Test Generation for Digital Hardware Based on High-Level Models

M.M. Chupilko <chupilko@ispras.ru>
A.S. Kamkin <kamkin@ispras.ru>
M.S. Lebedev <lebedev@ispras.ru>
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 to cover 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

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: $D_V \rightarrow \{0, 1\}$. An action is a transformation of valuations: $D_V \rightarrow D_V$. A pair $\gamma \rightarrow \delta$, where $\gamma$ is a guard and $\delta$ 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 \( M = (S_M, \mathcal{V}_M, T_M) \), where \( S_M \) is a set of states, \( \mathcal{V}_M = (I_M \cup O_M \cup R_M) \) 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 \((s_i, \gamma_t, s'_t, \delta_t)\), where \( s_i \) and \( s'_t \) are respectively the initial and the final state of \( t \), whereas \( \gamma_t \) and \( \delta_t \) are respectively the guard and the action of \( t \).

A pair \((s, v) \in S_M \times D_{\mathcal{V}_M}\) 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: \((s, v) \equiv (s_0, v_0)\), where \((s_0, v_0)\) is a predefined configuration. On every tick, it computes the set of enabled transitions:

\[
T_E := \{ t \in T_M \mid (s_t = s) \land (\gamma_t(v) = 1) \}.
\]

A single transition \( t \in T_E \) (chosen nondeterministically) fires: \((s, v) \equiv (s'_t, \delta_t(v))\).

A netlist is a tuple \( N = (\mathcal{V}_N, G_N, L_N) \), where \( \mathcal{V}_N \) 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 \((I_g, o_g, f_g)\), where \( I_g \subseteq \mathcal{V}_N \) and \( o_g \subseteq \mathcal{V}_N \) are respectively the inputs and the output of \( g \), and \( f_g : \text{Dom}(I_g) \to \{0, 1\} \) is the function of \( g \). A latch \( l \in L_N \) is a tuple \((i_l, o_l)\), where \( i_l \in \mathcal{V}_N \) and \( o_l \in \mathcal{V}_N \) 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 overviewes 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.

![Model checking-based approach to test generation for HDL descriptions](image)

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:

\[
F(c(s) \land \gamma_t),
\]

where \( F(x) \) means that \( x \) will eventually be true.

In accordance with [12], the system model and the coverage items in the negated form \((\sim F(c(s) \land \gamma_t))\), are translated into the input format of a model checker. For each
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 2 shows the test-related information: the length in ticks and the percentage of the killed mutants.
Given a propagation path \(\{(s_i, y_i \rightarrow \delta_i, s'_i)\}\) of length \(n\), the propagation condition can be expressed as follows:

\[
\varphi = \bigwedge_{i=1}^{n} \{\mathcal{F}[\mathcal{F}(s_0) \land y_0] \land \mathcal{F}[\mathcal{F}(s_1) \land y_1] \land \ldots \land \mathcal{F}[\mathcal{F}(s_n) \land y_n] \}.
\]

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, \(\varphi_0 \equiv \mathcal{F}[\mathcal{F}(s_0) \land y_0]\), 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 \(\varphi_0\) into the set of all possible fault propagation conditions: \(\{\varphi_1, \ldots, \varphi_m\}\);

2) **try at least one of the propagation paths:**
   a. replace the coverage item \(\varphi_0\) 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 strategies:

1. **Automatic Extraction of Models from HDL Descriptions**: This capability is extremely flexible and can be applied to various HDL descriptions, allowing for the automatic generation of test models.
2. **Directed Test Generation**: The method is based on model checking and can be used to detect faults at different abstraction levels. It is a topic for future research, especially in the context of HDL models.

### 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


Генерация тестов для цифровой аппаратуры на основе высокоуровневых моделей

1. М. Чупило <chupilko@ispras.ru>
2. А. С. Камкин <kamkin@ispras.ru>
3. 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 масштабирующего решения, применимого к сложной цифровой аппаратуре, нет до сих пор. В данной статье анализируется возможность использования концепции формализованной верификации для построения высокоуровневых тестов, построенных с помощью специальных средств. В статье предлагается модификация метода для преодоления указанного недостатка. Построенные 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