IEICE TRANS. FUNDAMENTALS, VOL.E87–A, NO.12 DECEMBER 2004

PAPER Special Section on VLSI Design and CAD Algorithms

A Design Scheme for Delay Testing of Controllers Using State Transition Information

Tsuyoshi IWAGAKI[†a)], ***Student Member***, **Satoshi OHTAKE**[†], ***Member***, ***and*** **Hideo FUJIWARA**[†], ***Fellow***

SUMMARY This paper presents a non-scan design scheme to enhance delay fault testability of controllers. In this scheme, we utilize a given state transition graph (STG) to test delay faults in its synthesized controller. The original behavior of the STG is used during test application. For faults that cannot be detected by using the original behavior, we design an extra logic, called an invalid test state and transition generator, to make those faults detectable. Our scheme allows achieving short test application time and at-speed testing. We show the effectiveness of our method by experiments.
key words: *controller, delay fault, non-scan design, invalid test state and transition generator, at-speed test*

1. Introduction

As the speed of modern VLSI circuits increases, delay fault testing is becoming essential to guarantee the timing correctness of the circuits. Delay test generation for such circuits is a challenging problem. This is because there exist many sequentially untestable delay faults in a circuit [2], and the task of identifying those faults is very time-consuming. It is virtually impossible to identify all the untestable faults in a large circuit. To facilitate delay test generation, standard scan methods [7], [8] and enhanced scan ones [2], [4] have been proposed. Given a sequential circuit, these design methods make most or all of the sequentially untestable faults detectable by making every flip-flop (FF) controllable and observable. As a result, the test generation time is significantly reduced and the fault coverage becomes higher. However, in scan-based delay testing, the test application time becomes longer because of the scan-shift operation. In addition, the scan-shift operation is generally performed at a low clock speed while the second vectors of two-pattern tests are launched at a rated clock speed. This situation may cause the IR-drop [9] because the operating speed rapidly changes, and it makes apparent circuit delay increase temporarily. In consequence, the test may detect temporary delay faults and it poses over-testing. Therefore, it is desirable that the operating speed is constant during test application.

Recently, design for testability (DFT) methods at register transfer (RT) level have been proposed [5]. In general, an RT-level circuit is composed of a controller, which is represented by a state transition graph (STG), and a data path, which is represented by hardware elements such as registers, multiplexers (MUXs) and operational modules. For delay faults, a non-scan DFT method of data paths, which overcomes the drawbacks of scan-based testing, has been proposed [1]. On the other hand, a DFT method for stuck-at faults in controllers has been proposed [6]. This method is also non-scan based, and achieves complete fault efficiency, short test application time and at-speed testing. In this method, the above merits are realized by utilizing a given STG and by appending an extra logic, called an *invalid test state generator (ISG)*, to the original controller.

This paper proposes a non-scan design scheme, which is an extension of one in [6], to enhance delay fault testability of controllers. In this scheme, we utilize a given STG to test delay faults in its synthesized controller. The original behavior of the STG is used during test application. For faults that cannot be detected by using the original behavior, we append an extra logic, called an *invalid test state and transition generator (ISTG)*, to the original controller. In this paper, we discuss a classification of untestable faults in a controller, and show our DFT flow based on the classification. Our scheme allows achieving short test application time and at-speed testing, which is always performed at a constant clock speed. By some experiments, we show the effectiveness of our method.

2. Preliminaries

2.1 Target Circuit and Fault Model

Our target circuit is a controller represented by an STG, and we target delay faults which can be tested by two-pattern tests (e.g., transition faults, path delay faults, etc.) in the circuit. In the following discussion, we focus on the transition fault model for simplicity. Figure 1 shows an example of a controller represented by an STG. In this paper, we assume that a gate-level implementation of a controller is given, and

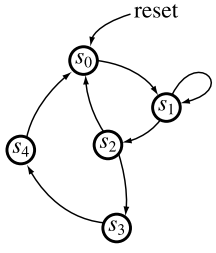

Fig. 1 State transition graph representing a controller.

Manuscript received March 18, 2004.
Manuscript revised June 18, 2004.
Final manuscript received August 5, 2004.
[†]The authors are with the Graduate School of Information Science, Nara Institute of Science and Technology (NAIST), Ikomashi, 630–0192 Japan.
a) E-mail: tsuyo-i@is.naist.jp

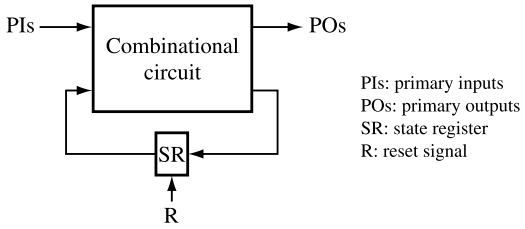

Fig. 2 Synthesized controller.

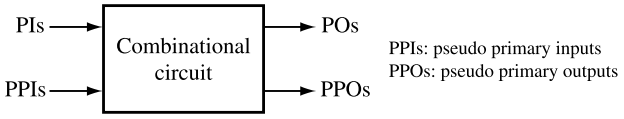

Fig. 3 Combinational test generation model (CTGM).

the controller has a reset signal, i.e., we can make a transition from any state to the reset state by activating the reset signal. Figure 2 represents a sequential circuit which can be obtained by synthesizing a given STG. We also assume that, for a given controller, the mapping information between each state in the STG and the value of the state register (SR) (state encoding information) is available.

2.2 Terminologies

Here, we define several terminologies. For any value of the SR in a sequential circuit synthesized from a given STG, the state corresponding to the value is called a *valid state* if it is reachable from the reset state in the STG. Otherwise, it is called an *invalid state*. For a synthesized controller, a combinational circuit extracted from the controller by replacing the SR with pseudo primary inputs (PPIs) and pseudo primary outputs (PPOs) is called a *combinational test generation model (CTGM)* (Fig. 3). Every two-pattern test for a CTGM, (V_1, V_2), can be denoted as $(I_1\&S_1, I_2\&S_2)$, where I_1 and I_2 are the values of primary inputs (PIs), S_1 and S_2 are the values of PPIs, and "&" is the concatenation operator. A two-pattern test for a CTGM, $(I_1\&S_1, I_2\&S_2)$, is said to be a *valid two-pattern test* if there exists an arc (transition) (I, P, N, O) in a given STG such that $I = I_1$, $P = S_1$ and $N = S_2$, where I is an input value, P is a present state value, N is a next state value, and O is an output value. Otherwise, it is called an *invalid two-pattern test*. The transition corresponding to a valid two-pattern test (resp. an invalid two-pattern test) is called a *valid test transition* (resp. an *invalid test transition*). Valid test transitions and invalid ones are collectively called *test transitions*. For each state included in a test transition, the state is called a *valid test state* (resp. an *invalid test state*) if it is a valid state (resp. an invalid state). Also, valid test states and invalid ones are collectively called *test states*.

Figure 4 shows an example of test states and test transitions. When two-pattern tests are generated for the CTGM (Fig. 3) of Fig. 1, the test transitions corresponding to the generated two-pattern tests can be classified into five types:

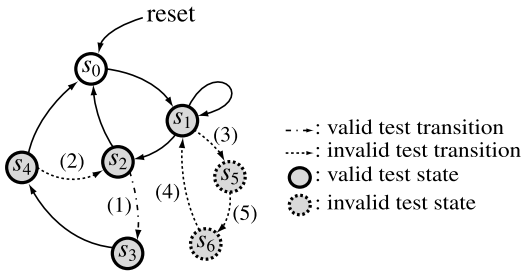

Fig. 4 Test states and transitions.

- valid test transition (Fig. 4(1)),
- invalid test transition from a valid test state to a valid test state (Fig. 4(2)),
- invalid test transition from a valid test state to an invalid test state (Fig. 4(3)),
- invalid test transition from an invalid test state to a valid test state (Fig. 4(4)) and
- invalid test transition from an invalid test state to an invalid test state (Fig. 4(5)).

3. Proposed Method

3.1 Test Architecture

In our testing scheme, the original behavior of a given STG is used during test application, i.e., valid two-pattern tests are applied by using the original behavior. Faults that cannot be detected by using the original behavior are tested by an extra logic, called an *invalid test state and transition generator (ISTG)*. Our test architecture is shown in Fig. 5, which is an extention of the test architecture [6]. In Fig. 5, the respective DFT elements play the following roles.

- The ISTG is used to generate invalid test states and invalid test transitions.
- The extra pin of t_{mode} is used to select between the normal mode and the test mode.
- The extra pins of t_{out} are used to observe the value of the SR. The bit width of t_{out} is the same as that of the SR.
- The extra pins of t_{sel} are used to distinguish among invalid two-pattern tests[†].
- The MUX is used to switch between the signal from the combinational part of the controller and that from the ISTG.

The differences between our test architecture and the previous one [6] are as follows. Unlike the ISG of the previous test architecture, the ISTG of our test architecture is used to generate not only invalid test states needed to apply the first vectors of invalid two-pattern tests but also invalid test transitions corresponding to the invalid two-pattern tests. Notice that the ISG has no t_{sel}. Furthermore, in our test architecture, t_{out} is appended to the output side of the SR while t_{out} of the

[†]The details will be described in Sect. 3.3.

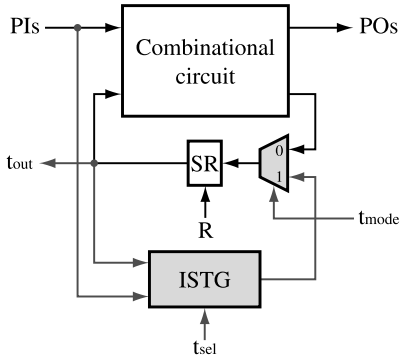

Fig. 5 Test architecture.

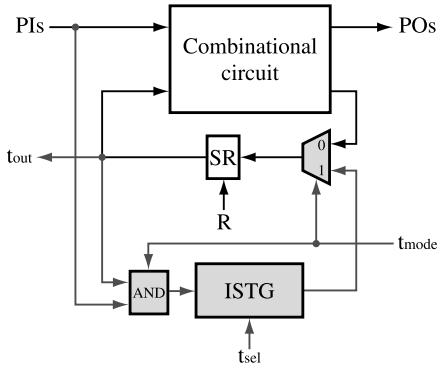

Fig. 6 Power-aware configuration.

previous test architecture is appended to the input side of the SR. In consequence, our test architecture can test delay faults appropriately because the value captured by the SR can be observed.

Our test architecture can achieve short test application time and at-speed testing because the scan-shift operation is never used. Note that we use the terminology of "at-speed test" only if test application can always be performed at a rated clock speed.

Here, we mention testing of delay faults in the ISTG. Since the ISTG is only used in the test mode, we do not need to care about its behavior in the normal mode. Therefore, to test delay faults in the ISTG, we need only to check whether the value captured by the SR is correct or not by using t_{out} in the test mode. It can be performed during testing of the controller simultaneously.

Let us consider an impact of power consumption on a controller, which is induced by the ISTG. Although the ISTG is only used during testing, it might consume power in the normal mode. If the impact is serious, we can avoid it by configuring the controller as Fig. 6. In Fig. 6, "AND" is used to suppress the power consumption in the ISTG during normal operation. If the value of t_{mode} is 1, "AND" supplies the values of the PIs and the SR to the ISTG. In the normal operation, the ISTG receives the constant value of zeros by setting the value of t_{mode} to 0. Note that, in the following discussion, we do not consider the power impact of an ISTG for simplicity.

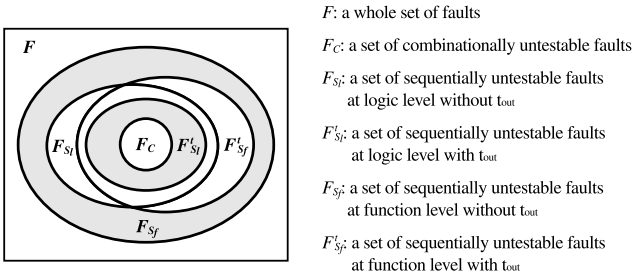

Fig. 7 Classification of untestable faults.

3.2 Test Quality

In a sequential circuit, untestable delay faults generally exist. Here, we classify untestable delay faults in a controller into five categories in terms of "logic level," "function level" and t_{out}. The classification is shown in Fig. 7. Let F be a whole set of faults. All the faults in a set of F_C are untestable in the combinational part of the controller if we consider the SR as POs and PIs, i.e., any two-pattern test can be applied to the combinational part and any response from that can be observed. These faults are called combinationally untestable faults. Some combinationally testable faults in $\overline{F_C}$ $(= F - F_C)$ are untestable because the value of the SR is restricted by the available state transitions in its synthesized controller. Such faults belong to a set of F_{S_l} $(\supset F_C)$. We call these faults sequentially untestable faults at logic level without t_{out}. When a given STG is synthesized, some new states and transitions are generally implemented in the synthesized controller. This implies that some testable faults in $\overline{F_{S_l}}$ are untestable if the original behavior of the given STG is only considered. We classify these faults into a set of F_{S_f} $(\supset F_{S_l})$. These faults are called sequentially untestable faults at function level without t_{out}. Let us consider appending t_{out} to the synthesized controller here. Appending t_{out} makes some untestable faults in F_{S_l} and F_{S_f} testable. Thus, F_{S_l} and F_{S_f} change into sets of $F^t_{S_l}$ $(\supset F_C)$ and $F^t_{S_f}$ $(\supset F^t_{S_l})$, respectively. Let M be a given STG, and M' be the STG corresponding to a sequential circuit derived by synthesizing M. In Fig. 7, some faults in F_{S_l} can be activated by using the behavior of M but the effects of the faults cannot be propagated to a PO by using the behavior of M'. Such faults can be detected by using the behavior of M if t_{out} is appended to the circuit. These faults belong to $F_{S_l} - F^t_{S_f}$.

In test generation for a given sequential circuit, a sequential test generator (ATPG) tries to identify all the untestable faults in F_{S_l} and to generate tests for all the testable faults in $\overline{F_{S_l}}$. The goal of this task can be achieved if the ATPG has enough time to complete the task. However, it is infeasible for a large circuit. For such a circuit, DFT approaches should usually be used to facilitate test generation. In our method, t_{out} is appended to a given circuit in order to facilitate test generation. As a result, some faults in F_{S_l} are made detectable. However, the number of these faults is not so large [2]. Our method aims to detect faults in

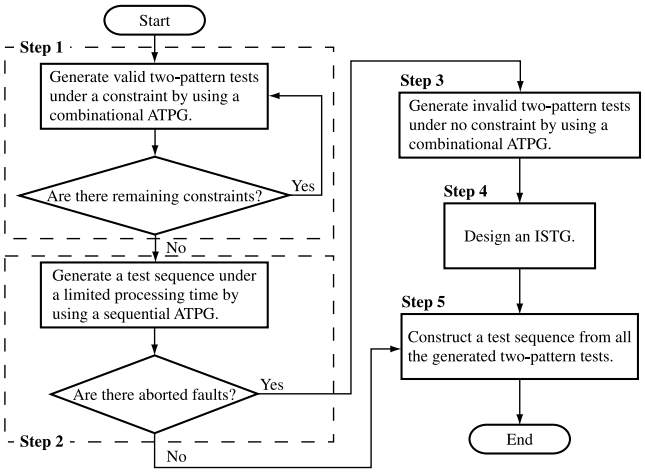

Fig. 8 Flow chart of our method.

$\overline{F^t_{S_l}}$ and identify faults in $F^t_{S_l}$ as much as possible in a reasonable time. Although a fault in $F_{S_f} - F^t_{S_l}$ itself does not affect the performance of a controller under the original behavior of its STG and under the single fault assumption, we try to detect the fault. The reason is as follows. Suppose that there exist an untestable fault f' in $F_{S_f} - F^t_{S_l}$ and a testable fault f in $\overline{F_{S_f}}$ simultaneously in a circuit, and the effect of f' cannot be propagated to a PO but f' can be activated during normal operation. In this case, if f' is not tested, we cannot evaluate whether a test generated for f is invalidated by f'. This implies that f can be missed if there are no tests that detect f in a generated test set. In order to avoid such a situation, we should test not only testable faults in $\overline{F_{S_f}}$ but also untestable faults in $F_{S_f} - F^t_{S_l}$. Under the single fault assumption, the above discussion dose not make sense. However, from a practical point of view, it is very useful because there can exist multiple faults in a circuit.

3.3 Flow of Our Method

Given a controller, the procedure of our method is performed as Fig. 8. In the following paragraphs, we explain each step of Fig. 8 in detail.

Step 1: For the CTGM of a given controller, we use a combinational ATPG. In order to generate valid two-pattern tests, we give some information (constraint) to a combinational ATPG. A constraint is defined as a vector pair $(I^C_1 \& S^C_1,\ I^C_2 \& S^C_2)$. Each bit of a constraint can take the value of 0, 1 or *don't care* (X). When we give a constraint to a combinational ATPG, the ATPG tries to generate two-pattern tests under the constraint, i.e., for every X of the constraint, a suitable value is specified.

A constraint C is derived as follows. First, a transition $T = (I, P, N, O)$ is selected from a given STG. Then, T is used as $C = (I\&P, \text{“}X\text{s”}\&N)$. It is obvious that two-pattern tests generated under this constraint can always be applied by using the original behavior of the controller. Thus, we can obtain valid two-pattern tests. Note that, because of the presence of a delay fault f, we may fail to justify a valid two-pattern test generated for f by using the original behavior. However, it does not matter because any error induced by f in the SR can always be observed from t_{out}. In Step 1, if we use all the constraints corresponding to the transitions in the STG, we can identify all the untestable faults in $F^t_{S_f}$, and valid two-pattern tests can be generated for all the faults in $\overline{F^t_{S_f}}$. The detected faults are dropped from the fault list.

Step 2: For the remaining faults in Step 1, we try to identify untestable faults in $F^t_{S_l}$ and generate a test sequence for faults in $F^t_{S_f} - F^t_{S_l}$ by applying a sequential ATPG to the controller with t_{out}. Since the circuit has t_{out}, test generation for it is easier than that for the original one. Moreover, the number of target faults in this step is much reduced compared with the total number of faults. Nevertheless, this task is very time-consuming. Therefore, we use a sequential ATPG under a limited processing time (or a limited number of backtracks) per fault. This implies that, in this step, we take into account faults that can be easily identified as untestable faults and easily detectable faults. The detected faults and the untestable faults are dropped from the fault list. Notice that if there are no aborted faults in this step, we do not need to perform Steps 3 and 4. This means that only the pins of t_{out} are added to the original controller as a DFT element.

Step 3: We generate two-pattern tests, which are invalid, for the remaining faults in Step 2 under no constraint by using a combinational ATPG. This step can identify all the untestable faults in F_C.

Step 4: We design an ISTG to test the faults detected in Step 3 because we do not identify whether these faults belong to $F^t_{S_l} - F_C$ or not. An ISTG must realize functions to apply invalid two-pattern tests to the combinational part of the controller. Furthermore, it must also have functions that make all of the invalid test states in the invalid two-pattern tests reachable from the reset state. For example, given n invalid two-pattern tests $t_1 = (I^1_1 \& S^1_1, I^1_2 \& S^1_2)$, $t_2 = (I^2_1 \& S^2_1, I^2_2 \& S^2_2)$, ..., $t_n = (I^n_1 \& S^n_1, I^n_2 \& S^n_2)$, an ISTG must realize the functions shown in the truth table (Table 1). Note that, in Table 1, if there exist m invalid two-pattern tests such that $I^1_1 \& S^1_1 = I^2_1 \& S^2_1 = \cdots = I^m_1 \& S^m_1$ and $S^i_2 \neq S^j_2$ ($\forall i, j, 1 \leq i, j \leq m, i \neq j$), we need t_{sel} to distinguish among them. The bit width of t_{sel} is $\lceil \log m_{\max} \rceil$, where $m_{\max}$ is the maximum number of two-pattern tests that satisfy the above conditions.

We touch on a problem to reduce the area of an ISTG here. If the truth table shown in Table 1 includes X values, i.e., X values are included in two-pattern tests, we can make use of them to reduce the area of an ISTG. This problem is considered as a type of an input encoding problem [10]. Therefore, we could apply some heuristics [10] for the input encoding problem to design an ISTG. However, this is still an open problem. In this paper, it is assumed that two-pattern tests generated in Step 3 do not include X values.

Step 5: In order to construct a test sequence for the original circuit, we determine an order of applying all the gener-

Table 1 Truth table of an ISTG.

Inputs	Outputs
$I_1^1 \& S_1^1$	S_2^1
$I_1^2 \& S_1^2$	S_2^2
⋮	⋮
$I_1^n \& S_1^n$	S_2^n

Table 2 Distance matrix.

	R	t_1	t_2	t_3	t_4	t_5
R	—	2	4	1	3	2
t_1	1	—	0	2	4	3
t_2	1	−1	—	2	4	3
t_3	1	3	5	—	4	3
t_4	1	1	3	0	—	1
t_5	1	3	5	2	4	—

ated two-pattern tests. Note that the test sequence generated in Step 2 is applied to the circuit before or after applying the test sequence obtained in this step. Here, we consider a problem to construct the test sequence that has the minimum length. It is solved as an asymmetric traveling salesperson problem (ATSP) on a complete weighted directed graph represented by a distance matrix, where a vertex t corresponds to a two-pattern test, an arc (t_i, t_j) corresponds to the path between t_i and t_j, and the weight of the arc corresponds to the distance from t_i to t_j. The distance $d(t_i, t_j)$ means the minimum clock cycles that are needed to apply the first vector of t_j after applying t_i. Note that if t_j is a valid two-patten test and the values of the second vector of t_i and the first vector of t_j are identical, the value of $d(t_i, t_j)$ is -1. Thus, we can construct a test sequence by solving the corresponding ATSP.

For example, let us consider the ATSP for Fig. 4. In Fig. 4, there are five test transitions. These test transitions (1), (2), (3), (4) and (5) correspond to two-pattern tests t_1, t_2, t_3, t_4 and t_5, respectively. Let the reaching states after applying t_1, t_2, t_3, t_4 and t_5 be S_4, S_3, S_U, S_1 and S_U respectively, where S_U denotes an unknown state. Table 2 shows the distance matrix for the five two-pattern tests. Note that, in this table, $d(R, t)$ (resp. $d(t, R)$) denotes the minimum distance from the reset state R (resp. S_a) to S_b (resp. R), where S_a is the reaching state after applying t, and S_b is the state in applying the first vector of t. A solution of this problem is $R \to t_1 \to t_2 \to t_4 \to t_3 \to t_5 \to R$. From this solution, t_1, t_2, t_4, t_3 and t_5 are applied in that order.

4. Advantages of Our Method

4.1 Conventional Methods and Our Method

In this subsection, we summarize the proposed method and conventional methods (standard scan and enhanced scan ones). In the following discussion, we assume that MUXs are used in the scan-based methods although there are some ways to implement a standard scan FF (SSFF) and an enhanced scan FF (ESFF).

Standard scan method: Test generation for a controller designed by this method requires a combinational ATPG which supports the skewed-load [7] mode and/or the broad-side [8] one. Generated two-pattern tests are applied to the controller through a scan chain in the skewed-load fashion and/or the broad-side one. The test application time is estimated as $n_{\mathrm{TPT}}(n_{\mathrm{SSFF}} + 2) + n_{\mathrm{SSFF}}$, where n_{TPT} and n_{SSFF} are the number of two-pattern tests and SSFFs, respectively. In this method, each SSFF in the controller has an additional MUX. Therefore, the area overhead is $A_{\mathrm{MUX}} \times n_{\mathrm{SSFF}}$, where A_{MUX} is the area of the additional MUX. As a result, the delay of an MUX are added as the additional circuit delay. This method needs three additional pins. Note that we assume that this method has a single scan chain for simplicity.

Enhanced scan method: We can generate tests for a controller designed by this method by using a combinational ATPG. The test application time is estimated as $2n_{\mathrm{TPT}}(n_{\mathrm{ESFF}} + 1) + 2n_{\mathrm{ESFF}}$, where n_{ESFF} is the number of ESFFs. Each ESFF in the controller has an additional MUX and a hold latch (HL) [4]. The area overhead is, therefore, $(A_{\mathrm{MUX}} + A_{\mathrm{HL}}) \times n_{\mathrm{ESFF}}$. Note that, although the area overhead can be reduced by using some techniques (e.g., [3]), we estimate it as the above equation for simplicity. The delay penalty is the same as that of the standard scan method because HLs themselves are not connected to the combinational part of the controller. Also, the pin overhead of this method is the same as that of the standard scan method because HLs can be controlled by the scan clock. Consequently, the total number of additional pins is 3. Note that it is also assumed that this method has a single scan chain.

Our method: In our method, we first generate tests for the combinational test generation model of a given controller by using a combinational ATPG under the constraints extracted from its STG. The test generation is repeated n_c times, where n_c is the number of constraints. Next, we generate a test sequence for the remaining faults under a limited processing time by using a sequential ATPG. Then, we try to generate two-pattern tests for the aborted faults in the previous step under no constraint. The test application time is determined by an order of applying all the generated two-pattern tests to the controller. The area overhead is $A_{\mathrm{MUX}} \times n_{\mathrm{FF}} + A_{\mathrm{ISTG}}$, where n_{FF} is the number of FFs, and A_{ISTG} is the area of an ISTG. The proposed method has the same delay penalty compared to that of the scan-based methods because ISTGs are not used during normal operation. However, in order to perform at-speed testing, we need to pay attention to the maximum delay of an ISTG. The maximum delay of an ISTG depends on its structure. In the next subsection, we evaluate the maximum delays of ISTGs by experiments. We believe that the ISTG of a given controller can be constructed with small maximum delay compared to that of the original circuit by contriving ways to synthesize the ISTG. The extra pins (t_{sel}, t_{out} and t_{mode}) are needed in our method. The sum of the bit width of these pins is $|t_{\mathrm{sel}}| + |t_{\mathrm{out}}| + 1$. Notice that, in the proposed method, if Steps 3 and 4 are not performed, the pin overhead is $|t_{\mathrm{out}}|$. In a controller-data path circuit, if we consider that its controller part is tested independently of its data path

part, the PIs and the POs of the data path can be used as t_{sel} and t_{out} during testing of the controller, respectively. The PIs of the data path are split and connected to t_{sel}, and the POs of the data path are shared with t_{out} by using MUXs. Let n_{DPI} and n_{DPO} be the bit widths of the PIs and the POs of the data path, respectively. In the sharing of test pins, if $n_{DPI} \geq |t_{sel}|$, no additional test pins are required for t_{sel}. Otherwise, the number of additional test pins is $|t_{sel}| - n_{DPI}$. For t_{out}, if $n_{DPO} \geq |t_{out}|$, we need one additional test pin to control MUXs. Otherwise, we need to apply one two-pattern test $\lceil |t_{out}|/n_{DPO} \rceil$ times and observe the value of t_{out} in $\lceil |t_{out}|/n_{DPO} \rceil$ batches. In this case, the number of additional test pins is $\lceil |t_{out}|/n_{DPO} \rceil$. As a result of the sharing, although the area overhead increases to $A_{MUX} \times (n_{FF} + |t_{out}|) + A_{ISTG}$, the pin overhead decreases to $(|t_{sel}| - n_{DPI}) + \lceil |t_{out}|/n_{DPO} \rceil + 1$ in the worst case. Since it is expected that $n_{DPI} \geq |t_{sel}|$ and $n_{DPO} \geq |t_{out}|$ can be satisfied for practical RT-level circuits, the pin overhead decreases to 2. It is also reduced to 1 if Steps 3 and 4 are skipped.

We mention here some differences among the three methods. In the scan-based methods, every FF is just replaced with an SSFF or an ESFF injudiciously, while ISTGs of our method are designed depending on generated invalid two-pattern tests. These features could cause the following. Suppose that additional two-pattern tests are required for some reasons (e.g., for fault diagnosis) after applying the respective methods to a circuit. In the enhanced scan method, these two-pattern tests can always be applied without modification of the circuit. However, the other methods could not cope with such a situation. The advantages of our method are as follows. Since the scan-shift operation is needed in the scan-based methods, at-speed test cannot be performed, i.e., a slow clock is used except in activating delay faults. However, our method can always apply tests at a rated clock speed. In this environment, the IR-drop will be suppressed. Moreover, our method can be performed flexibly according to a trade-off between hardware overhead and test generation time. The trade-off is determined by the number of constraints used in Step 1 of the proposed method and by the limited processing time per fault in Step 2. In the scan-based methods, all the FFs in a circuit are modified independently of the circuit function. Consequently, many untestable delay faults, which do not need to be tested, are made detectable. This implies that yield loss may potentially occur. In our method, over-testing is also caused by t_{out} and the ISTG. However, owing to Steps 1 and 2 of the proposed procedure, our method can alleviate over-testing compared with the scan-based methods.

4.2 Experimental Results

To evaluate our method, the following experiments were performed on a Sun Blade 2000 workstation. We used the MCNC '91 benchmark circuits shown in Table 3. A reset signal was appended to every benchmark circuit. Columns "#PIs," "#POs," "#FFs," "#States" and "#Arcs" denote the number of PIs, POs, FFs, states in an STG, and transitions in an STG, respectively. Column "Area" represents circuit size. The size was estimated by Design Compiler (Synopsys), and the value of "Area" was calculated by considering the area of a 2-input NAND gate to be 2. During logic synthesis, binary encodings were used. Note that, it is assumed that each benchmark circuit has a data path, which is controlled by the benchmark circuit, and the data path has enough PIs and POs to be shared with additional test pins. In the following experiments, we compared our method (NS) to the standard scan technique (SS) and the enhanced scan technique (ES). TestGen (Synopsys) and FlexTest (Mentor Graphics) were used as a combinational ATPG and a sequential one respectively, and the transition fault model was targeted. Note that, in SS and ES, we assumed that the both methods have a single scan chain. For SS, we compared only the hardware overhead because the ATPGs do not support the skewed-load mode and the broad-side one.

Table 3 Circuit characteristics.

Circuit name	#PIs	#POs	#FFs	#States	#Arcs	Area
bbsse	7	7	4	16	72	295
keyb	7	2	5	19	189	459
kirkman	12	6	4	16	446	360
planet	7	19	6	48	163	937
s298	3	6	8	218	1,314	3,662
s420	19	2	5	18	155	122
sand	11	9	5	32	216	866
scf	27	56	7	121	407	1,378

First, we show the test generation results. In this experiment, our method was performed as follows. Column "#Arcs" in Table 3 corresponds to the number of constraints in Step 1 of our method. We used all the constraints in Step 1, i.e., for each circuit, test generation was repeated #Arcs times. In Step 2, the backtrack limit was set to 64, which is not so large value. In Step 5, we used a simple algorithm to solve the ATSP and the processing time was negligibly short. Table 4 shows the test generation results of the respective methods. Columns "#All," "#Det" and "#Unt" give the number of total faults, detected faults and untestable faults, respectively. Columns "#TPT" and "#Vec" list the number of two-pattern tests and the length of the test sequence generated in Step 2. Columns "TGT [s]," "FC [%]" (= (#Det/#All) × 100) and "TAT [CC (clock cycles)]" denote test generation time, fault coverage and test application time, respectively. For reference, in Table 5, we list the test generation results of our method in more detail. Columns "#Tgt" and "#Abt" give the number of target faults and aborted faults in each step. Column "FC [%]" represents the cumulative results of fault coverage. In ES, there was no aborted fault during test generation, i.e., 100% fault efficiency was achieved in all the cases. Our method encountered no aborted fault in Step 3 for all the circuits. This implies that our method also achieved 100% fault efficiency. In Table 4, the value in each parenthesis represents the result in the case of removing untestable faults identified in Step 2 from the fault list of ES in advance. This can evaluate

Table 4 Test generation results.

Circuit name	#All	#Det		#Unt		#TPT		#Vec	TGT [s]		FC [%]		TAT [CC]	
		ES	NS	ES	NS	ES	NS		ES	NS	ES	NS	ES	NS
bbsse	782	782 (779)	760	0 (3)	22	57 (55)	82	95	0.2 (0.4)	6.5	100.00 (99.62)	97.44	578 (558)	269
keyb	1,196	1,196 (1,194)	1,184	0 (2)	12	110 (110)	170	279	0.5 (0.7)	33.1	100.00 (99.83)	99.00	1,330 (1,330)	641
kirkman	944	944 (944)	937	0 (0)	7	86 (90)	144	174	0.4 (0.6)	16.6	100.00 (100.00)	99.26	868 (908)	514
planet	2,580	2,579 (2,578)	2,553	1 (2)	27	122 (115)	169	191	1.6 (1.9)	60.1	99.96 (99.92)	98.95	1,720 (1,622)	542
s298	10,260	10,259 (10,259)	10,256	1 (1)	4	561 (564)	1,653	858	16.6 (17.5)	1,219.6	99.99 (99.99)	99.96	10,114 (10,168)	4,069
s420	254	232 (232)	216	22 (22)	38	27 (24)	30	82	0.1 (0.1)	5.8	91.34 (91.34)	85.04	334 (298)	156
sand	2,408	2,408 (2,405)	2,388	0 (3)	20	146 (142)	286	103	2.0 (1.1)	33.1	100.00 (99.88)	99.17	1,762 (1,714)	695
scf	3,850	3,844 (3,838)	3,784	6 (12)	66	188 (175)	331	628	2.7 (3.6)	209.2	99.84 (99.69)	98.29	3,022 (2,814)	1,300

Table 5 Detail of each step.

Circuit name		#Tgt	#Det	#Unt	#Abt	#TPT or #Vec	TGT [s]	FC [%]
bbsse	Step 1	782	612	170	0	80	4.2	78.26
	Step 2	170	132	32	6	95	2.3	95.14
	Step 3	6	6	0	0	2	0.0	97.44
keyb	Step 1	1,196	905	291	0	170	21.7	75.67
	Step 2	291	279	12	0	279	11.4	99.00
	Step 3	—	—	—	—	—	—	—
kirkman	Step 1	944	889	55	0	144	14.8	94.17
	Step 2	55	48	7	0	174	1.8	99.26
	Step 3	—	—	—	—	—	—	—
planet	Step 1	2,580	2,206	374	0	150	32.4	85.50
	Step 2	374	264	48	62	191	27.6	95.74
	Step 3	62	62	0	0	19	0.0	98.95
s298	Step 1	10,260	9,398	862	0	1,541	680.3	91.60
	Step 2	862	513	9	340	858	538.7	96.60
	Step 3	340	339	1	0	112	0.6	99.96
s420	Step 1	254	160	94	0	30	5.0	62.99
	Step 2	94	56	38	0	82	0.8	85.04
	Step 3	—	—	—	—	—	—	—
sand	Step 1	2,408	2,322	86	0	284	30.5	96.43
	Step 2	86	64	20	2	103	2.6	99.09
	Step 3	2	2	0	0	2	0.0	99.17
scf	Step 1	3,850	3,438	412	0	323	147.4	89.30
	Step 2	412	307	90	15	628	61.8	97.27
	Step 3	15	15	0	0	8	0.0	98.29

test application time in the both method fairly. Note that, in "TGT" of ES, the value in each parenthesis does not include the identification time for the removed untestable faults, and in "FC" of ES, the value in each parenthesis was calculated as (#Det/#All) $\times$ 100. In the test generation results, the test generation time of our method was longer than that of ES because we used all the constraints in Step 1, and sequential test generation was performed. However, we achieved low fault coverage under 100% fault efficiency compared with that of ES. This means that ES detected faults that do not need to be tested, and our method alleviated over-testing compared with the enhanced scan method. Furthermore, we obtained shorter test application time. Unlike ES, we can perform at-speed test in our method. It implies that the actual test application time of our method becomes much shorter than that of ES. If it is assumed that the scan clock speed of ES is $1/5$ as slow as the rated clock speed, the test application time of our method is 10 or more times faster, on average, than that of ES. Notice that, if we use one-hot encodings during logic synthesis, the advantage of our method will stand out further. This is because the test application time of ES depends on the number of ESFFs.

Table 6 Hardware overheads.

Circuit name	Area OH [%]			Pin OH		
	SS	ES	NS	SS	ES	NS
bbsse	9.5	23.1	9.8	2 (3)	2 (3)	2 (5)
keyb	7.6	18.5	7.6	2 (3)	2 (3)	1 (5)
kirkman	7.8	18.9	7.8	2 (3)	2 (3)	1 (4)
planet	4.5	10.9	15.6	2 (3)	2 (3)	2 (7)
s298	1.5	3.7	23.3	2 (3)	2 (3)	2 (10)
s420	28.7	69.7	28.7	2 (3)	2 (3)	1 (5)
sand	4.0	9.8	4.2	2 (3)	2 (3)	2 (6)
scf	3.6	8.6	6.6	2 (3)	2 (3)	2 (8)

Next, we show the hardware overhead of our method. Columns "Area OH [%]" and "Pin OH" of Table 6 denote the ratio of the area of additional hardware elements to that of the original circuit if the sharing of test pins was not adopted, and the number of additional test pins, respectively. To calculate "Area OH," we considered A_{MUX} and A_{HL} described in the previous section as 7 and 10, respectively. In Table 6, the area overhead of SS was the smallest of all. However, for three cases, our method achieved the same area overhead as that of SS, and low area overhead compared with that of ES except two cases. Note that, as mentioned in Sect. 3.3, if we utilize X values in two-pattern tests, the area overhead can be reduced. Besides, in a controller-data path circuit, the controller is generally much smaller than the data path. Therefore, even if the area overhead of a controller is large, it is not critical in the whole circuit. In the result of pin overheads, our method required a large number of additional test pins for each circuit, which is shown in a

parenthesis, if the sharing of test pins is not adopted. However, if the sharing of test pins is adopted for the respective methods, the pin overheads can be reduced as shown in Table 6.

Finally, we mention the maximum delays of ISTGs. For every case, the maximum delay of the ISTG was smaller than that of the original circuit. This means that our method can always apply tests at a rated clock speed.

5. Conclusions and Future Work

This paper proposed a non-scan design scheme to enhance delay fault testability of controllers. In this scheme, the original behavior of a given STG is used during test application. For faults that cannot be detected by using the original behavior, we append an extra logic, called an *invalid test state and transition generator (ISTG)*, to the original controller. Our scheme can achieve short test application time and at-speed testing. We showed that our method is effective compared with scan-based methods by experiments.

Our future work is to develop ways to reduce hardware overhead and make test generation under constraints more efficient.

Acknowledgments

We would like to thank Prof. Michiko Inoue of Nara Institute of Science and Technology for her valuable comments. This work was supported in part by 21st Century COE (Center of Excellence) Program "Ubiquitous Networked Media Computing" and in part by JSPS (Japan Society for the Promotion of Science) under Grants-in-Aid for Scientific Research B(2) (No. 15300018).

References

[1] Md. Altaf-Ul-Amin, S. Ohtake, and H. Fujiwara, "Design for hierarchical two-pattern testability of data paths," IEEE 10th Asian Test Symp., pp.11–16, 2001.
[2] T.J. Chakraborty, V.D. Agrawal, and M.L. Bushnell, "Improving path delay testability of sequential circuits," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.8, no.6, pp.736–741, Dec. 2000.
[3] K.-T. Cheng, S. Devadas, and K. Keutzer, "A partial enhanced-scan approach to robust delay-fault test generation for sequential circuits," Proc. Int. Test. Conf., pp.403–410, 1991.
[4] B.I. Dervisoglu and G.E. Stong, "Design for testability: Using scan-path techniques for path-delay test and measurement," Proc. Int. Test Conf., pp.365–374, 1991.
[5] M.T.-C. Lee, High-Level Test Synthesis of Digital VLSI Circuits, Artech House, 1997.
[6] S. Ohtake, T. Masuzawa, and H. Fujiwara, "A non-scan approach to DFT for controllers achieving 100% fault efficiency," J. Electron. Test., Theory Appl., vol.16, no.5, pp.553–566, Oct. 2000.
[7] J. Savir and S. Patil, "Scan-based transition test," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.12, no.8, pp.1232–1241, Aug. 1993.
[8] J. Savir and S. Patil, "Broad-side delay test," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.13, no.8, pp.1057–1064, Aug. 1994.
[9] J. Saxena, K.M. Butler, V.B. Jayaram, and S. Kundu, "A case study of IR-drop in structured at-speed testing," Proc. Int. Test Conf., pp.1098–1104, 2003.
[10] T. Villa, T. Kam, R.K. Brayton, and A. Sangiovanni-Vincentelli, Synthesis of Finite State Machines: Logic Optimization, Kluwer Academic Publishers, 1997.

Tsuyoshi Iwagaki received the B.E. degree in electronic engineering from Osaka Institute of Technology, Osaka, Japan, in 2000, and M.E. degree in information science from Nara Institute of Science and Technology, Nara, Japan, in 2002. Presently he is a Ph.D. candidate in Graduate School of Information Science, Nara Institute of Science and Technology. His research interests are VLSI CAD, design for testability and test generation. He is a member of IEEE.

Satoshi Ohtake received the B.E. degree in computer science from the University of Electro-Communications, Tokyo, Japan, in 1995, and M.E. and Ph.D. degrees in information science from Nara Institute of Science and Technology, Nara, Japan, in 1997 and 1999, respectively. He was a Research Fellow of the Japan Society for the Promotion of Science from 1998 to 1999. Presently he is an Assistant Professor at Graduate School of Information Science, Nara Institute of Science and Technology. His research interests are VLSI CAD, design for testability, and test pattern generation. He received IEICE (the Institute of Electronics, Information and Communication Engineers of Japan) Information and Systems Society 2001 Year Paper Award in 2002. He is a member of IEEE Computer Society.

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, and joined Nara Institute of Science and Technology in 1993. In 1981 he was a Visiting Research Assistant Professor at the University of Waterloo, and in 1984 he was a Visiting Associate Professor at McGill University, Canada. Presently he is a Professor at the Graduate School of Information Science, Nara Institute of Science and Technology, Nara, 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 is the author of Logic Testing and Design for Testability (MIT Press, 1985). He received the IECE Young Engineer Award in 1977, IEEE Computer Society Certificate of Appreciation Award in 1991, 2000 and 2001, Okawa Prize for Publication in 1994, IEEE Computer Society Meritorious Service Award in 1996, and IEEE Computer Society Outstanding Contribution Award in 2001. He is an advisory member of IEICE Trans. on Information and Systems and an editor of IEEE Trans. on Computers, J. Electronic Testing, J. Circuits, Systems and Computers, J. VLSI Design and others. Dr. Fujiwara is a fellow of the IEEE, a Golden Core member of the IEEE Computer Society, and a fellow of the Information Processing Society of Japan.