Test Generation for Digital Hardware Based on High-Level Models

[1] *M.M. Chupilko <chupilko@ispras.ru>*
[1,2,3] *A.S. Kamkin <kamkin@ispras.ru>*
[1] *M.S. Lebedev <lebedev@ispras.ru>*
[1] *S.A. Smolov <smolov@ispras.ru>*
[1] *Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
[2] *Lomonosov Moscow State University (MSU),*
GSP-1, Leninskie Gory, Moscow, 119991, Russia.
[3] *Moscow Institute of Physics and Technology (MIPT),*
9, Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russia

Abstract. Hardware testing is a process aimed at detecting manufacturing faults in integrated circuits. To measure test quality, two main metrics are in use: fault detection abilities (fault coverage) and test application time (test length). Many algorithms have been suggested for test generation; however, no scalable solution exists. In this paper, we analyze applicability of functional tests generated from high-level models for low-level manufacturing testing. A particular test generation method is considered. The input information is an HDL description. The key steps of the method are system model construction and coverage model construction. Both models are automatically extracted from the given description. The system model is a representation of the design in the form of high-level decision diagrams. The coverage model is a set of LTL formulae defining reachability conditions for the transitions of the extended finite state machine. The models are translated into the input format of a model checker. For each coverage model formula the model checker generates a counterexample, i.e. an execution that violates the formula (makes the corresponding transition to fire). The approach is intended for covering of all possible execution paths of the input HDL description and detecting dead code. Experimental comparison with the existing analogues has shown that it produces shorter tests, but they achieve lower stuck-at fault coverage comparing with the dedicated approach. An improvement has been proposed to overcome the issue.

Keywords: digital hardware; hardware description language; manufacturing testing; stuck-at fault; high-level decision diagram; extended finite state machine; model checking; fault propagation.

DOI: 10.15514/ISPRAS-2017-29(4)-16

For citation: Chupilko M.M., Kamkin A.S., Lebedev M.S., Smolov S.A. Test Generation for Digital Hardware Based on High-Level Models. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 247-256. DOI: 10.15514/ISPRAS-2017-29(4)-16

1. Introduction

Functional verification and test generation are resource-consuming activities of the hardware design process [1]. To automate these activities, *models* are frequently used. Models are mathematical abstractions that describe system structure and behavior. There is a variety of verification and test generation problems that can be solved with the help of models: checking system behavior in simulation-based verification [2], directed test generation [3], etc.

The essential stage of the hardware design process is *register-transfer-level* (*RTL*) design. This stage results in code in a *hardware description language* (*HDL*), such as VHDL and Verilog [4]. The RTL model is automatically synthesized into a *gate-level netlist* represented in an HDL or a special language, such as BLIF [5]. Finally, the place-and-route stage is applied to produce a chip layout.

Functional verification, including functional test generation, deals with RTL models, while generation of manufacturing tests uses gate-level netlists. In this paper, we analyze applicability of functional tests for manufacturing testing. The motivation is clear: the simpler the model, the easier to get tests. We extract high-level models from HDL descriptions and generate tests from them. The approach allows reaching good code coverage with short tests [6].

This paper continues research initiated in [7], where we compared fault detection abilities of different test generation methods. A test is said to *detect* a fault, if the *mutant*, i.e. the design with the injected fault, and the original design return different outputs for the test's input sequence. Fault detection ability is measured as the amount of faults having been detected.

The rest of the paper is organized as follows. Section 2 defines formalisms used in the work and gives a brief overview of a fault model. Section 3 summarizes works on applying model-based techniques to manufacturing testing. Section 4 describes the proposed approach. Section 5 reports experimental results. Section 6 suggests a possible approach improvement. Section 7 discusses the results of the work and concludes the paper.

2. Preliminaries

Let V be a finite set of *variables*. A *valuation* is a function that associates each variable with a value from the corresponding domain. Let D_V be the set of all possible valuations of V.

A *guard* is a Boolean function defined on valuations: $D_V \to \{0, 1\}$. An *action* is a transformation of valuations: $D_V \to D_V$. A pair $\gamma \to \delta$, where γ is a guard and δ is an action, is called a *guarded action*. It is implied that there is a description of every

function in an HDL-like language (thus, we can reason not only about semantics, but about syntax as well).

An *extended finite state machine* (EFSM) is a tuple $M = \langle S_M, V_M, T_M \rangle$, where S_M is a set of *states*, $V_M = (I_M \cup O_M \cup R_M)$ is a set of variables, consisting of *inputs* (I_M), *outputs* (O_M) and *registers* (R_M), and T_M is a set of *transitions*. A transition $t \in T_M$ is a tuple $(s_t, \gamma_t \to \delta_t, s_t')$, where s_t and s_t' are respectively the *initial* and the *final* state of t, whereas γ_t and δ_t are respectively the guard and the action of t.

A pair $(s, v) \in S_M \times D_{V_M}$ is referred to as a *configuration*. A transition t is said to be *enabled* in a configuration (s, v) if $s_t = s$ and $\gamma_t(v) = 1$.

An EFSM operates in discrete time. In the beginning, it resets the configuration: $(s, v) \coloneqq (s_0, v_0)$, where (s_0, v_0) is a predefined configuration. On every tick, it computes the set of enabled transitions:

$$T_E \coloneqq \{t \in T_M | (s_t = s) \wedge (\gamma_t(v) = 1)\}.$$

A single transition $t \in T_E$ (chosen nondeterministically) fires: $(s, v) \coloneqq \left(s_t', \delta_t(v)\right)$.

A *netlist* is a tuple $N = \langle V_N, G_N, L_N \rangle$, where V_N is a set of variables, G_N is a set of *gates*, and L_N is a set of *latches*. A gate $g \in G_N$ is a tuple $\left(I_g, o_g, f_g\right)$, where $I_g \subseteq V_N$ and $o_g \in V_N$ are respectively the *inputs* and the *output* of g, and $f_g: Dom_{I_g} \to \{0, 1\}$ is the function of g. A latch $l \in L_N$ is a tuple (i_l, o_l), where $i_l \in V_N$ and $o_l \in V_N$ are respectively the *input* and the *output* of l.

A netlist operates as follows. In the beginning, it initializes the latches' outputs with some predefined values. On every tick, it computes the gates' output values based on the input values and transmits the latches' input values to the outputs.

To compare test generation methods, the well-known *stuck-at fault* model is used. We consider the following variation of the model. There is a stuck-at fault if some gate is "corrupted" so as its function, which is not identically equal to a constant, always returns a constant, either 0 (stuck-at-0) or 1 (stuck-at-1).

3. Related work

This section overviews the existing model-based test generation methods aimed at covering stuck-at faults.

In [8], an approach to functional test generation for VHDL designs is proposed. The method consists of the following stages:

1) translation of an HDL description into *binary decision diagrams* (*BDD*);
2) insertion of a stuck-at fault into the BDD;
3) generation of a *distinguishing test* for the original BDD and the faulty one.

For the HDL-to-BDD translation, the approach uses a method described in [9].

In [10], a combined approach is suggested. It uses two kinds of models: a *high-level decision diagram* (*HLDD*) and an EFSM. Both models are automatically extracted from an HDL description. HLDD is a generalization of BDD: non-terminal nodes of

a diagram are marked not only by 0 and 1 but by arbitrary expressions. First, a test is generated that covers all branches of the diagram. Then, the test is passed to the EFSM simulator to measure the transition coverage. To cover the uncovered EFSM transitions, a special *backjumping* technique is applied.

In [6], another EFSM-based approach is proposed. It fixes several issues of the previously mentioned method and uses a different EFSM extraction technique. The experiments have shown that new tests are shorter, while code coverage is the same.

In [7], the method [6] is experimentally compared with another one, which uses the ABC equivalence checker [11] to generate a distinguishing sequence for two BLIF descriptions. The EFSM-based method demonstrates higher HDL code coverage and shorter tests, while the ABC-based one achieves higher stuck-at fault coverage.

4. Proposed approach

In this paper, we continue our work on applying the model-checking techniques for test generation [12]. The approach allows achieving high HDL code coverage with very short tests. Our current goal is to evaluate how good the approach is in terms of the stuck-at-fault coverage. The method flow is shown in Fig. 1.

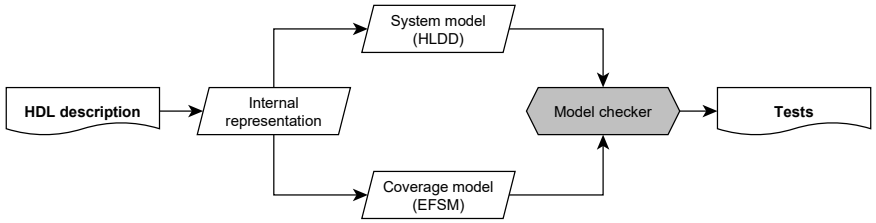

Fig. 1. Model checking-based approach to test generation for HDL descriptions

The method uses two models extracted from an HDL description: a *system model*, which is based on the HLDD formalism, and a *coverage model*, which utilizes the EFSM concept (see [12] and [13] for more details). The system model represent the system functionality, while the coverage model defines a set of conditions, so-called *coverage items*, to be covered by tests.

Let us say a few words about the coverage model. For each HDL process, a separate EFSM is extracted. The EFSM states are mutually disjoint constraints on *state-like registers* (*SLR*). The SLR are chosen automatically with the help of dataflow-based heuristics. The EFSM transitions are constructed from the process execution paths. Coverage items are *reachability conditions* for the EFSM transitions. Let s be an EFSM's state, $c(s)$ be the corresponding constraint, and t be an outgoing transition, i.e. $s_t = s$. In terms of the *linear temporal logic* (*LTL*), the reachability condition is as follows: $\mathbf{F}\{c(s) \wedge \gamma_t\}$, where $\mathbf{F}\{x\}$ means that x will eventually be true.

In accordance with [12], the system model and the coverage items in the negated form ($\neg\mathbf{F}\{c(s) \wedge \gamma_t\}$), are translated into the input format of a model checker. For each

item, the model checker constructs a *counterexample*, i.e. an execution that violates the corresponding formula. Since coverage is formulated in the negated form, the counterexamples are executions that reach the related EFSM transitions. Counterexamples are translated into testbenches and executed by an HDL simulator.

The method is aimed at covering EFSM transitions. However, being rather flexible, it can be applied to various coverage models.

5. Experiments

The proposed approach has been implemented in the Retrascope 0.2.1 tool [14]. The implementation uses the Fortress library [15] and the nuXmv model checker [16].

The method has been tested on some designs from the ITC'99 benchmark [17]. Three test generation methods were compared:

1) the method described in this paper (nuXmv);
2) the method based on EFSM traversal (RETGA) [6];
3) the method based on equivalence checking (ABC) [7].

The third method uses the ABC tool [11] to generate distinguishing sequences for design represented in the BLIF format.

Two metrics were used for test comparison: the length in ticks and the number of killed mutants (detected faults).

To generate mutants, a DTESK prototype was used. Given a fault model and an HDL description, the tool generates a number of mutants along with testbenches. Each testbench contains the original design and the mutant; it reads input values from the file, passes them to both designs, and compares the outputs' values; if there is a mismatch at some tick, then the mutant is considered to be *killed*.

Table 1 shows information about the ITC'99 designs: the source code size (in lines of code), the system model size, and the number of stuck-at fault mutants.

Table 1. ITC'99 designs

Design	***Source code***	***System model***	***Number of mutants***
B01	102	207	88
B02	70	143	48
B04	101	809	1342
B06	127	442	94
B07	92	370	784
B08	88	315	324
B09	100	263	284

Table 2 shows the test-related information: the length in ticks and the percentage of the killed mutants.

Table 2. Test generation methods comparison

Design	***ABC (ticks)***	***RETGA (ticks)***	***nuXmv (ticks)***	***ABC (%)***	***RETGA (%)***	***nuXmv (%)***
B01	227	49	37	90.91	98.86	90.9
B02	86	33	28	0	0	0
B04	—	36	56	—	99.93	99.93
B06	100	76	63	100	100	100
B07	133	166	162	0	0	0
B08	2745	52	31	98.77	79.94	44.44
B09	777	231	55	97.18	0	0

Comparison results are as follows. For some designs (B02 and B07), all methods achieve 0% stuck-at fault coverage. Such designs are classified as *untestable* [18]; their outputs are calculated in such a way that the internal stuck-at faults cannot affect their outputs. For some designs (B01, B04, and B06), the proposed method reaches the same or comparable stuck-at fault coverage as the leading one. Note that in such cases model-checking-based tests are usually shorter than tests produced by other methods. Finally, there are some designs (B08 and B09), where both nuXmv and RETGA reach lower stuck-at fault coverage than ABC. Additional efforts are needed to cope with this issue. A possible improvement is described below.

6. Future improvements

In our opinion, the main drawback of the proposed method and similar approaches is a lack of fault propagation. Broadly speaking, an EFSM model contains a stuck-at fault if some assignment ($v \coloneqq RHS$) of some transition is "corrupted" (RHS is substituted by 0 or 1). To activate the fault, a test should cause the transition to fire; however, this is not enough. The erroneous values should be propagated to the model outputs. Thus, the coverage model should be extended.

Given an EFSM model M, let us make some definitions. A variable v is *defined* in a transition x ($v \in Def_x$) if δ_x contains an assignment to v. A variable v is *used* in a transition y ($v \in Use_y$) if v appears either in γ_y or in the right-hand side of δ_y. A transition y *depends* on a transition x ($y \in DEP(x)$) if $Def_x \cap Use_y \neq \emptyset$. Depending on how v is used, in γ_y or in δ_y, they say about a *control dependency* ($y \in DEP_c(x)$) or a *data dependency* ($y \in DEP_d(x)$) respectively.

A *propagation path* for a transition t is a sequence of transitions $\{t_i\}_{i=0}^{n}$ such that:

1) $t_0 = t$;
2) $t_{i+1} \in DEP_d(t_i)$, for all $0 \leq i < n$;
3) $Def_{t_n} \cap O_M \neq \emptyset$.

Given a propagation path $\{(s_i, \gamma_i \to \delta_i, s_i')\}_{i=0}^{n}$, the propagation condition can be expressed as follows:

$$\varphi = \mathbf{F}\Big\{(c(s_0) \wedge \gamma_0) \wedge \mathbf{F}\big\{(c(s_1) \wedge \gamma_1) \wedge \mathbf{F}\{\dots \mathbf{F}\{c(s_n) \wedge \gamma_n\} \dots\}\big\}\Big\}.$$

Note that the notion of propagation path and the propagation condition can be refined so as to avoid variable redefinitions and other undesirable effects.

If there are no propagation paths for a given transition, the original coverage item, $\varphi_0 \equiv \mathbf{F}\{c(s_0) \wedge \gamma_0\}$, is removed. If there are multiple propagation paths, two main strategies can be applied:

1) *try all of the propagation paths*:
   a. split the coverage item φ_0 into the set of all possible fault propagation conditions: $\{\varphi_1, \dots, \varphi_m\}$;
2) *try at least one of the propagation paths*:
   a. replace the coverage item φ_0 with the disjunction $\bigvee_{i=1}^{m} \varphi_m$.

7. Conclusion

The primary scope of this work is reusing functional tests for manufacturing testing. The paper describes a high-level test generation approach and analyzes whether it is effective in detecting low-level faults. The approach implements two important facilities: automatic extraction of models from HDL descriptions and directed test generation based on model checking. The method is extremely flexible and can be customized by choosing a proper coverage model. The presented implementation tends to produce short tests with mediocre stuck-at fault coverage. We think that fault detection abilities of the approach can be increased by adding fault propagation conditions into coverage items. This may serve as a topic for future research.

Acknowledgment

The authors would like to thank Russian Foundation for Basic Research (RFBR). The reported study was supported by RFBR research project № 15-07-03834.

References

[1]. Bergeron J. Writing Testbenches: Functional Verification of HDL Models. Springer, 2003, 478 p. DOI: 10.1007/978-1-4615-0302-6.
[2]. Ivannikov V.P., Kamkin A.S., Kossatchev A.S., Kuliamin V.V., Petrenko A.K. The Use of Contract Specifications for Representing Requirements and for Functional Testing of Hardware Models. Programming and Computer Software, 33(5), 2007, pp. 47-61. DOI: 10.1134/s0361768807050039.
[3]. Mishra P., Dutt N. Specification-Driven Directed Test Generation for Validation of Pipelined Processors. ACM Transactions on Design. Automation of Electronic Systems, 13(3), 2008, pp 1-36. DOI: 10.1145/1367045.1367051.
[4]. Botros N.M. HDL Programming Fundamentals: VHDL and Verilog. Charles River Media, 2005, 506 p.

[5]. Berkeley Logic Interchange Format (BLIF). Berkeley, U.C., Oct Tools Distribution 2, 1992, pp. 197-247.
[6]. Melnichenko I., Kamkin A., Smolov S. An Extended Finite State Machine-Based Approach to Code Coverage-Directed Test Generation for Hardware Designs. Proceedings of ISP RAS, 2015, 27(3), pp. 161-182. DOI: 10.15514/ispras-2015-27(3)-12.
[7]. Smolov S., Lopez J., Kushik N., Yevtushenko N., Chupilko M., Kamkin A. Testing Logic Circuits at Different Abstraction Levels: An Experimental Evaluation. Proceedings of IEEE East-West Design Test Symposium (EWDTS), 2016, pp. 189-192. DOI: 10.1109/ewdts.2016.7807687.
[8]. Ferrandi F., Fummi F., Gerli L., Sciuto D. Symbolic Functional Vector Generation for VHDL Specifications. Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 1999, pp. 442-446. DOI: 10.1145/307418.307541.
[9]. Minato S. Generation of BDDs from Hardware Algorithm Descriptions. Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 1996, pp. 644-649. DOI: 10.1109/iccad.1996.571340.
[10]. Guglielmo G.D., Fummi F., Jenihhin M., Pravadelli G., Raik J., Ubar R. On the Combined Use of HLDDs and EFSMs for Functional ATPG. Proceedings of IEEE East-West Design and Test Symposium (EWDTS), 2007, pp. 503-508.
[11]. Brayton R., Mishchenko A. ABC: An Academic Industrial-Strength Verification Tool. Proceedings of the Computer Aided Verification Conference (CAV), 2010, pp. 24-40. DOI: 10.1007/978-3-642-14295-6_5.
[12]. Kamkin A., Lebedev M., Smolov S. An EFSM-Driven and Model Checking-Based Approach to Functional Test Generation for Hardware Designs. Proceedings of IEEE East-West Design and Test Symposium (EWDTS), 2016, pp. 60-63. DOI: 10.1109/ewdts.2016.7807736.
[13]. Smolov S., Kamkin A. A Method of Extended Finite State Machines Construction From HDL Descriptions Based on Static Analysis of Source Code. St. Petersburg State Polytechnical University Journal. Computer Science, Telecommunications, 1(212), 2015, pp. 60-73 (in Russian). DOI: 10.5862/jcstcs.212.6.
[14]. Retrascope site. http://forge.ispras.ru/projects/retrascope
[15]. Fortress library site. http://forge.ispras.ru/projects/solver-api
[16]. Cavada D., Cimatti A., Dorigatti M., Griggio A., Mariotti A., Micheli A., Mover S., Roveri M., Tonetta S. The nuXmv symbolic model checker. Proceedings of the 16th International Conference on Computer Aided Verification (CAV), LNCS, No.8559, 2014, pp. 334-342. DOI: 10.1007/978-3-319-08867-9_22.
[17]. ITC'99 benchmark site. http://www.cad.polito.it/tools/itc99.html
[18]. Liu X., Hsiao M.S. On Identifying Functionally Untestable Transition Faults. Proceedings of the Ninth IEEE International High-Level Design Validation and Test Workshop, 2004, pp. 121-126. DOI: 10.1109/hldvt.2004.1431252.

Генерация тестов для цифровой аппаратуры на основе высокоуровневых моделей

[1] *М.М. Чупилко <chupilko@ispras.ru>*
[1,2,3] *А.С. Камкин <kamkin@ispras.ru>*
[1] *М.С. Лебедев <lebedev@ispras.ru>*
[1] *С.А. Смолов <smolov@ispras.ru>*
[1] *Институт системного программирования 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.
[3] *Московский физико-технический институт,*
141701, Россия, Московская обл., г. Долгопрудный, Институтский пер., д. 9.

Аннотация. Тестирование аппаратуры — это процесс, нацеленный на обнаружение неисправностей, внесенных в интегральные схемы в процессе производства. Для оценки качества таких тестов используют две основные метрики: способность обнаруживать ошибки (покрытие ошибок) и время тестирования (длина теста). Известно множество методов генерации тестов, однако масштабируемого решения, применимого к сложной цифровой аппаратуре, нет до сих пор. В данной статье анализируется возможность использования функциональных тестов, построенных по высокоуровневым моделям (прежде всего, моделям уровня регистровых передач), для низкоуровневого производственного тестирования. Рассматривается конкретный метод генерации тестов, использующий технику проверки моделей (model checking). Входной информацией выступает HDL-описание. Метод состоит из двух ключевых шагов: построение модели системы и построение модели покрытия. Указанные модели автоматически извлекаются из HDL-описания. Модель системы представлена в виде высокоуровневых решающих диаграмм. Модель покрытия — это множество LTL-формул, определяющих условия достижимости переходов расширенного конечного автомата, описывающего систему. Построенные модели транслируются во входной формат инструмента проверки моделей (model checker), который для каждой формулы модели покрытия генерирует контрпример — вычисление, нарушающее эту формулу, то есть приводящее к срабатыванию соответствующего перехода автомата. Изначально рассматриваемый метод предназначался для покрытия всех путей исполнения HDL-описания и обнаружения недостижимого кода. Экспериментальное сравнение метода с существующими аналогами показало, что он строит более короткие тесты, однако эти тесты достигают меньшего уровня покрытия константных неисправностей, чем тесты, построенные с помощью специальных средств. В статье предлагается модификация метода для преодоления указанного недостатка.

Ключевые слова: цифровая аппаратура; язык описания аппаратуры; производственное тестирование; константная ошибка; высокоуровневая решающая диаграмма; расширенный конечный автомат; проверка модели; дерево распространения.

DOI: 10.15514/ISPRAS-2017-29(4)-16

Для цитирования: Чупилко М.М., Камкин А.С., Лебедев М.С., Смолов С.А. Метод генерации тестов для цифровой аппаратуры, основанный на высокоуровневых моделях. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 247-256 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-16

Список литературы

[1]. Bergeron J. Writing Testbenches: Functional Verification of HDL Models. Springer, 2003, 478 p. DOI: 10.1007/978-1-4615-0302-6.
[2]. Иванников В.П., Камкин А.С., Косачев А.С., Кулямин В.В., Петренко А.К. Использование контрактных спецификаций для представления требований и функционального тестирования моделей аппаратуры. Программирование, т. 33, № 5, 2007 г., стр. 272-282.
[3]. Mishra P., Dutt N. Specification-Driven Directed Test Generation for Validation of Pipelined Processors. ACM Transactions on Design. Automation of Electronic Systems, 13(3), 2008, pp 1-36. DOI: 10.1145/1367045.1367051.
[4]. Botros N.M. HDL Programming Fundamentals: VHDL and Verilog. Charles River Media, 2005, 506 p.
[5]. Berkeley Logic Interchange Format (BLIF). Berkeley, U.C., Oct Tools Distribution 2, 1992, pp. 197-247.
[6]. Melnichenko I., Kamkin A., Smolov S. An Extended Finite State Machine-Based Approach to Code Coverage-Directed Test Generation for Hardware Designs. Proceedings of ISP RAS, 2015, 27(3), pp. 161-182. DOI: 10.15514/ispras-2015-27(3)-12.
[7]. Smolov S., Lopez J., Kushik N., Yevtushenko N., Chupilko M., Kamkin A. Testing Logic Circuits at Different Abstraction Levels: An Experimental Evaluation. Proceedings of IEEE East-West Design Test Symposium (EWDTS), 2016, pp. 189-192. DOI: 10.1109/ewdts.2016.7807687.
[8]. Ferrandi F., Fummi F., Gerli L., Sciuto D. Symbolic Functional Vector Generation for VHDL Specifications. Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 1999, pp. 442-446. DOI: 10.1145/307418.307541.
[9]. Minato S. Generation of BDDs from Hardware Algorithm Descriptions. Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 1996, pp. 644-649. DOI: 10.1109/iccad.1996.571340.
[10]. Guglielmo G.D., Fummi F., Jenihhin M., Pravadelli G., Raik J., Ubar R. On the Combined Use of HLDDs and EFSMs for Functional ATPG. Proceedings of IEEE East-West Design and Test Symposium (EWDTS), 2007, pp. 503-508.
[11]. Brayton R., Mishchenko A. ABC: An Academic Industrial-Strength Verification Tool. Proceedings of the Computer Aided Verification Conference (CAV), 2010, pp. 24-40. DOI: 10.1007/978-3-642-14295-6_5.
[12]. Kamkin A., Lebedev M., Smolov S. An EFSM-Driven and Model Checking-Based Approach to Functional Test Generation for Hardware Designs. Proceedings of IEEE East-West Design and Test Symposium (EWDTS), 2016, pp. 60-63. DOI: 10.1109/ewdts.2016.7807736.
[13]. Смолов С., Камкин А. Метод построения расширенных конечных автоматов по HDL-описанию на основе статического анализа кода. Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление, 1(212), 2015, стр. 60-73. DOI: 10.5862/jcstcs.212.6.
[14]. Страница инструмента Retrascope. http://forge.ispras.ru/projects/retrascope (дата обращения: 18.07.17)
[15]. Страница библиотеки Fortress. http://forge.ispras.ru/projects/solver-api (дата обращения: 18.07.17)
[16]. Cavada D., Cimatti A., Dorigatti M., Griggio A., Mariotti A., Micheli A., Mover S., Roveri M., Tonetta S. The nuXmv symbolic model checker. Proceedings of the 16th International Conference on Computer Aided Verification (CAV), LNCS, No.8559, 2014, pp. 334-342. DOI: 10.1007/978-3-319-08867-9_22.
[17]. Страница набора тестов ITC'99. http://www.cad.polito.it/tools/itc99.html (дата обращения: 18.07.17)
[18]. Liu X., Hsiao M.S. On Identifying Functionally Untestable Transition Faults. Proceedings of the Ninth IEEE International High-Level Design Validation and Test Workshop, 2004, pp. 121-126. DOI: 10.1109/hldvt.2004.1431252.