. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 257-268. DOI: 10.15514/ISPRAS-2017-29(4)-17.

[11]. F. Budylin, I. Polishyk, M. Slesarev, S. Yurlin. The experience of prototyping MCST CJSC' microprocessors. Issues of Radio Elactronics, series EVT, 2012, no. 3 (in Russian) / Ф.К. Будылин, И.А. Полищук, М.В. Слесарев, С.В. Юрлин. Опыт прототипирования микропроцессоров компании ЗАО «МЦСТ». Вопросы радиоэлектроники, серия ЭВТ, 2012, вып. 3.

[12]. Frolov P.V. System-level test integration based on static resource allocation. Issues of Radio Electronics, series EVT, 2018, no. 2, pp. 76–80 (in Russian) / П.В. Фролов. Система интеграции инженерных тестов на основе статического распределения ресурсов. Вопросы радиоэлектроники, 2018, вып. 2, стр. 76–80.

[13]. Central processor unit «Elbrus-4C». [Online]. Available at: http://www.mcst.ru/elbrus-4c, accessed: 11-May-2019 / Центральный процессор «Эльбрус-4С».

[14]. Central processor unit «R1000». [Online]. Available at: http://www.mcst.ru/r1000, accessed: 11-May-2019 / Центральный процессор «R1000».

Информация об авторе / Information about author

Павел Викторович ФРОЛОВ – окончил Московский физико-технический институт в 2010 году. Начальник сектора системной верификации отдела моделирования и верификации АО «МЦСТ». Сфера научных интересов: логическая верификация аппаратуры, системы-на-кристалле, автоматизация верификации.

Pavel Viktorovich FROLOV graduated from Moscow Institute of Physics and Technology in 2010. Currently he is a head of the sector of system-level verification in the Department of Modeling and Verification in JSC MCST. Research interests: hardware verification, systems-on-chip, verification automation.