Simulation-based Verification of System-on-Chip Bus Controllers

[1] *M.M. Chupilko <chupilko@ispras.ru>*
[2] *E.A. Drozdova <drozd_96@mail.ru>*
[1] *Ivannikov Institute for System Programming of RAS,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. The paper presents an approach to verification of commutation components of Systems-on-Chip. The core idea is to verify bus controllers and supporting interface parts connected to a reference model at unit-level. The reference model in the approach is suggested to be written in SystemC so that to be easily adjusted to the required bus parameters. The in-house prototype implementing the approach has been applied to the verification of a Verilog model of Wishbone controller. There is a possibility to extend the approach to support other busses and protocols by development of the interface library.

Keywords: unit-level verification; C++TESK

DOI: 10.15514/ISPRAS-2018-30(4)-8

For citation: Chupilko M.M., Drozdova E.V. Simulation-based Verification of Hardware Bus Controllers. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 4, 2018. pp. 129-138. DOI: 10.15514/ISPRAS-2018-30(4)-8

1. Introduction

This paper is devoted to a problem of a technology of unit-level verification of commutation parts of hardware description level (HDL) models. Each System-on-Chip (SoC) in fact is an HDL model, where IP blocks (intellectual property, proprietary functional units), being parts of the system, are connected according to some traditional communication protocol (Wishbone [1], OCP-IP [2], or something else). To verify it, one has to obtain a golden model to be referred to in the verification process, either to create such a model. In case of IP blocks, their reference models are usually provided by their vendors. In case of the commutation part connecting IP blocks, the situation is more difficult. There might be a standard bus controller with a predefined bus width, or, that is more common, there will be an implementation of the standard protocol. The integration problem also looks quite important, as physical layer of the bus protocol is not the only thing that can be erroneous, but the incompatible logic of data transfers between different IP blocks also might be a weak point.

In this research, we propose a technology of a unit-level verification of communication models, using a SystemC reference model provided by a vendor or created manually. C++TESK [3], C++ library of macros meeting all typical requirements of unit-level verification, including reference modeling, stimulus generation, and coverage estimation, is selected as a basic tool for the technology implementation. This tool supports reference model development at least in two ways: in terms of its macro library and easy attachment to any C++-code.

The rest of this paper is divided into five sections. Section 2 contains more information about communication protocols, including information about Wishbone. Section 3 describes related works in the field of verification. Section 4 presents a proposed approach to unit-level verification. Section 5 studies an example of the approach application. Section 6 discusses the results of the work and outlines directions of future research and development.

2. Communication protocols and Wishbone standard

As such, each communication protocol is a system of rules allowing several entities to transmit data to each other. More specific definition includes descriptions of possible entities and fixes a physical parameters of transferring. Typically, when speaking about communication part of SoC, one means transmission layer between active (e.g., processor blocks) and passive (e.g., RAM) blocks inside of SoC. To implement the standard interface for data transmission between IP-blocks, a bus controller is used. The aim of the standard interface is to decrease the number of integration problems and support possible re-use.

There are different ways to connect IP-blocks together. The simplest one is a point-to-point connection. As an example of point-to-point connection, let us consider OCP-IP [2] (Open Core Protocol International Partnership, see fig. 1), which is a medium layer between blocks and the bus. OCP is oriented to typical master (active component) – slave (passive component) communication. The protocol was proposed several years ago as a first step in development of a single standard and flexible solution in communication.

Another standard of communication in SoC is Wishbone [1]. Being widely distributed, it was selected in this work for being an example for test system development. The Wishbone standard specifies a standard interconnection between computational IP cores. It supports interconnection of few IP cores using such methods as a point-to-point, a shared bus (see fig. 2), a crossbar switch (see fig. 3), and a switched fabric. The first one represents a simple interconnection between two IP cores where the one called Master initiates the data exchange and another one called Slave responds to this call. The second method supports binding of more than two blocks in a consequent order. This method is efficient when the data should be transferred from one IP core to another repeatedly. The third and the fourth methods are similar, also representing the interconnection between several IP cores. In both of them, there is a common bus, connecting more than two Master and Slave cores. The Master core can evoke any Slave connected to this bus but the only one at the cycle.

Using the crossbar switch method, several Master-Slave cores can be interconnected simultaneously; using the switch fabric — only one pair of cores is connected.

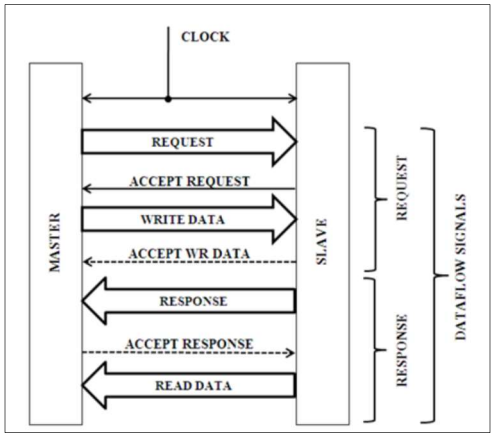

Fig. 1. Open core protocol architecture

From this review, one can derive the following ideas. First, due to the point-to-point connection being the basics of communication in all cases, when testing a bus controller, IP core interfaces should be also taken into consideration. Second, the bus controller is limited in its types of requests (mainly, send and receive); the most important for testing situations seem to be with handling of protocol violations and prevent collisions in bus access.

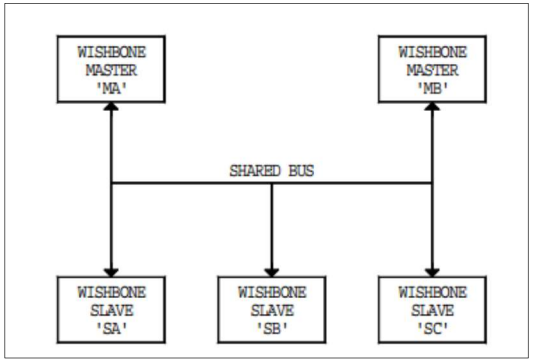

Fig. 2. Wishbone Shared Bus Interconnection

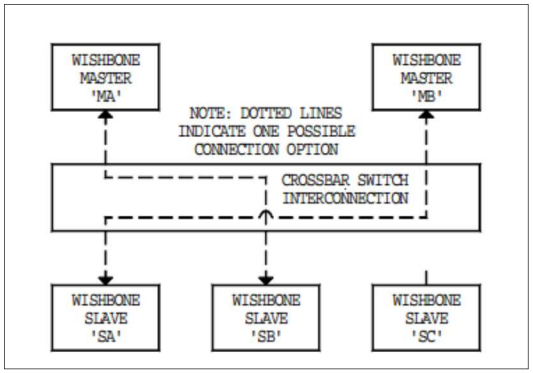

Fig. 3. Wishbone Crossbar Switch Interconnection

3. Related work

The task of SoC verification is typically considered as a problem of a mainly unit-level verification. In this case, a design under test (DUT) is taken separately from its environment. Stimuli are applied and reactions to check are received via DUT wires.

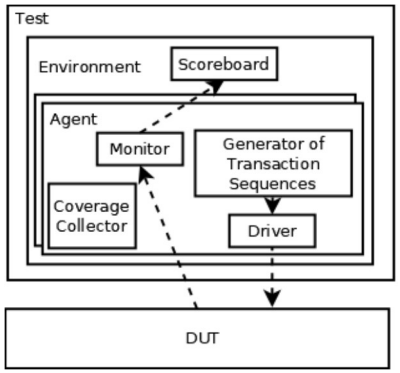

Fig. 4. UVM Test Architecture

Among many unit-level verification approaches, UVM (Universal Verification Methodology) created by Accelera is the most popular. It represents the union of Open Verification Methodology (OVM), Advanced Verification Methodology (AVM) and Universal Reuse Methodology (URM). UVM is a library built upon the SystemVerilog language that provides some basic classes such as the class constructing the testbench structure, the class serving as a basic data structure, the class defining transactions to be passed through components of UVM. This methodology can be used for a constrained random, a coverage-driven, an assertion-based, and emulation-based verification. The UVM testbench structure is as follows (see fig. 4 schematically depicturing UVM test). To begin with, there is the DUT. The transaction sequencer block serves to interact with the DUT by generating sequences

of bits to be transmitted to the DUT. The monitor block is responsible for listening the communication of the DUT and the sequencer, and gets responses from the DUT. The block called scoreboard compares and evaluates all the information that the monitor is receiving from the DUT and the prediction made by the monitor, describing which output is expected to be taken from the DUT. Sequencers, monitors, and coverage collectors together are called agents. An agent and a scoreboard form the environment. At the same time, there is no any explicit method for a making a golden-model as itself, it is up to engineers. The SystemVerilog language is also more a congregation of different methods to describe properties to be checked, rather than a general-purpose programming language convenient for the golden-model development.

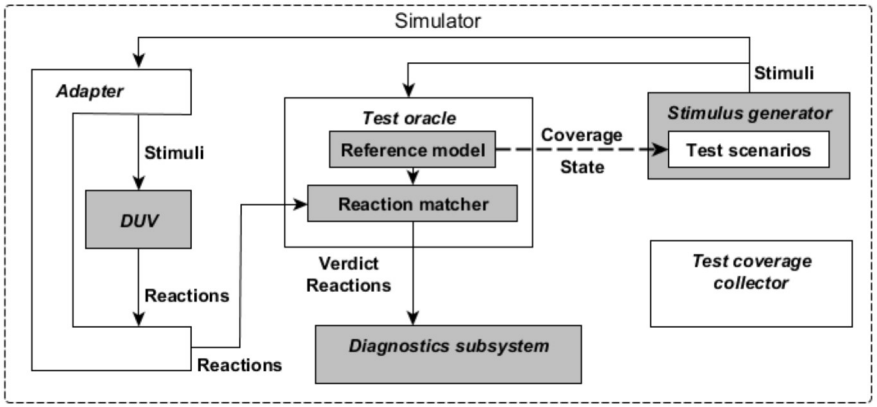

Fig. 5. Verification Environment Architecture

Another approach to unit-level verification is C++TESK Testing Toolkit [3-4], which is represented by a library of C++ macros. The methodology can be used for constrained random and coverage-driven verification. The C++TESK testbench structure (see fig. 5) is similar to the UVM testbench structure but there are some distinctions. According to a selected strategy, C++TESK's stimulus generator chooses one of the predefined stimuli to be sent to the DUT and to the reference model expressed explicitly. The comparator gets the output from the model and from the DUT, compares it and evaluates the coverage. C++TESK is compatible with SystemC, aimed to reference models development, which is one of the key advantages comparatively with UVM.

4. Proposed approach

As it has been already mentioned, either each bus controller is represented by a SystemC model as well as by an RTL description or there is only RTL code and no reference model at all. The conformance between the abstract (SystemC) reference

model and the RTL description should be established. Following a simulation-based tradition, it is to be done by verifying both sides against the same specification or by verifying the RTL description against the (SystemC) reference model.

To perform verification, a C++TESK Testing Toolkit has been selected, being a C++ library with all necessary classes and macros for specifying design behavior, generating stimuli, and checking reactions. A C++TESK-based verification environment is structured as shown in fig. 5.

The central component of the verification environment is a test oracle, which is responsible for checking whether the DUT behaves properly. It typically includes a reference model that takes stimuli as an input and produces reference reactions as an output, and a reaction matcher that intercepts reference reactions and implementations reactions provided by the DUT and composes reactions pairs. Usually, the reaction matcher works independently for each output interface.

To transform high-level stimuli to the low-level ones and low-level reactions to the high-level ones, the adapter is used. It consists of multiple interface adapters, each being connected with a single input or output interface. An interface adapter describes a simple protocol of putting a single stimulus or getting a single reaction. A special component of the reaction matcher, called a reaction arbiter, specifies reactions ordering. The task of the reaction arbiter is to choose a reference reaction (if there are any) for a given implementation one. There are two main predefined strategies: (1) model-based arbitration and (2) adaptive arbitration. The first strategy implies that the reference model is an accurate enough to predict reaction order for some output interface (the arbiter selects the next reference reaction stored in the interface buffer). The second strategy is used when the reference model is time inaccurate, in which case, given an implementation reaction, the arbiter searches for a reference reaction being equal or similar to the given one. If some reactions are mismatched, a diagnostics subsystem explains what is wrong with the DUT in terms of incorrect, missing, and unexpected reactions.

Other components of a verification environment are a stimulus generator and a test coverage collector. The stimulus generator creates stimuli by exploring the abstract state space of the reference model. The generator is supplied with a set of available stimuli and a function for abstract state calculation; it tries to apply each stimulus in each reachable abstract state. Speaking about verification of a bus controller, it is natural to consider e.g. the number of messages in the bus channel controllers as being the abstract state (though any other abstraction is possible). Such an adjustable stimulus generator allows to produce hard-to-get-into situations which include those with missing messages in one long packet transmission. The test coverage collector estimates the verification completeness basing on user-defined functional coverage metrics, which is more efficient than simply code coverage.

The typical way of C++TESK usage for unit-level verification in case of its own reference model is described in earlier papers (e.g. [3] or [4]), and the method of connection to SystemC reference model should be developed. To be more precise, SystemC model should include not only the model of DUT itself, but also the model

of its environment, including full communication topology. It allows sending complex requests, model collisions, and so on.

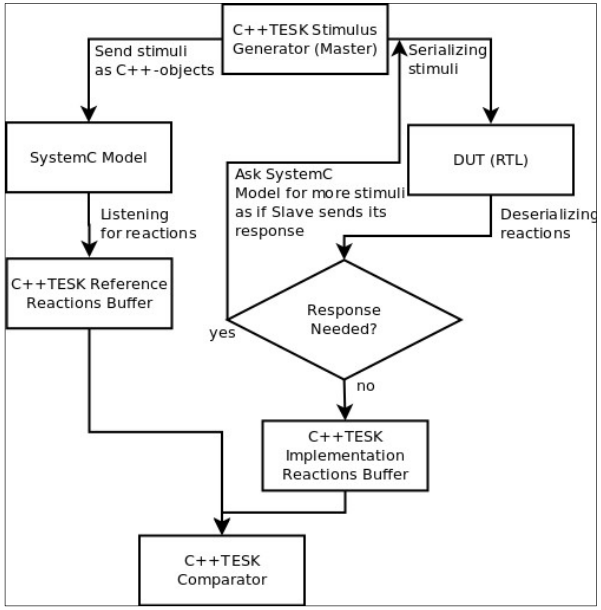

Fig. 6. Architecture of C++TESK, DUT, and SystemC Models Interconnection

The following scheme of commutation between C++TESK and SystemC model is proposed (see Figure 6). C++TESK stimulus generator substitute a master component for the controller bus. Stimuli are applied to both SystemC model (via function calls; to its selected master as if this master wants to send stimuli which the generator applied instead of it) and to DUT (via procedural interface between C and HDL-simulator; to the input master interface of DUT). SystemC computes the behavior of all environment and creates reactions those are to be got from DUT and checked, and those, which are in fact additional stimuli to DUT, e.g. responses from slaves to DUT which are requested by the bus controller and which are substituted by test environment. Output DUT reactions are checked against correspondent SystemC reactions by C++TESK reaction comparator. To provide the program interface between C++TESK and SystemC model, there is a top SystemC class encapsulating all the interfaces between masters and slaves, and the model of bus controller. This class is referenced to in the C++TESK golden model. In order to register reactions from SystemC in proper C++TESK adapters (serializers for stimuli and deserializers for reactions to be checked), there are special listeners of SystemC model activities (more precisely, bool vector of new reactions on different interfaces). When there is some new reaction, it is registered either to be applied to DUT as a stimulus, or to be

added into list of reference reactions to find correspondent implementation reaction in given time.

5. Case study

To develop a prototype of the verification system and to make experiments with it, a Verilog model of simple Wishbone controller has been taken. The controller supports only point-to-point connection. A correspondent SystemC model has been developed from scratch, taking into account the necessity to support different bus sizes and different topologies of master-slave interconnections. The resulted class (see the following listing to see the main part of it) has been attached to C++TESK reference model as an object that should be stimulated by the stimulus generator, and with a possibility of sending reactions that are to be checked against the Verilog model. The whole aspects of the earlier proposed architecture including stimulus generation, HDL implementation reaction checking, and coverage estimation have been kept alive in the prototype of test system. Experiments show that the proposed ideas really work and that future research into this field is required.

```
template <typename adr_bus_size, typename data_bus_size>
  SC_MODULE(Controller) {
    typedef Master<adr_bus_size, data_bus_size> master_type;
    typedef Slave<adr_bus_size, data_bus_size> slave_type;
    // types for containers with masters and slaves
    typedef std::map<master_type*> masters_type;
    typedef std::map<slave_type*> slaves_type;

  ...

  SC_HAS_PROCESS(Controller);
    Controller(sc_module_name _name, bus_mode mode,
               masters_type &masters, slaves_type &slaves):
                 sc_module(name), mode(mode);
    // to register all system's masters and slaves
    // and to bind all masters and slaves to the controller
    void register_master(master_type &master);
    void register_slave(slave_type &slave);
    // to listen for request messages
    void request_listener();
}
```

6. Conclusion

The approach of verification of communication parts of SoC by means of adjustable SystemC reference models and by means of C++TESK's stimulus generator, reaction matcher, and coverage collector has been proposed. Description of the approach includes the architecture of test systems. The idea has been checked in form of a test system prototype for a Verilog model of Wishbone controller. This research should result in the creation of a SystemC library of adjustable models of widely distributed bus standards.

References

[1]. Specification for the WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores. Revision: B.3. Available at: https://cdn.opencores.org/downloads/wbspec_b3.pdf, accessed 20.07.2018
[2]. Open Core Protocol Specification 3.0. Available at: http://www.accellera.org/images/downloads/standards/ocp/OCP_3.0_Specification.zip, accessed 20.07.2018
[3]. M. Chupilko, A. Kamkin. A TLM-Based Approach to Functional Verification of Hardware Components at Different Abstraction Levels. In Proceedings of the Latin American Test Workshop (LATW), 2011, 1-6 pp. DOI: 10.1109/LATW.2011.5985902
[4]. M. Chupilko, A. Kamkin. Runtime Verification Based on Executable Models: On-the-Fly Matching of Timed Traces. In Proceedings of the Model-Based Testing Workshop (MBT), 2013, pp. 67-81. DOI: 10.4204/EPTCS.111.6

Динамическая верификация контроллеров шин систем-на-кристалле

[1] *М.М. Чупилко <chupilko@ispras.ru>*
[2] *Е.А. Дроздова <drozd_96@mail.ru>*
[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. Александра Солженицына, д. 25*
[2] *Московский государственный университет им. М.В. Ломоносова, 119991, Россия, г. Москва, Ленинские горы, д. 1*

Аннотация. В работе представлен подход к верификации коммутационных компонентов систем на кристалле. Основной идеей подхода является верификация контроллеров и поддерживающих интерфейсный обмен частей устройств на модульном уровне с помощью моделей, написанных на SystemC. Эталонные модели в предлагаемой тестовой системе должны быть легко настраиваемыми под требуемые параметры шины. Прототип реализации подхода был применен для верификации Verilog-модели контроллера шины Wishbone. В подходе заложена возможность расширения поддержкой других шин и протоколов посредством разработки библиотеки интерфейсов.

Ключевые слова: модульная верификация, C++TESK

DOI: 10.15514/ISPRAS-2018-30(4)-8

Для цитирования: Чупилко М.М., Дроздова Е.А. Динамическая верификация контроллеров шин систем-на-кристалле. Труды ИСП РАН, том 30, вып. 4, 2018 г., стр. 129-138 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(4)-8

Список литературы

[1]. Specification for the WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores. Revision: B.3. Доступно по ссылке:https://cdn.opencores.org/downloads/wbspec_b3.pdf, дата обращения: 20.07.2018

[2]. Open Core Protocol Specification 3.0. Доступно по ссылке:: http://www.accellera.org/images/downloads/standards/ocp/OCP_3.0_Specification.zip, дата обращения: 20.07.2018
[3]. M. Chupilko, A. Kamkin. A TLM-Based Approach to Functional Verification of Hardware Components at Different Abstraction Levels. In Proceedings of the Latin American Test Workshop (LATW), 2011, 1-6 pp. DOI: 10.1109/LATW.2011.5985902
[4]. M. Chupilko, A. Kamkin. Runtime Verification Based on Executable Models: On-the-Fly Matching of Timed Traces. In Proceedings of the Model-Based Testing Workshop (MBT), 2013, pp. 67-81. DOI: 10.4204/EPTCS.111.6