J Electron Test (2012) 28:511–521
DOI 10.1007/s10836-012-5312-5

Identifying Untestable Faults in Sequential Circuits Using Test Path Constraints

Taavi Viilukas · Anton Karputkin · Jaan Raik · Maksim Jenihhin · Raimund Ubar · Hideo Fujiwara

Received: 14 November 2011 / Accepted: 26 June 2012 / Published online: 13 July 2012

Abstract The paper proposes a hierarchical untestable stuck-at fault identification method for non-scan synchronous sequential circuits. The method is based on deriving, minimizing and solving test path constraints for modules embedded into Register-Transfer Level (RTL) designs. First, an RTL test pattern generator is applied in order to extract the set of all possible test path constraints for a module under test. Then, the constraints are minimized using an SMT solver Z3 and a logic minimization tool ESPRESSO. Finally, a constraint-driven deterministic test pattern generator is run providing hierarchical test generation and untestability proof in sequential circuits. We show by experiments that the method is capable of quickly proving a large number of untestable faults obtaining higher fault efficiency than achievable by a state-of-the-art commercial ATPG. As a side effect, our study shows that traditional bottom-up test generation based on symbolic test environment generation at RTL is too optimistic due to the fact that propagation constraints are ignored.

Keywords Automated test pattern generation · Untestable faults · Register-transfer level

Responsible Editor: C. Metra

T. Viilukas · A. Karputkin · J. Raik (✉) · M. Jenihhin · R. Ubar
Department of Computer Engineering,
Tallinn University of Technology,
Raja 15,
Tallinn 12618, Estonia
e-mail: jaan@pld.ttu.ee

T. Viilukas
e-mail: viilukas@ati.ttu.ee

A. Karputkin
e-mail: kanton@ati.ttu.ee

M. Jenihhin
e-mail: maksim@ati.ttu.ee

R. Ubar
e-mail: raiub@ati.ttu.ee

H. Fujiwara
Faculty of Informatics, Osaka Gakuin University,
Suita, Osaka 564-8511, Japan
e-mail: fujiwara@ogu.ac.jp

1 Introduction

Test generation for sequential synchronous circuits is a time-consuming task. Automated Test Pattern Generation (ATPG) tools spend a lot of effort not only for deriving test vectors for testable faults but also for proving that there exist no tests for the untestable faults. Because of this reason, the identification of untestable faults has been an important aspect in speeding up the sequential ATPG. The percentage of untestable faults in sequential circuits tends to be considerably higher than in the combinational ones. For combinational circuits, untestable faults occur due to the redundant logic in the circuit, while for sequential circuits untestable faults may also result due to unreachable states or due to impossible state transitions.

A number of works have been proposed in order to tackle the problem of identifying sequentially untestable faults. The first methods [1] were fault-oriented and based on applying combinational ATPG to the expanded time-frame model of the sequential circuit. However, such approach does not scale because of the size-explosion of the unrolled sequential models. Thus, the fault independent method was introduced by Iyer et al. in [10]. The new algorithm was called FIRES and it implemented illegal state information to complement redundancy analysis. This was followed by a number of fault independent methods including MUST [16], FUNI [14], FILL [14] and others. Liang [13] proposed a simulation based approach for sequential untestable fault

identification. However, it was shown in [14] that this method may result in 'false positives', i.e. a fault may be declared untestable when there actually exists a test for it. The common limitation of the above methods is that they operate at the logic-level representation of the design. Thus a considerable amount of effort is put on the implication process carried out at the level of logic netlists.

In their previous work [17], the authors introduced a specific subclass of sequentially untestable faults, called register enable stuck-on faults and a method for proving them untestable using a model checker. In this paper we propose a hierarchical untestability identification method. The new method allows detecting sequential untestability in combinational modules (functional units, multiplexers) embedded into a hierarchical circuit and is based on path activation constraints extracted by a Register-Transfer Level (RTL) ATPG.

In hierarchical RTL test generation, top-down and bottom-up strategies are known. In the bottom-up approach, tests generated at the low-level will be later assembled at the higher abstraction level. Such algorithms yield short runtimes but ignore the incompleteness problem: constraints imposed by other modules and/or the network structure may prevent test vectors from being assembled. In the top-down approach, constraints are extracted at the higher level as a goal to be considered when deriving tests for modules at the lower level. This approach allows testing modules embedded deep into the RTL structure. However, as modules are often tested through highly complex constraints, their fault coverage may be compromised.

Early methods on bottom-up RTL testing relied on the assembly of module tests and were applicable of the simplest systems only [15]. A more solid basis for the bottom-up paradigm was laid by Ghosh et al. in [7]. In their work, test environments are generated for each functional module of a given functional RTL circuit described in an Assignment Decision Diagram (ADD) [3] using symbolic justification/propagation rules using a nine-valued algebra. In this method, a test sequence is then formed by substituting the corresponding test patterns into the test environment. However, the proposed nine-valued algebra cannot guarantee the generation of a test environment, even if it exists. To overcome this drawback, Zhang et al. upgraded the nine-valued algebra to a ten-valued algebra by taking the variable line value range into consideration. This algebra is able to generate much more test environments [23]. In [6], Zhang's approach has been further improved by introducing additional propagation rules.

Lee and Patel introduced top-down constraint-based test pattern generation for microprocessors in [12]. Several constraint-based top-down approaches followed, including [19, 21]. [11] proposed a bottom-up approach based on a High-Level Decision Diagram (HLDD) engine and on applying a commercial constraint solver SICStus. As experiments show, the tool achieves lower fault coverage in comparison to a commercial logic-level Automated Test Pattern Generator (ATPG). In [22], a top-down approach including a constraint solving package ECLiPSe [20] has been proposed.

None of the previous methods apply RTL constraints in order to prove logic-level untestable faults. Thus, the fault efficiency reported by the approaches [11, 12, 19, 21, 22] is often low, which decreases the test engineer's confidence in the test. Here, by fault efficiency we refer to the ratio of the number of tested faults to the number of testable faults. In addition, as we will show in this paper, in many cases fault coverage obtained for the modules in RTL test generation may considerably decrease if test path constraints are being ignored.

The authors of this paper previously introduced an RTL untestability identification method where a specific class of untestable register control faults was proven untestable by applying model-checking at the RTL [17]. In this paper, we present a hierarchical untestability identification method for the general case of sequentially untestable stuck-at faults within RTL modules. First, an RTL test pattern generator is applied in order to extract the set of all possible test path activation constraints for a module under test within a certain clock cycle limit. Then, the constraints are minimized and a constraint-driven deterministic test pattern generator is run providing a time-limit-bounded hierarchical test generation and untestability proof for sequential circuits. This general concept has been published in [18]. However, in this paper we have formalized and implemented the test path constraint minimization step. We have evaluated the minimization method and studied the complexity of minimization on several constraint sets.

We show by experiments that the tool is capable of quickly proving a large number of untestable faults obtaining higher fault efficiency than achievable by a state-of-the-art commercial ATPG. As a side effect, our study shows that traditional bottom-up test generation based on symbolic test environment generation at RTL is too optimistic due to the fact that propagation constraints are ignored.

Figure 1 presents the corresponding top-down test flow for targeting a Module Under Test (MUT) in a hierarchical RTL design. The flow contains three main phases. During the first phase, the full set of constraints for setting up a test path to test an RTL module are extracted on the RTL design representation. We apply RTL ATPG Decider [22] in order to extract the constraints for accessing the MUT. Decider activates as many sets of constraints as there are test paths for that module in a bounded limit of clock-cycles. In [22], test constraints were utilized to propagate test patterns to and from the MUT. However, in this paper the purpose is to process the set of constraints in order to derive conditions for a dedicated logic-level ATPG in proving untestability.

During the second phase this set of constraints is minimized as presented in Section 3 resulting in a compact test environment for accessing the MUT. The test environment

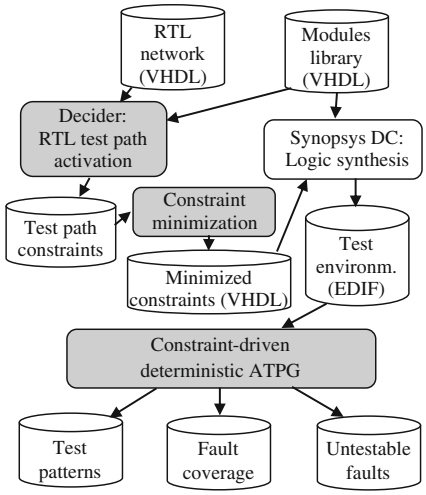


Fig. 1 Constraint-based untestability proof flow

is translated into VHDL and synthesized to logic-level using Synopsys Design Compiler.

The third phase generates deterministic tests to the logic-level module taking into account the minimized path constraints. Here, the constraint-driven logic-level ATPG is run on the logic-level description of the MUT instantiated into the synthesized test environment. As a result we obtain the list of sequentially untestable faults in the MUT as well as test patterns for the entire design.

An RTL design contains a large number of modules to be tested. By far not all of them are affected by sequential untestability issues. Therefore, the method requires a pre-processing step that would identify the modules for which untestability analysis should be carried out. This can be done e.g. by selecting modules whose fault coverage is below some threshold.

The rest of the paper is organized as follows. Section 2 presents the concept of test path constraints. Section 3 discusses minimization of the constraints. In Section 4 the constraint based top-down hierarchical test generation for proving untestable stuck-at faults is presented. In addition, an example explaining the limitations of bottom-up hierarchical test generation approaches with respect to top-down ones is presented. Section 5 provides experimental results. Sections 6 provide the limitations and threats to validity. The paper ends with Conclusions.

2 Test Path Constraints Extraction at the RTL

In this Section, we define the Register-Transfer Level (RTL) representation of a digital system. We introduce the concept of test path constraints for testing a module in the RTL design and describe the procedure of extracting them by the algorithm implemented in the hierarchical test pattern generator DECIDER [22].

2.1 RTL Representation

Figure 2 presents a structural RTL view of a digital system. At the RTL, the design is assumed to be partitioned into a control part and a datapath. An RTL digital system $S=<F, X>$ is regarded as a network of interconnected blocks, or modules, where F is the set of modules in the network and X is the set of variables connecting the modules. The control part is represented by a Finite State Machine (FSM) consisting of three modules: a state register $f_S \in F$ with an output variable $x_S \in X$, a next state logic module $f_T \in F$ with an output variable $x_T \in X$ and an FSM output logic module $f_G \in F$ with primary outputs $X_O \subset X$ and control variables $X_C \subset X$ as its outputs, respectively. Input variables to the control part are comprised of the primary inputs of the design (variables $X_I \subset X$), status bit variables originating from the datapath $X_N \subset X$ and current value of the state variable x_S. Outputs of the control part consist of the primary outputs X_O of the design, control signal variables X_C and the next value of the state register variable X_S.

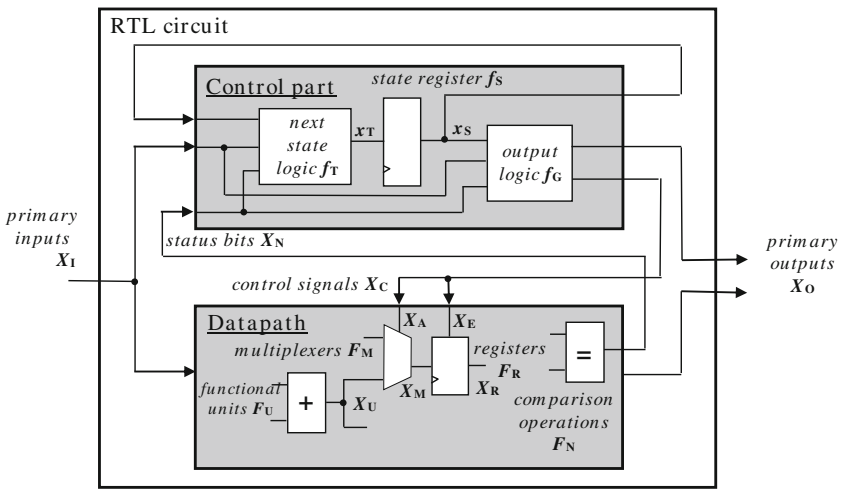


Fig. 2 Register-transfer level view of a digital circuit

Similarly, the datapath is regarded as a network consisting of interconnected modules. The modules include registers $F_R \subset F$ with output variables $X_R \subset X$, multiplexers $F_M \subset F$ with outputs $X_M \subset X$, Functional Units (FUs) for implementing arithmetic operations $F_U \subset F$ with outputs $X_U \subset X$ and comparison operator modules $F_N \subset F$ with outputs X_N, respectively. As inputs for the datapath are the primary inputs X_I and control signal variables X_C. The latter are partitioned into multiplexer addresses $X_A \subset X_C$ and register enable variables $X_E \subset X_C$. Outputs are the primary outputs X_O as well as the status bit variables X_N from comparison operator modules leading to the control part. The datapath of the RTL is structured according to the so-called mux-FU-mux-register architecture, where the first level of multiplexers select operands for the operations implemented by the functional units followed by another level of multiplexers for selecting the operation results to be read by the next stage of registers.

2.2 Extraction of Test Path Constraints

For each datapath Module Under Test (MUT), we extract control part FSM state sequences in order to propagate fault effects from the output of the MUT to primary outputs and to propagate the values from the primary inputs to the inputs of the MUT. Such state sequences constitute test paths for accessing MUT. We represent the test paths by sets of constraints. All test paths within a certain cycle-limit are activated and the corresponding constraints extracted by the proposed algorithm. This cycle-limit is first set to 1 and then gradually incremented until the obtained constraints will be non-empty after the minimization. In order to extract the RTL test path constraints in current paper, a test path activation tool DECIDER [22] is applied.

The concept of the constraints for a single test path for a datapath MUT is visualized in Fig. 3. The test path constraints are divided into three categories. These are the set of *path activation constraints* C_A, the *transformation constraints* C_J and the *propagation constraint* c_P, respectively. Path activation constraints correspond to the conditions in the FSM state transitions that have to be satisfied in order to perform propagation and value justification through the circuit. Transformation constraints, in turn, reflect the value changes along the paths from the inputs of the high-level MUT to the primary inputs of the whole circuit. These constraints are needed in order to derive the local test patterns for the module under test. The propagation constraints show how the value propagated from the output of the MUT to a primary output is depending on the values of the primary inputs. The main idea here is to check whether the fault effect will be masked when propagated to a primary output. All the above categories of constraints are represented by common data structures and manipulated by common procedures for creation, update, modeling and simulation. In the following, the data structure and update operations of test path constraints are defined.

Definition 1: A condition c that is $d=g(X')$, where d is a bitvector or Boolean constant or a variable $x \in X$, and $g(X')$ is a logic, arithmetic or comparison expression on a subset of variables $X' \subset X$, is referred to as a *test path constraint*. From this point on, we refer to test path constraints as constraints.

Definition 2: Constraint c: $d=g(X')$ is said to be *justified* iff $X' \subseteq X_I$, where X_I is the set of primary inputs of the system. Otherwise, c is said to be an *unjustified* constraint.

Definition 3: If a constraint c: $d=g(X')$ is unjustified then all the variables in the set X' that are not input variables X_I are said to be *unjustified variables* of the constraint. The input variables belonging to the constraint are called *justified variables*.

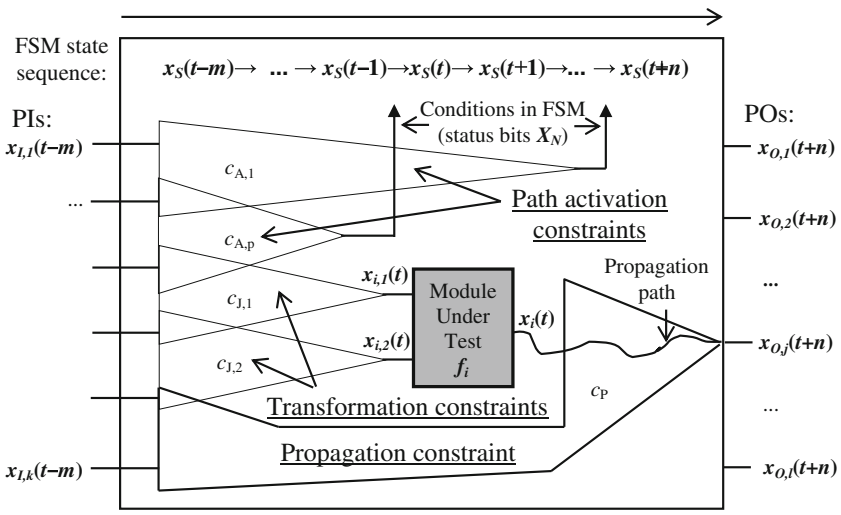


Fig. 3 An unrolled RTL circuit with test generation constraints for a test path for a MUT

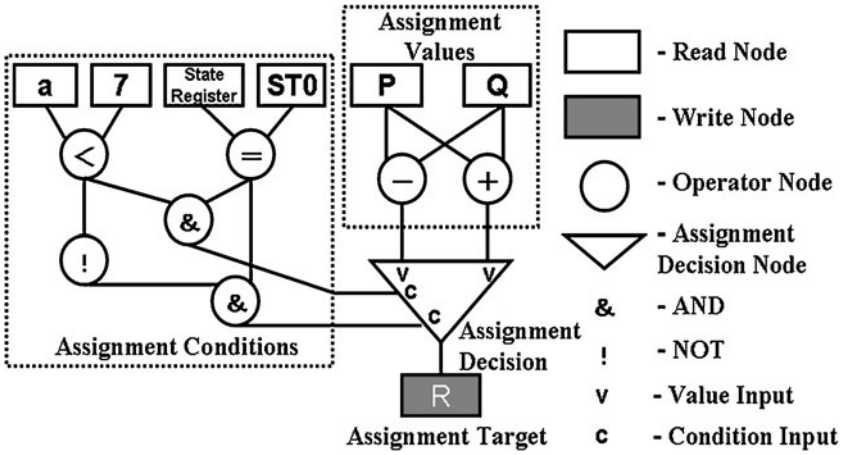


Fig. 4 Assignment Decision Diagram (ADD)

Definition4: Let X' be the set of justified variables and X'' be the set of unjustified variables of a constraint c: $d=g(X', X'')$. The process, where each variable $x''_i \in X''$ is substituted by an expression $h_i(X'''_i)$ on model variables $X'''_i \subseteq X$, is referred to as *updating the constraint c*.

Consider the general case of test path constraints for a MUT presented in Fig. 3. Such constraints are extracted as follows. First, the value from the output variable x_i of the MUT f_i is propagated to a primary output $x_{O,j}$ by activating a state sequence $x_S(t) \rightarrow x_S(t+1) \rightarrow \ldots \rightarrow x_S(t+n)$ in the control part. Here, by $x(t)$ we denote the value of variable x at the clock-cycle t. Thus, the propagation state sequence starts at a time step t, which is referred to as the *manifestation step*, and it ends at a clock-cycle $t+n$. During propagation, path activation constraints $c_{A,p} \in C_A$ are created at time steps where the next state value of x_S is depending on the status bits X_N. When the fault effect value propagates from x_i to $x_{O,j}$ at the time step $t+n$ then the propagation constraint c_P is created.

Subsequent to propagation, the constraint justification process begins. Starting from the time step $t+n$, we move backward in time until the manifestation step t is reached. At each time step we update the propagation constraint c_P and those path activation constraints $c_{A,p}$ whose creation time step is later than current time step. During the update, the unjustified variables $X'' \subseteq X_R$ of the constraint expressions $g(X', X'')$ for all the constraints are substituted by expressions $h_i(X'''_i)$ on model variables $X'''_i \subseteq X_R \cup X_I$, where $h_i(X'''_i)$ are the expressions implemented by functional units F_U selected according to the values of control signal variables X_C at the current time step.

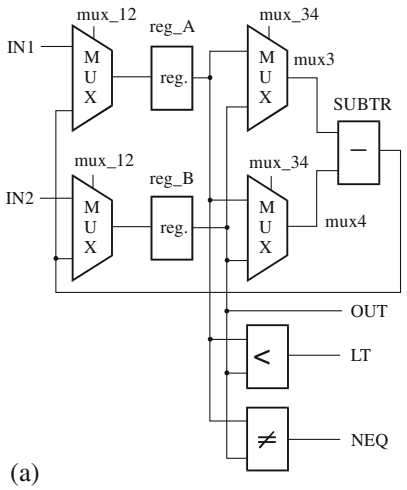
At the manifestation time step t, we create the transformation constraints for each input of the MUT. Without loss of generality, Fig. 3 shows a MUT with two inputs $x_{i,1}$ and $x_{i,2}$. Thus, in current case the transformation constraints $c_{J,1} \in C_J$ and $c_{J,2} \in C_J$ are created, respectively. We continue moving backwards in time until at some time step $t-m$ all the variables in the constraints are primary inputs X_I. During this process we update all the created constraints and create new path activation constraints $c_{A,p}$ at time steps where the previous state value of x_S is depending on the status bits X_N.

Note, that the extracted constraints contain expressions $g(X)$ on primary inputs X_I and constants. (In the case of the propagation constraint c_P the expression also depends on the MUT output x_i). The expressions are determined by the functions implemented by functional units F_U and, in the case of path activation constraints $c_{A,p}$, also by comparison operations F_N. The exponential size complexity of the constraints expression $g(X)$ is avoided by uniting multiple occurrences of the same variable (i.e. the literals) in the constraints at each time step into one single fanout variable. Because of this, the size requirements for the constraints are linear with respect to justification time-frames and they represent a subset of the expanded time-frame model of the circuit.

Finally, consistency of test paths is verified by applying constraint solving to the extracted path activation constraints $c_{A,p}$. We have selected the open source ECLiPSe constraint solver (ECLiPSe5.10_41) [20] for this purpose. After one consistent set of test path constraints are extracted by Decider, a backtrack occurs and the tool attempts to use alternative propagation and justification paths. The process ends when all

```
A := IN1;
B := IN2;
while (A    ≠ B)
  if (A < B) then
    B := B – A;
  else
    A := A – B;
  end if;
end while;
OUT := A;
```

Fig. 5 HDL description of the GCD algorithm



RESET	LT	NEQ	Pres. State	next state	A_enabl	B_enable	mux_12	mux_34
1	X	X	X	S_0	1	1	0	X
0	X	1	S_0	S_1	0	0	X	X
0	X	0	S_0	S_0	0	0	X	X
0	1	X	S_1	S_2	0	0	X	X
0	0	X	S_1	S_3	0	0	X	X
0	X	X	S_2	S_0	0	1	1	1
0	X	X	S_3	S_0	1	0	1	0

(b)

Fig. 6 RT-level architecture of the GCD circuit

the consistent test paths within a certain time-step limit are activated and respective test path constraints are extracted.

3 Minimization of Test Path Constraints

In this Section we explain minimization of the test path constraints for a MUT. The minimization step is required due to the fact that the full set of test path constraints, may become large considering representing them in VHDL and performing logic synthesis on them. The latter is needed to handle the constraints by the gate-level ATPG for proving untestable faults. (See Fig. 1 for the untestability identification flow).

Every test path $p_i \in P$, with P being the set of all the test paths for a MUT within a given time-frame, may be represented as a triple $\langle c_{P,i}, C_{J,i}, C_{A,i} \rangle$, where $c_{P,i}$ is the propagation constraint, $C_{J,i}$ is the set of justification constraints and $C_{A,i}$ is the set of path activation constraints extracted for the test path p_i, respectively. We can represent the full set of test paths P by a formula Φ in Disjunctive Normal Form (DNF), where terms correspond to the test paths p_i and literals are the constraints $c_{i,j}$ belonging to the test paths and represented as quantifier-free bitvector (QFBV) predicates. The three groups of constraints $c_{P,i}$, $C_{J,i}$ and $C_{A,i}$ are each minimized separately.

Minimization of the DNF formula Φ takes place as follows. First of all, some constraints in the test paths can be redundant. In order to remove such redundancies we apply a method presented in [5] and briefly described here. Consider a first-order logic formula Φ given in a negation normal form. First, we build a tree where intermediate nodes represent either $\vee$ or $\wedge$ operations and leafs represent QFBV predicates. The idea is to test each leaf L against a special formula α_L, called the *critical constraint*. If $\alpha_L \Rightarrow L$ then L can be replaced by true, and if $\alpha_L \Rightarrow \neg L$ then L can be replaced by false. Assume, for example, that Φ is presented in DNF:

$$\Phi = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} L_{ij}$$

Then, for a leaf L_{kl}, $1 \le k \le n$, $1 \le l \le m_k$,

$$\alpha_{L_{kl}} = \left(\bigwedge_{\substack{i=1 \\ i \neq k}}^{n} \bigvee_{j=1}^{m_i} \neg L_{ij} \right) \wedge \left(\bigwedge_{\substack{i=1 \\ i \neq l}}^{m_k} L_{ki} \right)$$

To test whether α_L implies L or $\neg L$ we use an SMT solver Z3 [4].

Example 1: Consider the DNF formula $(x = 1) \wedge x > 0 \vee x + y = 3 \wedge y > 4 \wedge x > 0$. Let us test the

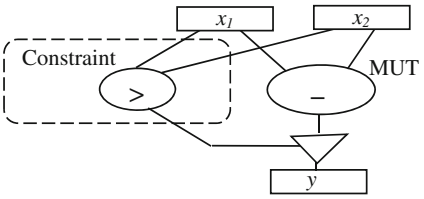


Fig. 8 Constraint-based test environment for MUT

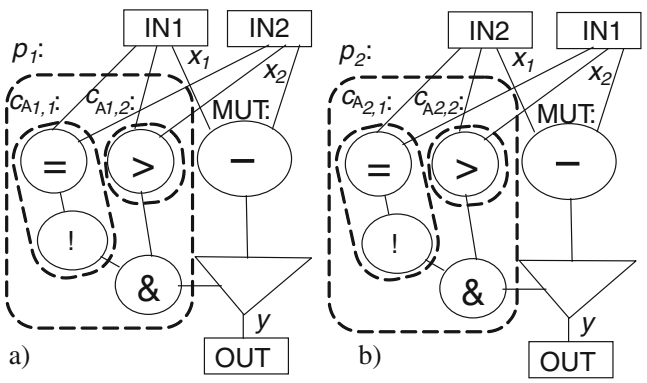


Fig. 7 Full set of test path constraints for MUT

Table 1 Number of leaves in the DNF as a function of the cycle limit k

k	Number of leaves in the DNF		
	b04 (AVERAGE1)	b04 (AVERAGE2)	gcd (SUBTR)
1	0	0	0
2	20	10	2
3	1216	610	14
4	50867	25408	54
8	N/A	N/A	444

leaf L_{21}=x+y=3. The critical constraint for that leaf would be $\alpha_{L_{21}} = y > 4 \wedge x > 0 \wedge (x \neq 1 \vee x \leq 0)$. It appears that $\alpha_{L_{21}} \Rightarrow x + y \neq 3$, so we can replace x+y=3 by false and remove the second conjunct. The remaining formula can be simplified a bit more, because (x=1)⇒x>0 and we can replace x>0 by true. Thus, it appears that ((x=1) ∧ x>0 ∨ x+y=3 ∧ y>4 ∧ x>0)=(x=1).

Second, we minimize the propositional skeleton of the remaining formula (a Boolean expression where all predicates are replaced by propositional variables) using a state-of-the-art algorithm ESPRESSO [2].

4 Constraint-Driven ATPG for Proving Untestability

4.1 Assignment Decision Diagrams

The minimized set of test paths *P* obtained by the constraints extraction defined in Section 2 and constraint minimization presented in Section 3 forms the *test environment* for the constraint-driven ATPG. In the following examples, we use the assignment decision diagram data structures in order to illustrate the test path constraints.

Assignment decision diagram (ADD) [3] is an acyclic graph that consists of a set of nodes that can be categorized into four types: read node, write node, operation node and assignment decision node (ADN), and a set of edges which contain the connectivity information between two nodes (Fig. 4). A read node represents a primary input port, a storage unit or a constant while a write node represents a primary output port or a storage unit. An operation node expresses an arithmetic operation unit or a logic operation unit while an ADN selects a value from a set of values that are provided to it based on the conditions computed by the logic operation units. If one of the condition inputs becomes true, the value of the corresponding data input will be selected.

4.2 Constraint-Driven ATPG Example

Figure 5 presents a VHDL code fragment of the Euclidean algorithm for calculating the Greatest Common Divisor (GCD) of two unsigned variables IN1 and IN2. An RTL architecture implementing this algorithm is shown in Fig. 6, with Fig. 6a presenting the datapath and Fig. 6b presenting the state table of the control part.

Without entering into further details, consider Fig. 7, which gives the ADD for the full set of constraints *P* extracted for the GCD example. In other words, the MUT can only be tested using one of the two test paths presented in Fig. 7a and b. The two test paths contain only path activation constraints and the paths are identical except for the fact that the primary inputs IN1, IN2 are swapped in them.

Note, that from the point of view of accessing the MUT these two environments are equivalent. It is irrelevant which primary input is used in applying the test patterns when representing the constraint-based test environment for proving untestability. Therefore, we denote the value justified from the *k*-th input of the MUT by x_k and the value propagated from the MUT output by *y*.

The test paths p_1 and p_2 both consist of two path activation constraints $c_{A1,1}$, $c_{A1,2}$ and $c_{A2,1}$, $c_{A2,2}$,

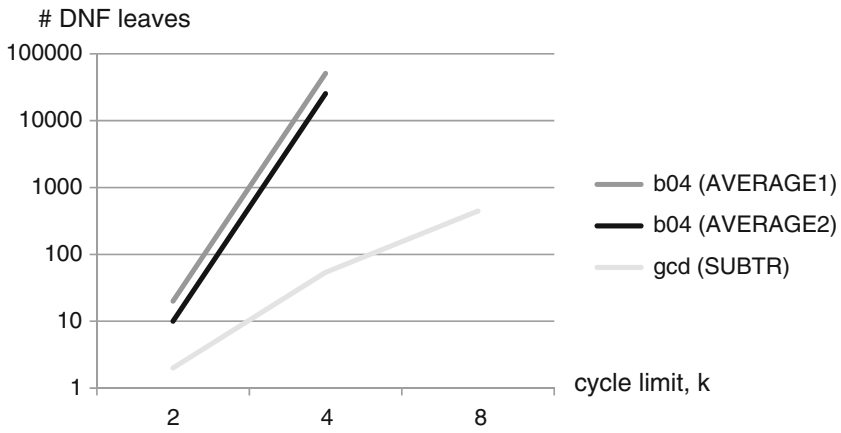


Fig. 9 Number of leaves in the DNF as a function of the cycle limit *k*

Table 2 Benchmark characteristics

Circuit	#faults	PI bits	PO bits	# reg. ($\|F_R\|$)	# Mux ($\|F_M\|$)	# FU ($\|F_U\|$)	Time limit
gcd	472	33	16	3	4	3	5
mult8x8	2356	17	16	7	4	9	8
diffeq	10326	81	48	7	9	5	7

respectively. $c_{A1,1}$ (which is equivalent to $c_{A2,1}$) states that x_1 must not be equal to x_2. $c_{A1,2}$ (equivalent to $c_{A2,2}$) states that x_1 must be greater than x_2. Since all the path activation constraints $c_{i,j}$ within a test path should hold simultaneously they are combined using the conjunction operator. In turn, all the test paths p_i are combined using the disjunction operation because any one of them may be applied for accessing the MUT. Therefore, we can combine the constraints into a Disjunctive Normal Form (DNF) as follows: $\bigvee_i \bigwedge_j c_{i,j}$.

Subsequent to combining the test path constraints the constraint minimization is performed. Using the method presented in previous Section we obtain for the example in Fig. 7:

$$(x_1 \neq x_2) \wedge (x_1 > x_2) \vee (x_1 \neq x_2) \wedge (x_1 > x_2) = x_1 > x_2.$$

Figure 8 shows the ADD for the minimized test environment resulting for testing the MUT of the non-minimized constraints example presented in Fig. 7. The constraint shows that the MUT (a subtractor) may only be accessed when the first input of it, i.e. x_1 is greater than the second one, x_2.

The obtained test environment, excluding the MUT, is automatically translated into VHDL and synthesized to logic-level using Synopsys Design Compiler. MUT is linked by instantiating its logic level description into the VHDL of the test environment. Subsequently, the constraint-driven logic-level ATPG is run. As a result we obtain the list of sequentially untestable faults in the MUT as well as test patterns for testing the MUT.

4.3 Discussion on the Effect of the Top-Down Proof

As a side-effect, the test environment allows us to evaluate the accuracy of bottom-up hierarchical ATPG. In particular, the strict interpretation of Ghosh's algebra [7] leads to overly pessimistic results because tests for some MUTs are aborted due to justification conflicts. On the other hand, the weak interpretation is too optimistic and can also lead to loss of fault coverage because some of the test patterns that are expected to cover faults in the MUT do not propagate.

Consider the case where in a bottom-up scenario we have a deterministic test T_q generated for the MUT in a stand-alone mode reaching the maximum fault coverage W_q for the module under test. Then, we generate the test environment for the module and substitute T_q into this test environment. Due to the test path constraints the actual fault coverage that can be achieved for the MUT embedded inside the network is W_a, which is generally lower than the stand-alone fault coverage W_q. However, when we fault simulate T_q substituted into the test environment we obtain a fault coverage W_r, where $W_r \leq W_a \leq W_q$.

In other words, the bottom-up approach may lose some fault coverage with respect to the top-down one because the set of the tests to choose from is restricted to T_q. If the bottom-up test generation algorithm for the MUT had had some knowledge about the test path constraints it would have generated a different test T_a, whose fault coverage would have been equal to W_a. Thus, a deterministic ATPG taking into account the test path constraints is necessary in order to achieve maximum fault coverage and also to prove untestability within sequential circuits. Experiments with the constraint-driven deterministic ATPG presented in Section 5 show that the difference between the coverages W_r and W_a may be even as high as 8–14 % of stuck-at coverage.

5 Limitations and Threats to Validity

One of the main limitations of the current implementation of the hierarchical untestability identification tool is the fact that the RTL circuits considered are strictly divided into a control and datapath parts. Vast majority of real-world RTL designs are not restricted to the single control part concept.

Table 3 Untestability identification run-times

Circuit	gcd	*mult8x8*			*diffeq*	
Module	SUBTR	ADD2	ADD3	SUBTR2	MUX3	MUX4
Constraint extraction, s	2.90	47.86			9.18	
Constraint minimization, s	0.05	4710	< 0.01	52	14	82
Synthesis, s	5.38	5.33	9.52	5.25	5.10	5.10
ATPG, s	0.01	0.01	< 0.01	0.02	< 0.01	< 0.01

Table 4 Constraint-driven top-down ATPG versus BOTTOM-UP ATPG results for circuit modules

Circuit	gcd	*mult8x8*			*diffeq*	
Module	SUBTR	ADD2	ADD3	SUBTR2	MUX3	MUX4
W_q, %	100	100	100	100	100	100
W_a, %	95.74	86.64	55.88	85.33	75.00	75.00
W_r, %	85.11	72.49	47.06	74.07	64.71	64.71

However, this limitation is related to the path activation engine applied [22] and it is not a principal one for the presented method. For example the steps of minimization of constraints and the constraint-driven gate-level ATPG are completely independent of this restriction. Furthermore, it is possible to extend [22] into an RTL path activation tool that would support a network of control part FSMs as opposed to a single one.

Another limitation is the requirement that the modules selected for untestability analysis from the RTL design must be combinational. The method could be easily extended to support pipelined modules. In addition, there exists an efficient top-down method for proving untestable faults in register modules based on bounded model-checking [17]. However, the method cannot be currently applied to arbitrary sequential modules.

Finally, the complexity of the DNF of the constraints in the minimization step of the method grows exponentially with the increase of the cycle-limit k of the path activation. Table 1 shows the dependency between the number of leaves in the constraints DNF as a function of the cycle-limit k. Three modules have been included to the analysis: a subtraction function from the *gcd* benchmark and two additional modules from the *b04* circuit from the ITC99 benchmark family [9]. Figure 9 visualizes that dependency on a logarithmic scale. The benchmarks *gcd* and *b04* were selected to analyze the complexity because the curve cannot be explored on *diffeq* and *mult8x8* examples tested in the experimental results section. This is due to the fact that for both the DNF is empty until a certain cycle limit is reached.

6 Experimental Results

In order to evaluate the hierarchical untestability identification and test generation method, experiments on HLSynth92 [8] and HLSynth95 [18] benchmarks were run. In addition, to compare the solution with the traditional bottom-up approach (e.g. [23]) and assess its fault efficiency, a comparative study was carried out.

Table 2 presents the characteristics of the example circuits used in test pattern generation experiments in this paper. The following benchmarks were included to the test experiment: a Greatest Common Divisor (*gcd*), an 8-bit sequential multiplier (*mult8x8*), and a Differential Equation (*diffeq*). In the Table, the number of single stuck-at faults, the number of primary input and primary output bits, and the number of registers F_R, multiplexers F_M and functional units F_U in the RTL code are reported, respectively. The final column presents the upper limit for control part FSM cycles (i.e. the maximum times the same control state is traversed) as a time-step bound for the untestability proof. This bound is dependent on the design functionality and can be set by the test engineer.

Table 3 shows experiments reporting the time spent by different stages of the constraint-driven untestability identification flow developed in this paper. As explained in the Introduction, not all the modules (multiplexers F_M and functional units F_U) in the RTL designs are affected by sequential untestability. Our method identified one module from *gcd*, three modules from *mult8x8* and two modules from *diffeq* that had testability problems. Thus, only the above-mentioned six modules were considered in the hierarchical untestability proof by the constraint-driven logic-level ATPG.

As it can be seen from the Table, the extraction of test path constraints required up to 1 min of run time. As discussed in Section 5 the constraint minimization step is very much dependent on the time-step bound. In the case of ADD2 the time-step bound k is 7 and the time for minimizing the constraints is accordingly more than 4,000 s. The test environment synthesis

Table 5 Distribution of faults

	gcd	*mult8x8*	*diffeq*
# total faults	472	2356	10326
# tested faults [22]	439	1737	9867
# unobs./uncontr. faults	28	195	252
# untestable register faults [17]	0	130	130
# sequentially untestable faults	4	156	68
# remaining faults	1	138	9

Table 6 Comparison of fault efficiency

Circuit	Fault efficiency, %	
	Commercial ATPG	Constraint-based + register untestability [17]
gcd	76.55 %	99.79 %
mult8×8	89.06 %	89.90 %
diffeq	97.25 %	99.91 %

from VHDL to logic-level using Synopsys Design Compiler remained almost constant and was around 5 to 10 s per module while the deterministic constraint-based ATPG spent less than 0.02 s per module under test.

Constraint extraction was performed on a 2.5 GHz, Intel Core2 Duo T9300 PC with 4 GB of RAM, constraint minimization on a 2 GHz, Intel Core 2 Duo P7350, 3 GB RAM on Windows 7 Pro OS and the synthesis and test experiments were carried out on a Sun-Fire-V250 station with 1.28 GHz sparcv9 processor on Solaris 2.9 OS.

The experiments in Table 4 present comparison of the proposed method to the bottom-up paradigm [23]. For creating the test library for the bottom-up approach, the modules were first tested by the ATPG in a stand-alone mode. As a result a test sequence T_q yielding 100 % stuck-at fault coverage W_q was obtained. The proposed top-down constraint-driven ATPG reached fault coverage W_a which was less than W_q because of the constraints when accessing the module under test that was embedded into the network. However, the fault efficiency of the proposed approach was always 100 % for all the modules.

When test T_q was substituted to the test environment in a bottom-up manner then fault coverage W_r was reached, which was always lower than W_a because some of the tests were invalidated by sequential dependencies. In fact, W_r was considerably lower (by 8–14 %) for all the four modules analyzed. Thus, the proposed top-down method was capable of reaching maximum fault coverage for the analyzed modules with respect to the test path constraints and proving all of the sequentially untestable faults in them.

Table 5 presents detailed statistics of the circuits analyzed. The Table lists the total number of stuck-at faults in the whole circuit, the number of tested faults, number of unobservable/uncontrollable faults, untestable register faults from [17], the number of faults proven sequentially untestable by the proposed constraint-based approach and finally the number of all the remaining faults. The experiments show the efficiency of the constraint-driven engine in untestability identification. Though the method quickly classifies untestable faults caused by sequential untestability in the considered modules with 100 % efficiency, there remains a number of faults in other modules, including in the control part, which are still neither tested nor proven untestable. Some of these remaining faults can be tested or proven untestable by ATPG approaches at the logic-level.

In order to evaluate the fault efficiency (i.e. the ratio of the number of tested faults to the number of testable faults) of the proposed approach we compared it to a commercial ATPG from a major CAD vendor. The commercial ATPG is based on a deterministic gate-level algorithm. The results of the experiments are shown in Table 6. As it can be seen, the gate-level tool obtained comparable fault efficiency only in the case of the *mult8x8* example. In the case of *gcd* and *diffeq* benchmarks there was a large percentage of faults aborted by the tool.

7 Conclusion

The paper presents a method for hierarchical untestable stuck-at fault analysis of non-scan sequential circuits. The method is based on extracting and minimizing RTL test path activation constraints that drive a dedicated logic-level deterministic ATPG. In this paper we propose a formal method for minimizing test path constraints and evaluate it on sequential benchmarks Experiments show that it is capable of generating tests yielding maximum fault efficiency for modules embedded into the RTL. The fault efficiency achieved by the method is higher than the one obtained by a commercial sequential ATPG.

In addition, our study shows that traditional test generation at RTL based on symbolic test environment generation is too optimistic due to the fact that constraints in accessing the modules under test have been ignored. Experiments presented in this paper showed that bottom-up strategies caused a decrease of stuck-at fault coverage up to the range of 8–14 % in the modules tested when compared to the proposed approach. This short-coming is overcome by the proposed top-down constraint-based method which obtains 100 % stuck-at fault efficiency with respect to the sequential testability constraints for all the modules considered.

Acknowledgments The work has been supported by European Commission Framework Program 7 project FP7-ICT-2009-4-248613 DIAMOND, by Research Centre CEBE funded by European Union through the European Structural Funds and by Estonian Science Foundation grants 9429 and 8478.

References

1. Agrawal VD, Chakradhar ST (1995) Combinational ATPG theorems for identifying untestable faults in sequential circuits. IEEE Trans Comput Aided Des 14(9):1155–1160
2. Brayton RK, Hachtel GD, McMullen CT, Sangiovanni-Vincentelli AL (1984) Logic Minimization Algorithms for VLSI Synthesis. Kluwer Academic Publishers, Boston
3. Chayakul V, Gajski DD, Ramachandran L (1993) High-Level Transformations for Minimizing Syntactic Variances, DAC (Proceedings of the Design Automation Conference), Dallas, Texas, USA, p 413–418
4. De Moura L, Bjørner N (2008) Z3: An Efficient SMT Solver. TACAS (International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)), Budapest, Hungary, p 337–340
5. Dillig I, Dillig T, Aiken A (2010) Small Formulas for Large Programs, Proc. of the 17th intl. conf. on Static Analysis, Springer-Verlag, Berlin, Heidelberg, p 236–252
6. Fujiwara H, Ooi CY, Shimizu Y (2008) Enhancement of Test Environment Generation for Assignment Decision Diagrams, 9th IEEE Workshop on RTL and High Level Testing, Nov. IEEE, Sapporo, Japan, 45–50
7. Ghosh I, Fujita M (2000) Automatic test pattern generation for functional RTL circuits using assignment decision diagrams, Proc. DAC

(Proceedings of the Design Automation Conference), Los Angeles, California, USA, p 43–48
8. HLSynth92 benchmarks. http://ftp.ics.uci.edu/pub/hlsynth/HLSynth92
9. ITC benchmarks. http://www.cerc.utexas.edu/itc99-benchmarks/bench.html
10. Iyer MA, Long DE, Abramovici M (1996) Identifying sequential redundancies without search. In: Proc. 33rd Annu. Conf. DAC, LasVegas, pp 457–462
11. Jervan G et al (2002) High-Level and Hierarchical Test Sequence Generation. IEEE HLDVT, Cannes, 169–174
12. Lee J, Patel JH (1994 Oct) Architectural level test generation for microprocessors, IEEE Trans. CAD (IEEE Transactions on CAD of Integrated Circuits and Systems), Piscataway, New Jersey, USA, p 1288–1300
13. Liang H-C, Lee CL, Chen EJ (1995) Identifying untestable faults in sequential circuits. IEEE Des Test Comput 12(3):14–23
14. Long DE, Iyer MA, Abramovici M (2000) FILL and FUNI: Algorithms to identify illegal states and sequentially untestable faults. ACM Trans Des Autom Electron Syst 5(3):631–657
15. Murray BT, Hayes JP (1988) Hierarchical test generation using precomputed tests for modules, Proc. ITC (Proceedings of the International Test Conference), Washington, D.C., USA, p 221–229
16. Peng Q, Abramovici M, Savir J (2000) MUST: multiple stem analysis for identifying sequential untestable faults. In: Proc. Int. Test Conf. IEEE, Atlantic City, NJ, USA, p 839–846
17. Raik J, Fujiwara H, Ubar R, Krivenko A (2008) Untestable fault identification in sequential circuits using model-checking. ATS (Proceedings of the Asian Test Symposium), Sapporo, Japan, p 667–672
18. Raik J, Rannaste A, Jenihhin M, Viilukas T, Ubar R, Fujiwara H (2011) Constraint-Based Hierarchical Untestability Identification for Synchronous Sequential Circuits, Proc. of the European Test Symposium, IEEE Computer Society, Trondheim, Norway, p 147–152
19. Raik J, Ubar R (1999) Sequential Circuit Test Generation Using Decision Diagram Models, Proceedings of the DATE Conference, IEEE Computer Society, Munich, Germany, p 736–740
20. The ECLiPSe Constraint Programming System http://eclipseclp.org/
21. Vedula V, Abraham J (2002) FACTOR: A Hierarchical Methodology for Functional Test Generation and Testability Analysis, DATE Conf., IEEE Computer Society, Paris, France, p 730–734
22. Viilukas T, Raik J, Jenihhin M, Ubar R, Krivenko A (2010) Constraint-based test pattern generation at the register-transfer level, 13th IEEE DDECS Symposium, IEEE Computer Society, Vienna, Austria, p 352–357
23. Zhang L, Ghosh I, Hsiao M (2003) Efficient Sequential ATPG for Functional RTL Circuits, Int. Test Conf., IEEE, Charlotte, NC, USA, p 290–298

Taavi Viilukas received his M.Sc. degree from Tallinn University of Technology (TUT) in 2006. Currently he is a Ph.D. student at TUT. His research interest is hierarchical test pattern generation. He has co-authored 7 papers.

Anton Karputkin received his M.Sc. degree from University of Tartu in 2008. Currently he is a Ph.D. student at Tallinn University of Technology. His research interest is formal methods for test and verification. He has co-authored 4 papers.

Jaan Raik received his M.Sc. and Ph.D. degrees in Computer Engineering from Tallinn University of Technology in 1997 and in 2001, respectively, where he currently holds the position of a postdoc researcher. He is a member of IEEE Computer Society, a member of program committees for several top-level conferences and has co-authored more than 100 scientific publications. In 2004, he was awarded the national Young Scientist Award. Starting from 2010 he acts as the scientific co-ordinator of the EU FP7 DIAMOND research project. His main research interests include high-level test generation and verification.

Maksim Jenihhin received his M.Sc. and PhD degrees in Computer Engineering from Tallinn University of Technology (TUT) in 2004 and in 2008, respectively. Currently he is a senior research fellow at TUT. He has co-authored more than 50 papers.

Raimund Ubar received his Ph.D. degree in 1971 at the Bauman Technical University in Moscow. He is a professor of Computer Engineering at Tallinn University of Technology. His research interests include computer science, electronics design, design verification, test generation, fault simulation, design-for-testability, fault-tolerance. He has published more than 200 papers and two books. R. Ubar has given seminars or lectures in 20–25 universities in more than 10 countries. In 1993–1996 he was the Chairman of the Estonian Science Foundation and a member of the Estonian Science Council. He is a Golden Core Member of the IEEE, a member of ACM, SIGDA, Gesellschaft der Informatik (Information Society, Germany), European Test Technology Technical Committee and Estonian Academy of Sciences.

Hideo Fujiwara received the B.E., M.E., and Ph.D. degrees in electronic engineering from Osaka University, Osaka, Japan, in 1969, 1971, and 1974, respectively. He was with Osaka University from 1974 to 1985 and Meiji University from 1985 to 1993, Nara Institute of Science and Technology (NAIST) from 1993 to 2011, and joined Osaka Gakuin University in 2011. Presently he is Professor Emeritus of NAIST and a Professor at the Faculty of Informatics, Osaka Gakuin University, Osaka, Japan.

His research interests are logic design, digital systems design and test, VLSI CAD and fault tolerant computing, including high-level/logic synthesis for testability, test synthesis, design for testability, built-in self-test, test pattern generation, parallel processing, and computational complexity. He has published over 400 papers in refereed journals and conferences, and nine books including the book from the MIT Press (1985) entitled "Logic Testing and Design for Testability." He received the IECE Young Engineer Award in 1977, IEEE Computer Society Certificate of Appreciation Awards in 1991, 2000 and 2001, Okawa Prize for Publication in 1994, IEEE Computer Society Meritorious Service Awards in 1996 and 2005, IEEE Computer Society Continuing Service Award in 2005, and IEEE Computer Society Outstanding Contribution Award in 2001 and 2009.

He served as an Editor of the IEEE Trans. on Computers (1998–2002), Journal of Electronic Testing: Theory and Application (1989–2004), Journal of Circuits, Systems and Computers (1989–2004), VLSI Design: An Application Journal of Custom-Chip Design, Simulation, and Testing (1992–2005), and several guest editors of special issues of IEICE Transactions of Information and Systems. Dr. Fujiwara is a life fellow of the IEEE, a Golden Core member of the IEEE Computer Society, a fellow of the IEICE (the Institute of Electronics, Information and Communication Engineers of Japan) and a fellow of the IPSJ (the Information Processing Society of Japan).