PAPER

Acceleration of Test Generation for Sequential Circuits Using Knowledge Obtained from Synthesis for Testability

Masato NAKAZATO[†a)], **Satoshi OHTAKE**[†], ***Members***, **Kewal K. SALUJA**[††], ***Nonmember***, ***and*** **Hideo FUJIWARA**[†], ***Fellow***

SUMMARY In this paper, we propose a method of accelerating test generation for sequential circuits by using the knowledge about the availability of state justification sequences, the bound on the length of state distinguishing sequences, differentiation between valid and invalid states, and the existence of a reset state. We also propose a method of synthesis for testability (SfT) which takes the features of our test generation method into consideration to synthesize sequential circuits from given FSM descriptions. The SfT method guarantees that the test generator will be able to find a state distinguishing sequence. The proposed method extracts the state justification sequence from the FSM produced by the synthesizer to improve the performance of its test generation process. Experimental results show that the proposed method can achieve 100% fault efficiency in relatively short test generation time.

key words: *sequential circuit, test generation, synthesis for testability, finite state machine, test knowledge*

1. Introduction

For general sequential circuits, it is difficult to achieve 100% fault efficiency in reasonable test generation time even for single stuck-at faults. The full-scan design is utilized to ease the test generation for sequential circuits [1]. However, we cannot perform *at-speed testing* for full-scan designed sequential circuits. To realize at-speed testing, an efficient test generation algorithm for sequential circuits, which generates tests for all the detectable faults and identifies all the untestable faults in reasonable test generation time, is necessary.

Most test generation algorithms for sequential circuits (e.g. HITEC [2], VERITAS [3], STALLION [5] and FASTEST [6]) employ a time frame expansion model of a sequential circuit. The time frame expansion model is a combinational circuit that simulates the exact behavior of the sequential circuit for a given number of time frames.

The HITEC is a well known test generator for sequential circuits. This method consists of two phases. The first phase is the forward time processing phase in which a fault is activated and the resulting fault effect is propagated to a primary output. The second phase is the backward time processing phase which justifies the state required for activating the fault.

The VERITAS test generation method is an extension of the finite state machine (FSM) verification approach. This method constructs a product machine of a good FSM and its faulty version, and carries out reachability analysis by traversing the product machine. The information obtained by the reachability analysis is used to generate a test sequence. Although this simplifies generation of state justification sequences, it is not efficient to generate tests because it has to deal with huge product machines.

In this paper, we propose a method of accelerating test generation for sequential circuits using the knowledge about a set of state justification sequences, the bound on the maximum length of state distinguishing sequences, the information about the valid states and the value of the reset state. We assume that circuits are given in FSM description. For circuits designed at register transfer level (RTL), controllers of the circuits are generally specified by FSM description. The proposed method is effective for such controllers. The sequential circuit is synthesized from a given FSM by a synthesis for testability (SfT) method proposed in this paper which takes the features of our test generation method into consideration. The SfT method guarantees the existence of state distinguishing sequences of the specified length by making the given FSM reduced. Thus, the performance of the test generator is improved as it uses state justification sequences extracted from the completely specified state transition function of the FSM produced by the synthesizer. The proposed method can completely identify every fault in the circuit obtained by the proposed SfT method to be detectable or untestable. In our experiments, 100% fault efficiency is achieved for all the benchmark circuits in relatively short test generation time.

The rest of this paper is organized as follows. Section 2 introduces our circuit model and defines the basic concepts. Section 3 gives the outline of the proposed method. Section 4 describes the proposed SfT algorithm. Section 5 describes the proposed test generation algorithm for sequential circuits synthesized by our SfT method. Section 6 reports the results of experiments of our method using MCNC benchmark circuits. Finally, Sect. 7 describes conclusions and future work.

2. Preliminaries

In this paper we consider synchronous sequential circuits composed of combinational logic and D-type flip-flops

Manuscript received May 8, 2006.
Manuscript revised July 3, 2006.

[†]The authors are with the Nara Institute of Science and Technology (NAIST), Kansai Science City, 630–0192 Japan.
[††]The author is with the University of Wisconsin-Madison, USA.
a) E-mail: masato-n@is.naist.jp

DOI: 10.1093/ietisy/e90–d.1.296

(FFs). All the FFs are controlled by a single clock. We assume that a reset state is defined and a reset signal is available. We also assume that both the good and the faulty circuits can be put on the reset state by applying the reset signal. We consider the single stuck-at fault model but the faults on the clock lines, inside the FFs, and on the reset lines are not included in the fault set.

This paper deals with completely and incompletely specified Mealy-type FSMs. A Mealy-type FSM M is defined as a 6-tuple $\langle \Sigma, O, S, s_r, \delta, \lambda \rangle$. $\Sigma = \{x_0x_1 \ldots x_{n_i-1} \mid x_k \in \{0,1,X\},\ 0 \le k < n_i\}$ is the set of *input vectors* and $O = \{z_0z_1 \ldots z_{n_o-1} \mid z_k \in \{0,1,X\},\ 0 \le k < n_o\}$ is the set of *output vectors*, where X is the don't care, and n_i and n_o are the numbers of inputs and outputs, respectively. $S = \{s_r, s_0, s_1, \ldots, s_{n-2}\}$ is the set of states, where n is the number of states and s_r is the reset state. The functions δ and λ are the state transition function $S \times \Sigma \to S$ and the output function $S \times \Sigma \to O$, respectively. We assume that all the states defined in the FSM are reachable from the reset state s_r. For example, an incompletely specified Mealy-type FSM is shown in Fig. 1. A sequential circuit M_s composed of a combinational circuit part (CC) and FFs as shown in Fig. 2 is synthesized from an FSM, where $x_0, x_1, x_2, \ldots, x_{n_i-1}$ are the primary inputs, $z_0, z_1, z_2, \ldots, z_{n_o-1}$ are the primary outputs and r is the reset input. We classify states represented by FFs of M_s into valid states and invalid states as defined below.

Definition 1 (Valid State and Invalid State): A state s_i represented by the FFs of a sequential circuit M_s is valid if s_i is reachable from the reset state of M_s. Otherwise, s_i is invalid. □

Definition 2 (State Distinguishing Sequence): Let I be an input sequence of an FSM M. Let o_i and o_j be output sequences of I for M with states s_i and s_j, respectively. I is called a state distinguishing sequence with respect to the pair of states s_i and s_j if o_i and o_j are not identical. □

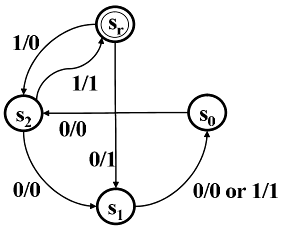


Fig. 1 An incompletely specified finite state machine.

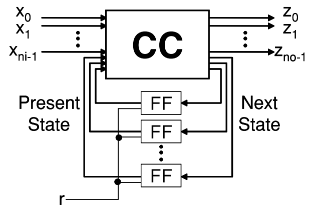


Fig. 2 A sequential circuit M_s synthesized from an FSM.

Definition 3 (Reduced FSM): An FSM is said to be reduced if every pair of states has at least one state distinguishing sequence. □

The proposed test generation method employs a time frame expansion model for the test generation.

Definition 4 (Time Frame): A time frame is the combinational circuit extracted from a sequential circuit by treating its present state lines and next state lines as pseudo primary inputs and pseudo primary outputs, respectively. □

Definition 5 (Time Frame Expansion Model): A time frame expansion model of length l ($l \ge 2$) for a sequential circuit is a combinational circuit constructed by connecting time frames such that the pseudo primary outputs of a time frame i ($0 \le i \le l-2$) is connected to the pseudo primary inputs of a time frame $i+1$. □

Examples of a time frame and a time frame expansion model are shown in Fig. 3 (a) and (b), where $(Y_0^i, Y_1^i, \ldots, Y_{q-1}^i)$ and $(y_0^i, y_1^i, \ldots, y_{q-1}^i)$ are the pseudo primary inputs and the pseudo primary outputs of each time frame i, respectively.

3. Outline of the Proposed Method

The proposed method consists of an SfT method and a test generation method for sequential circuits synthesized by the SfT method. The SfT method synthesizes a sequential circuit to have the three specific characteristics from a given FSM. The proposed test generation method for the sequential circuit utilizes higher level knowledge of its characteristics. By considering each characteristic, we can accelerate the fault excitation, the state justification and the error propagation, respectively. These three specific characteristics are the following.

Characteristic I: Any state in a sequential circuit synthesized from an FSM can be identified as either valid or invalid.
Characteristic II: There exists one to one correspondence between each state of the FSM and each valid state of the sequential circuit.
Characteristic III: For each pair of states in the sequential circuit, there exists a state distinguishing sequence. The maximum length of distinguishing sequences is k which is known.

The flow chart of the proposed method is shown in Fig. 4. The area surrounded by the dotted line shows the SfT method and the outside area is our proposed automatic test pattern generation (ATPG) method. The italicized types in Fig. 4 show knowledge extracted by the SfT. The knowledge is useful for the proposed test generation as follows.
***Information of valid states*:**
In a justification process of test generation, we don't need to justify a fault excitation state of a sequential circuit from the reset state if the fault excitation state is invalid. We can

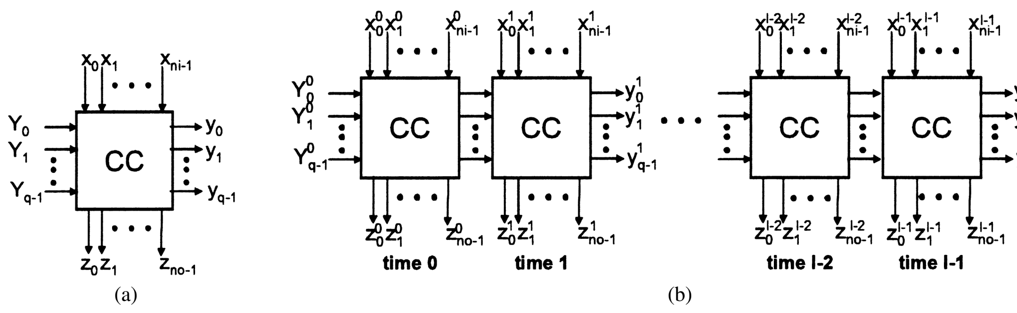


Fig. 3 A time frame of a sequential circuit M_s (a) and a time frame expansion model of M_s (b).

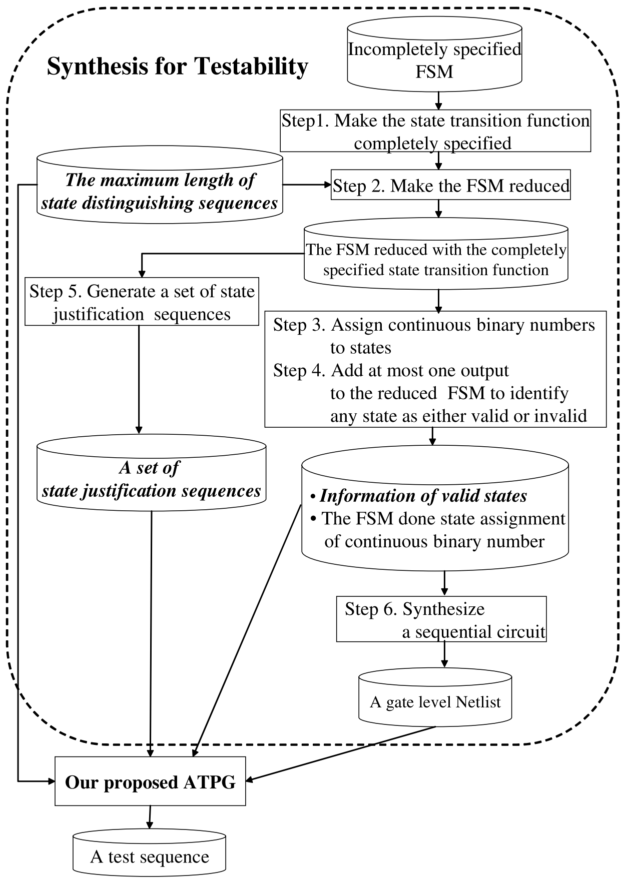

Synthesis for Testability
Incompletely specified FSM
Step1. Make the state transition function completely specified
The maximum length of state distinguishing sequences
Step 2. Make the FSM reduced
The FSM reduced with the completely specified state transition function
Step 5. Generate a set of state justification sequences
Step 3. Assign continuous binary numbers to states
Step 4. Add at most one output to the reduced FSM to identify any state as either valid or invalid
A set of state justification sequences
• Information of valid states
• The FSM done state assignment of continuous binary number
Step 6. Synthesize a sequential circuit
A gate level Netlist
Our proposed ATPG
A test sequence

Fig. 4 The flow chart of the proposed method.

prune the search space of the justification process if we utilize the knowledge that helps to identify the fault excitation state as either valid or invalid. The knowledge "information of valid states" can be obtained since Characteristic I is satisfied. We can accelerate the whole fault excitation process during executing our proposed ATPG by reducing the number of calls of the fault excitation procedure by utilizing this information.

***A set of state justification sequences*:**
The state transition function of a given FSM is incompletely specified. The behavior of the FSM and the behavior of a sequential circuit synthesized from the FSM may be different, because the state transition function of the FSM is appropriately specified during the synthesis process and the state transition function of the sequential circuit becomes completely specified. We can justify the state of the sequential circuit easily if we can utilize the knowledge that helps to justify it by utilizing an input sequence, which is extracted from the FSM description, from the reset state to the excitation state. The knowledge "a set of state justification sequences" can be obtained since Characteristic II is satisfied. We can accelerate the state justification process using this information.

***The maximum length of state distinguishing sequences*:**
In general, we can't know the number of time frames which are required for propagating errors from the fault excitation frame to primary outputs of a sequential circuit in advance. However, we may limit the number of time frames expanded from the fault excitation frame if we have the knowledge of the number. The knowledge "the maximum length of state distinguishing sequences" is given by k since Characteristic III is satisfied. We can accelerate the error propagation process using this information.

4. Synthesis for Testability

In this section, we describe the proposed synthesis for testability (SfT) method for FSMs in detail. In the method, a sequential circuit which has three specific characteristics described in Chapter 3 is synthesized from a given FSM. In order to synthesize a sequential circuit with such characteristics, a given FSM is modified as follows.

- Appropriate values are assigned to some of the coordinates which have don't care values in output vectors of the FSM.
- Extra outputs, if needed, are added to the FSM and appropriate values are assigned to them.

4.1 Formulation of SfT Problem

We formulate the SfT problem as an optimization problem as follows.

Input: An FSM with a reset state and the maximum length of state distinguishing sequences.

Output: A gate level netlist of a sequential circuit which has the three characteristics with a reset, a set of state justification sequences and the number of valid states.

Objective: Minimization of the number of extra outputs.

4.2 Synthesis for Testability Algorithm

In this section, we propose a heuristic algorithm of the SfT since the minimization of the number of extra outputs is NP hard. The heuristic algorithm of the SfT consists of 6 steps as follows:

Step 1: Make the state transition function completely specified

Step 2: Make the FSM reduced

- **Step 2.1:** Try to generate state distinguishing sequences of length 1 for each pair of states of the FSM
- **Step 2.2:** Generate the k-partial state distinguishing tree in order to confirm that there exists a state distinguishing sequence of length less than or equal to k for each pair of states of the FSM and a state compatibility graph
- **Step 2.3:** Determine the number of extra outputs from the state compatibility graph

Step 3: Assign continuous binary numbers to states in order to identify as either a valid or an invalid state

Step 4: Add an extra output to the FSM in order to guarantee existence of a state distinguishing sequence of length 1 for each pair of any valid state and any invalid state

Step 5: Generate a set of state justification sequences

Step 6: Synthesize a sequential circuit

In the heuristic algorithm, we use a k-partial state distinguishing tree and a state compatibility graph. We first define them as follows.

To clarify the discussion of state distinguishing sequences, we extend the definition of the successor tree defined in the literature [4] as follows.

Definition 6 (k-Partial State Distinguishing Tree): Let M be an FSM. Let $T_k = (V_{T_k}, E_{T_k})$ be a tree of level k $(0 \leq k)$, where V_{T_k} is a set of nodes $\{v_{i,j_i} \mid 0 \leq i \leq k, 0 \leq j_i < |\Sigma|^i\}$ and E_{T_k} is a set of edges $\{(v_{i,j_i}, v_{i+1,j_i\cdot|\Sigma|+t}) \mid 0 \leq i < k, 0 \leq j_i < |\Sigma|^i, 0 \leq t < |\Sigma|\}$. An edge $(v_{i,j_i}, v_{i+1,j_i\cdot|\Sigma|+t})$ is also referred to as $e_{i,j_i\cdot|\Sigma|+t}$ and $\sigma_t \in \Sigma$ is associated with the edge. Let $\mathcal{U}$ be a set of states, which will be tried to be distinguished, of M and it is referred to as an *initial uncertainty*. Let U^p_{i,j_i} be a set of 3-tuples $\{u_n \mid 0 \leq n < |\mathcal{U}|\}$ and be associated with v_{i,j_i}, where p is the characteristic number and a 3-tuple $u_n \in U^p_{i,j_i}$ is composed of $s_n \in \mathcal{U}$, s_ℓ which is a state succeeded by applying the input sequence, which corresponds to a path from $v_{0,0}$ to v_{i,j_i}, to s_n, and o_ℓ, which appears as the last output vector by applying the input sequence to s_n, and is denoted in $< s_n, s_\ell, o_\ell >$. Here, u_n is called a *distinguished state history* (DSH). For each U^p_{i,j_i} of v_{i,j_i}, sets of DSHs of $v_{i+1,j_i\cdot|\Sigma|+t}$ are generated so that the DSHs are obtained by applying σ_t to M with the state of the second element of each $u_n \in U^p_{i,j_i}$ and these are classified into the sets where a set has the DSHs whose third elements are the same and they are different from the third elements of the DSHs in the other sets. The tree T_k is called a *k-partial state distinguishing tree*. □

Figure 5 shows the 2-partial state distinguishing tree $T_2 = (V_{T_2}, E_{T_2})$ for the FSM of Fig. 1. Here, we suppose an initial uncertainty $\mathcal{U}$ of the FSM is a set of all the states of the FSM. Suppose a set of DSHs, $U^0_{0,0} = [u_0, u_1, u_2, u_3] = [< s_r, s_r, \mathcal{X} >, < s_0, s_0, \mathcal{X} >, < s_1, s_1, \mathcal{X} >, < s_2, s_2, \mathcal{X} >]$ is assigned to $v_{0,0} \in V_{T_2}$, where $\mathcal{X}$ is don't care vector such that all the bits of the output vector are don't care. By applying the vector $\sigma_1 = 1$ to each DSH of $v_{0,0}$, two sets $U^0_{1,1}$ and $U^1_{1,1}$, where $U^0_{1,1}$ is $[u_0] = [< s_r, s_2, 0 >]$ and $U^1_{1,1}$ is $[u_1, u_2, u_3] = [< s_0, s_0, 1 >, < s_1, s_0, 1 >, < s_2, s_r, 1 >]$, respectively, are associated with $v_{1,1}$. By applying the sequence $\sigma_1\sigma_0 = 10$ to each DSH of $v_{0,0}$, three sets $U^0_{2,2}$, $U^1_{2,2}$ and $U^2_{2,2}$, where $U^0_{2,2}$ is $[u_0] = [< s_r, s_1, 0 >]$, $U^1_{2,2}$ is $[u_1, u_2] = [< s_0, s_2, 0 >, < s_1, s_2, 0 >]$ and $U^2_{2,2}$ is $[u_3] = [< s_2, s_1, 1 >]$, respectively, are associated with $v_{2,2}$.

Definition 7 (State Compatibility Graph): An undirected graph $G = (V_G, E_G)$, where $v \in V_G$ is a vertex corresponding to a state of an FSM and $e \in E_G$ is an edge corresponding to a pair of indistinguishable states of the FSM, is said to be *a state compatibility graph*. □

$\mathcal{U}$ is the initial uncertainty of an FSM. Let $D^{j_k}_s$ be a set of the distinguished states for $s \in \mathcal{U}$ of a leaf node $v_{k,j_k} \in V_{T_k}$ of a k-partial state distinguishing tree T_k obtained from the FSM where a distinguished state is a state in $\mathcal{U}$ except for s and is distinguishable from s. For all the leaf node of T_k, a set of states, which are distinguished from s, of $\mathcal{U}$ is obtained by the following formula: $\bigcup^{|\Sigma|^k}_{j_k=0} D^{j_k}_s$. The set of indistinguishable states of s is the complement of $\bigcup^{|\Sigma|^k}_{j_k=0} D^{j_k}_s$ for $\mathcal{U}$. We make the state compatibility graph based on pairs of indistinguishable states obtained from the above. Figure 6 shows the state compatibility graph corresponding to Fig. 5.

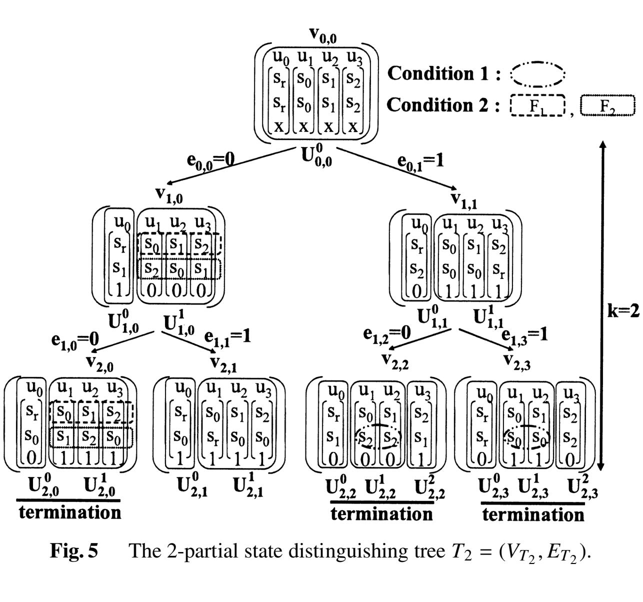


Fig. 5 The 2-partial state distinguishing tree $T_2 = (V_{T_2}, E_{T_2})$.

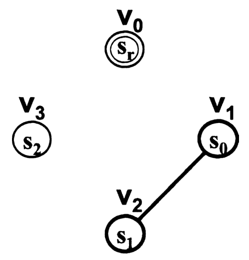


Fig. 6 The state compatibility graph corresponding to Fig. 5.

In this figure, indistinguishable states are s_0 and s_1.

Then, we describe the process for every step in detail.

Step 1: Let s_i be a state in M such that, there exist input vectors for which next states of the state are not specified in the state transition function. For each input vector $\sigma \in \Sigma$, which is not defined for a transition from the state s_i, of M, a state transition from s_i to s_i (i.e., a self-loop) in M for σ is added to the state transition function. An output vector o_i for the self-loop is added to the output function. All the bits of o_i are don't care. The FSM obtained in this step is referred to as M^α.

Step 2: To make every pair of states defined in M^α distinguishable, we perform the following three processes.

Step 2.1: For each input vector $\sigma \in \Sigma$ of M, we try to distinguish all the pairs of states s_i and s_j ($s_i \neq s_j$) of M. We perform the following two processes.

Step 2.1.1: Let o_i and o_j be output vectors of σ for M^α with s_i and s_j, respectively. We assign '0' or '1' to appropriate don't care bits of o_i in order to differentiate o_i and o_j if o_j is covered by o_i. Here, we define the relation between vectors a and b which have don't care values. We say that a covers b if $A \supset B$, where A and B are the sets of values represented by a and b, respectively.

Step 2.1.2: If o_i and o_j are the same and still have don't care bits, we assign '0' or '1' to some don't care bits of o_i and o_j to make o_i and o_j different. Let K be a set of such the same output vectors. Let X $(= x_0x_1 \ldots x_{n_X-1})$ be a vector composed of don't care bits in $\kappa \in K$, where n_X is the number of don't care bits in κ. The number of values represented by X is 2^{n_X}. If $|K| \leq 2^{n_X}$, the unique value can be assigned to each κ. In this case, for each κ, we assign an unique value among 2^{n_X} to the don't care bits. If $|K| > 2^{n_X}$, we assign a value to each κ so that the number of the same output vectors is minimized. In this case, the continuous binary number is cyclically assigned to the don't care bits in each κ. The FSM obtained in this step is referred to as M^β.

Step 2.2: We construct the k-partial state distinguishing tree to examine whether a pair of states s_i and s_j of M could be distinguishable by applying input sequences of length less than or equal to k to M^β with s_i and with s_j. We use the following two conditions of pruning for construction of the tree. Here, v and U^p are a current observed node of the k-partial state distinguishing tree and a set of DSHs of v whose the third elements are the same and they are different from the third elements of DSHs in the other sets. Let V_q be a set of nodes existing on the path from the root to v. Let U_q^p be a set of DSHs of $v_q \in V_q$ whose the third elements are the same and they are different from the third elements of DSHs in the other sets. Let $F_1(U^p)$ and $F_2(U^p)$ be a set of first elements of all the DSHs in U^p and a set of the second elements of all the DSHs in U^p, respectively.

Condition 1: For each U^p of v such that $|U^p| \geq 2$, all the elements of $F_2(U^p)$ are the same.

Condition 2: There exists v_q such that for each U^p, whose number of DSHs is larger than 1, of v, there exists U_q^p, which satisfies $F_1(U_q^p) = F_1(U^p)$ and $F_2(U_q^p) = F_2(U^p)$, of v_q.

If v satisfies Condition 1 or Condition 2, v is a termination node. For example, in Fig. 5, $v_{2,2}$ and $v_{2,3}$ satisfy Condition 1 and $v_{2,0}$ satisfies Condition 2. For $v_{2,2}$, all the elements of $F_2(U_{2,2}^1)$ are the same state s_2. For $v_{2,0}$, $F_1(U_{2,0}^1)$ and $F_2(U_{2,0}^1)$ of $v_{2,0}$ are equal to $F_1(U_{1,0}^1)$ and $F_2(U_{1,0}^1)$ of $v_{1,0}$ on the path from $v_{0,0}$ to $v_{2,0}$, respectively. In this case, the level of termination nodes is the same as the maximum level of the 2-partial state distinguishing tree.

Step 2.3: We construct the state compatibility graph obtained from the k-partial state distinguishing tree for representing all the indistinguishable state pairs of M^β.

For example, we obtain the state compatibility graph in Fig. 6 from Fig. 5. We can see that the indistinguishable states are s_0 and s_1 in the state compatibility graph.

We perform the following process in order to distinguish these indistinguishable states. Some outputs are added to M^β to distinguish all the indistinguishable state pairs. The problem to find the minimum number of additional outputs to distinguish all the indistinguishable state pairs is solved as a vertex coloring problem [7] of the state compatibility graph. The number of outputs to be added to M^β is obtained by the following formula: $n_a = \left\lceil \frac{\log_2 C}{|\Sigma|} \right\rceil$, where C is the number of colors obtained by solving the vertex coloring problem and n_a is the number of the additional outputs.

Let P be a set of values represented by the additional outputs. Let f_i be a mapping $\Sigma \xmapsto{f_i} P$ such that $f_i \neq f_j$, $^\forall i, j \mid 1 \leq i, j \leq C \wedge i \neq j$. For any $\sigma \in \Sigma$, the output function of M^β is changed so that the value of the additional outputs become $f_i(\sigma)$ for the state corresponding to each vertex, whose degree is more than or equal to 1, of the state compatibility graph. Thus, a state distinguishing sequence of length less than or equal to k is guaranteed for any state pair. The FSM obtained by this step is referred to as M^γ.

Step 3: Let n_s be the number of states of the FSM M^γ. The number of FF, n_{ff}, in a sequential circuit synthesized from M^γ is equal to $\lceil \log_2 n_s \rceil$. The number of valid states of the circuit is equal to n_s and the number of invalid states, n_{iv}, is equal to $2^{n_{ff}} - n_s$. Binary numbers within the range of 0 to $n_s - 1$ are used for the state assignment of M^γ and binary numbers within the range of n_s to $2^{n_{ff}} - 1$ (if $n_{iv} \neq 0$) are used for values of the state variables of invalid states of the sequential circuit. The value assigned to the reset state s_r is referred to as n_r.

Step 4: To guarantee existence of a state distinguishing sequence of length 1 for each pair of any valid state and any invalid state of the sequential circuit synthesized by the SfT, one output is added to the FSM if n_{iv} is not equal to 0. This process means that a pair of any valid state and any invalid state is made distinguishable in order to realize Characteristic III. For a transition from a valid state to a valid state, '0' is assigned to the output. For a transition from an invalid state, '1' is assigned to the output. For a transition from a valid state to an invalid state, we have already considered in Step 1; all the transitions from valid states are succeeded by valid states. The FSM obtained by this step is referred to as M^{ϵ}.

Step 5: For each valid state of M^{ϵ}, an input sequence to reach the state from the reset state is generated by a breadth-first search on the state transition graph of M^{ϵ}. By searching breadth-first fashion, the shortest input sequence is guaranteed for each state. The input sequence is called a state justification sequence.

A set of state justification sequences for all the valid states of M^{ϵ} is referred to as S_{si}.

Step 6: A gate level sequential circuit is synthesized from M^{ϵ} by a logic synthesis tool.

5. Test Generation Algorithm for Sequential Circuits

In this section, we describe the proposed test generation method that utilizes the knowledge of S_{si}: a set of state justification sequences, k: the maximum length of state distinguish sequences, n_s: the number of valid states, and n_r: the value of a reset state extracted by the SfT.

Our test generation method uses a time frame expansion model. A time frame expansion model has multiple faults because every time frame has the same single stuck-at fault. Therefore, our test generation method uses a 9 valued logic system [8], [9] for the test generation to deal with multiple faults.

Figure 7 shows the flow chart of our test generation method for sequential circuits. The proposed test generation method consists of three processes: fault excitation, state justification and error propagation.

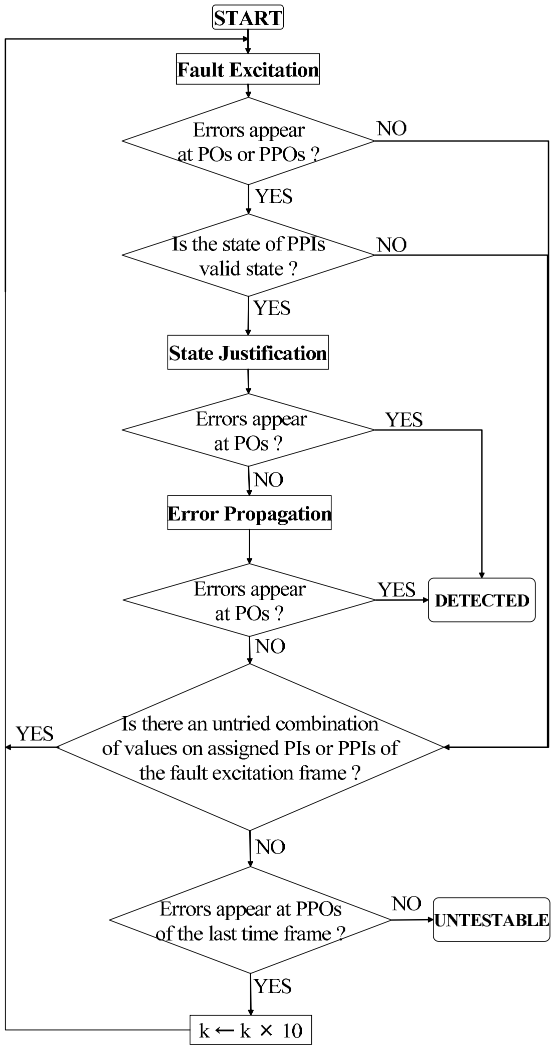

START
Fault Excitation
Errors appear at POs or PPOs ?
NO
YES
Is the state of PPIs valid state ?
NO
YES
State Justification
Errors appear at POs ?
YES
NO
Error Propagation
Errors appear at POs ?
YES
DETECTED
NO
YES
Is there an untried combination of values on assigned PIs or PPIs of the fault excitation frame ?
NO
Errors appear at PPOs of the last time frame ?
NO
UNTESTABLE
YES
k ← k × 10

Fig. 7 The flow chart of the proposed test generation method for sequential circuits.

5.1 Fault Excitation

For a target fault, fault excitation finds an excitation vector which is assigned to primary inputs and pseudo primary input to produce errors and to propagate them to the primary outputs and/or the pseudo primary outputs of the fault excitation frame. The pseudo primary input part of an excitation vector is referred to as an excitation state n_e. The number of valid states, n_s, helps generating a valid excitation vector which is an excitation vector whose excitation state is a valid state. If an excitation state is valid, the state may be justified from the reset state. However, if the excitation state is invalid, state justification is not required because the state cannot be justified from the reset state. Hence, the proposed method can prune a part of search space of a test generation. This search space pruning is realized by comparing n_s with n_e. If n_e is less than n_s, the excitation state is valid. Otherwise, the state is invalid. This feature saves a large amount of time for trying to generate invalid excitation vector and trying to justify the invalid excitation state. If there exists no valid state to excite the fault, the fault is proved untestable.

5.2 State Justification

Once an excitation vector is found, state justification is performed. The excitation state must be justified for both the fault-free circuit and the faulty circuit. We have a set of state justification sequences, S_{si}, for the fault-free circuit. The fault-free state justification can be easily done by choosing the state justification sequence for the excitation state from S_{si}. No backtracking is required and no failure can occur in this step. The next step is to confirm if the fault-free state justification sequence is also valid for the faulty circuit. This is confirmed by fault simulation using the fault-free state justification sequence and observing if any invalidation occurs. An invalidation means that a state transition of the faulty circuit is different from the fault free circuit. If an invalidation occurs, the state justification sequence cannot justify the given excitation state because the state justification sequence is not guaranteed to work under the faulty circuit. However, if an invalidation occurs, some error must appear on the pseudo primary outputs of some frame (we call this an actual excitation state) between the reset frame and the fault excitation frame. We try to propagate errors from the actual excitation state. If some error appears on the primary outputs between the reset frame and the fault excitation frame, the fault is detected.

5.3 Error Propagation

If a fault is not identified as detected or untestable by the first two processes, error propagation is performed. Time frames of length k are added to the (actual) fault excitation frame. The error propagation process determines primary input values of the expanded time frames to propagate an error to a primary output. This process may not propagate any error to any primary output and any pseudo primary output because errors may be masked by the multiple faults within the added k time frames. In this case, we try to search a different excitation state by returning to the fault excitation process. On the other hand, any error is not propagated to any primary output but any error is propagated to some pseudo primary output of the last time frame. This is because k-state distinguishing sequence is not guaranteed for faulty circuit. Therefore, in order to make error propagation complete, the number of time frames expanded from the fault excitation frame has to be increased (e.g., $k = k \times 10$, where this number 10 might be changed empirically). We perform fault excitation again.

6. Experimental Results

Table 1 shows characteristics of the MCNC FSM benchmarks [10] and the results of SfT. All the experiments except for the proposed SfT were performed on a SUN Blade 2000 (CPU 1 GHz × 2) with 8 GB memory. The experiments for the proposed SfT were performed on a PC/AT machine (CPU Athlon 3000+) with 1 GB memory. Design Compiler (Synopsys) is used as a logic synthesis tool for the Step 6 of the proposed SfT method. The number of benchmarks is 53. For all the benchmarks, the proposed method could perform until Step 5. However, Design Compiler was unable to perform Step 6 for 14 benchmarks because of restrictions on Design Compiler. One of the restrictions is that the size of FSM descriptions which Design Compiler can read is limited.

All the benchmarks shown in Table 1 were synthesized by using some optimization options which optimize the area and the delay of a circuit. The first four columns give the benchmark name and the numbers of primary inputs, primary outputs and states, respectively. The column "k" gives the maximum length of the state distinguishing sequences. The columns "#EO," "HOH" and "#MLG" give the results of the proposed SfT. The column "#EO" denotes the number of extra outputs added to each benchmark. The column

Table 1 Characteristics of FSM benchmarks and results of SfT.

Circuit	#Input	#Output	#State	k	#EO	HOH (%)	#MLG	
							Prop.	Orig.
bbara	6	2	10	3	2	3.36	14	17
bbsse	9	7	13	2	1	43.03	28	34
bbtas	4	2	6	5	1	3.45	11	8
beecount	5	4	7	2	1	12.82	27	11
cse	9	7	16	1	0	27.3	22	30
dk14	5	5	7	1	1	−0.28	22	26
dk15	5	5	4	1	0	0	18	18
dk16	4	3	27	2	1	9.76	48	39
dk17	4	3	8	1	0	0	15	15
dk27	3	2	7	2	1	−1.28	12	8
ex1	11	19	20	1	1	69.61	34	35
ex3	4	2	10	2	1	35.15	18	23
ex4	8	9	14	1	1	16.36	16	25
ex5	4	2	9	2	1	39.56	17	18
ex6	7	8	8	1	0	−1.69	17	26
keyb	9	2	19	1	1	44.6	36	29
kirkman	14	6	16	1	0	104.34	45	44
lion	4	1	4	2	0	0	9	12
lion9	4	1	9	6	1	14.29	17	33
mc	5	5	4	1	0	0	6	8
opus	7	6	10	1	1	28.52	21	28
planet	9	19	48	2	1	24.76	41	33
planet1	9	19	48	2	1	24.76	41	33
pma	10	8	24	1	1	303.04	100	59
s1	10	6	20	1	1	−5.83	47	67
s1488	10	19	48	2	1	−0.71	48	88
s1494	10	19	48	2	1	−20.73	42	73
s208	13	2	18	3	2	2.41	23	24
s27	6	1	6	2	2	4.4	14	15
s298	5	6	218	6	2	11.68	107	146
s386	9	7	13	2	1	−1.47	29	41
s420	21	2	18	1	2	56.02	14	24
styr	11	10	30	1	2	62.42	41	30
sse	9	7	16	1	0	77.27	76	48
tma	9	6	20	1	1	120.75	5	8
tbk	8	3	32	1	1	36.39	46	67
tav	6	4	4	3	0	0	48	50
train11	4	1	11	2	2	28.83	26	19
train4	4	1	4	2	0	4.08	8	17

"HOH" denotes the hardware overhead which is the ratio of the area of the sequential circuit synthesized by the proposed SfT to that of the original sequential circuit synthesized by the ordinary synthesizer. The subdivided columns "Orig." and "Prop." in "#MLG" are the maximum level of gates in the original sequential circuit and that of gates in the sequential circuit obtained by the proposed SfT, respectively.

In this experiment, the maximum number of outputs added to the FSMs is two: one for distinguishing between valid and invalid states and the other for making the given FSM reduced. The number of extra outputs decreases when the FSM is reduced by only assigning values to the don't cares in the output vectors of the FSM.

The average hardware overhead is 30.18%. However, the hardware overhead of the circuit 'pma' is more than 300%. It is considerable that the logic synthesis tool may not be able to simplify logic because we assign logic values to coordinates with don't cares in input vectors and output vectors to make a given FSM reduced. However, a hardware

Table 2 Test generation results for each method.

Circuit	TGT [s]			TTGT [s]			FC [%]			FE [%]			TSL		
	m1	m2	m3	m1	m2	m3	m1	m2	m3	m1	m2	m3	m1	m2	m3
bbara	> 10h	2.12	0.09	> 10h	2.26	0.24	94.95	95.63	95.63	96.46	100.00	100.00	891	690	486
bbsse	1.70	2.03	0.28	1.83	2.23	0.44	98.27	97.46	97.46	100.00	100.00	100.00	486	762	801
bbtas	1.60	2.89	0.03	1.68	3.03	0.11	98.61	95.24	95.24	100.00	100.00	100.00	276	318	216
beecount	0.26	0.28	0.02	0.38	0.38	0.14	97.53	97.80	97.80	100.00	100.00	100.00	342	285	270
cse	2.28	1.69	0.21	2.49	1.87	0.40	100.00	100.00	100.00	100.00	100.00	100.00	1119	915	924
dk14	0.32	0.29	0.02	0.45	0.41	0.13	98.61	98.21	98.21	100.00	100.00	100.00	300	288	270
dk15	0.09	0.15	0.00	0.21	0.28	0.11	100.00	100.00	100.00	100.00	100.00	100.00	108	108	117
dk16	> 10h	92.75	0.63	> 10h	93.06	0.87	98.29	98.17	98.17	99.74	100.00	100.00	1323	1368	951
dk17	0.13	0.12	0.02	0.22	0.24	0.16	100.00	100.00	100.00	100.00	100.00	100.00	171	180	171
dk27	0.10	0.12	0.00	0.21	0.21	0.11	95.59	94.44	94.44	100.00	100.00	100.00	81	75	54
ex1	5538.05	> 10h	1.02	5538.23	> 10h	1.32	97.57	98.52	98.52	100.00	99.89	100.00	906	1026	990
ex3	10.53	0.53	0.06	10.65	0.65	0.19	96.46	97.27	97.27	100.00	100.00	100.00	372	399	330
ex4	0.82	0.31	0.03	0.97	0.39	0.15	97.08	97.08	97.08	100.00	100.00	100.00	660	306	372
ex5	5639.33	3.86	0.04	5639.44	3.96	0.16	95.65	95.20	95.20	100.00	100.00	100.00	417	390	294
ex6	0.19	0.16	0.01	0.30	0.27	0.14	100.00	100.00	100.00	100.00	100.00	100.00	240	183	201
keyb	10.39	30.20	1.12	10.59	30.42	1.35	97.81	97.38	97.38	100.00	100.00	100.00	1050	1104	1188
kirkman	1.13	1.51	0.61	1.34	1.80	0.96	100.00	100.00	100.00	100.00	100.00	100.00	1437	1698	1839
lion	0.08	0.11	0.00	0.16	0.22	0.08	100.00	100.00	100.00	100.00	100.00	100.00	120	120	81
lion9	> 10h	36.69	0.05	> 10h	36.81	0.16	95.71	95.62	95.62	98.57	100.00	100.00	408	456	432
mc	0.09	0.09	0.00	0.19	0.20	0.07	100.00	100.00	100.00	100.00	100.00	100.00	87	87	51
opus	7.72	7.19	0.10	7.85	7.30	0.22	98.76	96.50	96.50	100.00	100.00	100.00	408	456	498
planet	1562.30	1160.20	2.26	1562.83	1160.95	3.22	98.96	98.97	98.97	100.00	100.00	100.00	2721	3426	3963
planet1	1562.30	1160.20	2.26	1562.83	1160.95	3.22	98.96	98.97	98.97	100.00	100.00	100.00	2721	3426	3963
pma	> 10h	10185.70	25.71	> 10h	10187.76	27.88	98.83	99.44	99.44	99.61	100.00	100.00	1248	4404	4527
s1488	1282.41	4544.74	1.18	1283.00	4545.43	1.79	98.74	99.01	99.01	100.00	100.00	100.00	2787	3693	3129
s1494	1965.04	6027.33	1.37	1965.60	6027.95	2.07	98.71	98.87	98.87	100.00	100.00	100.00	2676	2856	3375
s1	> 10h	170.99	1.01	> 10h	171.23	1.22	97.02	96.84	96.84	99.46	100.00	100.00	1182	1542	972
s208	> 10h	2.45	0.04	> 10h	2.55	0.18	95.00	94.44	94.44	98.57	100.00	100.00	603	612	510
s27	0.27	0.19	0.12	0.36	0.28	0.21	91.03	90.12	90.12	100.00	100.00	100.00	102	102	81
s298	> 10h	> 10h	48.01	> 10h	> 10h	57.09	95.75	95.75	92.12	96.12	92.56	100.00	29325	22422	14052
s386	4.60	2.44	0.16	4.78	2.60	0.30	97.22	96.27	96.27	100.00	100.00	100.00	663	795	567
s420	> 10h	3.66	0.08	> 10h	3.78	0.22	95.00	96.12	96.12	98.57	100.00	100.00	651	858	1020
sse	377.75	1.61	0.44	377.91	1.79	0.63	97.17	97.26	97.26	100.00	100.00	100.00	543	645	747
styr	18.30	30.22	4.25	18.79	30.92	4.95	99.54	99.72	99.72	100.00	100.00	100.00	2367	2460	1783
tav	0.09	0.11	0.01	0.18	0.19	0.11	100.00	100.00	100.00	100.00	100.00	100.00	81	81	120
tbk	> 10h	7.47	2.99	> 10h	8.12	3.08	98.85	100.00	100.00	98.85	100.00	100.00	1200	2586	2025
tma	> 10h	> 10h	1.17	> 10h	> 10h	1.69	99.02	98.92	98.92	99.41	99.91	100.00	789	1287	1233
train11	55.01	0.71	0.05	55.13	0.86	0.18	97.31	97.06	97.06	100.00	100.00	100.00	342	300	321
train4	0.13	0.13	0.00	0.23	0.22	0.09	100.00	100.00	100.00	100.00	100.00	100.00	120	138	99

overhead may be able to be reduced by carrying out recoding of input vectors and output vectors. Hence, still there is a room for research, how to assign coordinates with don't cares in input vectors and output vectors during SfT.

The maximum time spent for the proposed SfT method is about twenty minutes. For small circuits (ex. lion, bbara, bbsse, bbtas and so on), the time spent for the SfT is 0.1 second or less. Notice that since we have no way to obtain an optimal length of state distinguishing sequences, in the experiments, we determined a practical length k by running trials as follows. For an FSM, we set a time limit to determine k for the trials. For k, we tried recursively to create k-partial state distinguishing tree and to confirm that there exists a state distinguishing sequence of length less than or equal to k for each pair of states of an FSM by running the procedure of Step 2.2 in the proposed SfT method. If the whole run time of the trials exceeds the time limit or the number of distinguishable states does not increase compared with that of the previous trial, we take k of the previous trial. Otherwise, we increment k by one and perform the next trial. In the experiments, we incremented k from one and set the time limit one hour for each benchmark circuit. To find a way to do that is included in our future work.

For almost all the circuits, the maximum level of gates in the sequential circuits obtained by the SfT is less than that in the sequential circuit synthesized from the FSM without using the SfT. However, the maximum level of gates in some circuits synthesized by the proposed SfT becomes about twice compared with that synthesized by the ordinary synthesizer. In this case, the performance of these circuits degrades from the original sequential circuits. Typically, the maximum level of gates in these circuits is much less than that of the data path in a VLSI. We believe that the performance of these circuits depends on the performance of a data path and the delay of a circuit is able to be absorbed on the data path side.

Table 2 shows the test generation results for three different methods. Method 1 applies TestGen (Synopsys) to the original sequential circuit synthesized from the FSM without using the proposed SfT. Method 2 applies TestGen to the sequential circuit obtained by the proposed SfT. Method 3 applies the proposed test generation method to the sequential circuit obtained by the proposed SfT. The column "TGT [s]" denotes the time, in seconds, which was spent on the test generation excluding the fault simulation time. The subdivided columns "m1," "m2" and "m3" denote the method 1, the method 2 and the method 3, respectively. The column "TTGT [s]" denotes the time, in seconds, which is the sum of "TGT" and the fault simulation time. The fault simulation time is calculated by "TTGT" - "TGT." The time '> 10h' in the columns "TGT [s]" and "TTGT [s]" means that the test generation did not finish within 10 hours.

All the methods performed the equivalent fault analysis and the fault simulation which are implemented in TestGen. Since our test generator does not have the desired fault simulation capability, the method 3 requires a call to the external fault simulator. Therefore, in order to compare the test generation time of the proposed method with that of TestGen on equal terms, we performed the fault simulation for the test sequence obtained by the test generation.

The total test generation time for each benchmark of the method 3 is shorter than that of the other methods. For some benchmarks, the total test generation time of the method 2 is longer than that of the method 1. For all the experiments of the method 3, *k* was not increased. In other words, the error propagation process of the method 3 performed completely within *k* frames expanded for the process where *k* was given as the input of our proposed ATPG. We believe that the method 3 can effectively use the knowledge obtained by the SfT but the method 2 cannot effectively use it. The average total test generation time of the method 1, that of the method 2 and that of the method 3 are 9694.06 (s), 3371.99 (s) and 2.97 (s), respectively. The actual average total test generation time of the method 1 and the method 2 will be longer because these methods did not achieve 100% fault efficiency for some circuits. The method 3 identified all the untestable faults within reasonable time.

The columns "FC [%]," "FE [%]," and "TSL" are the fault coverage, the fault efficiency and the length of the test sequence, respectively. The method 3 can achieve 100% fault efficiency for all the benchmarks within reasonable time and the time is shorter than both the test generation time of the method 1 and that of the method 2. However, the method 1 and the method 2 did not achieve 100% fault efficiency for several these benchmarks. Particularly, both the method 1 and the method 2 for 's298' did not achieve 100% fault efficiency within 10 hours. The proposed method can perform faster test generation than the conventional method for benchmarks.

The test sequence length of the method 3 is shorter than that of other methods for about a half of benchmarks. There are some cases where the test sequence length of the method 3 is longer than that of other methods, this is because TestGen uses techniques of test sequence compression.

7. Conclusions and Future Work

In this paper, we proposed a method for high speed test generation for sequential circuits with some specific characteristics. Such a sequential circuit can be synthesized from a given FSM by the synthesis for testability (SfT) method which takes the features of our test generation method into consideration. We accelerated test generation for sequential circuits by utilizing the knowledge that consisted of a set of state justification sequences, the maximum length of state distinguishing sequences, the number of valid states, and the value of the reset state extracted by the SfT. The experimental results show that the proposed method can achieve 100% fault efficiency in shorter test generation time compared to a state of the art test generator. Reduction of hardware overhead caused by the SfT method, its speed up, and further reduction of test generation time are the issues to be investigated in our future work.

Acknowledgements

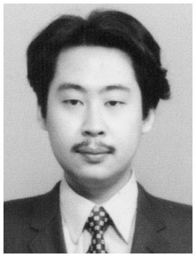
Authors would like to thank Prof. Michiko Inoue, Dr. Tomokazu Yoneda, Dr. Virendra Singh and other members of Computer Design and Test Laboratory of Nara Institute Science and Technology for their valuable discussions. This work was supported in part by 21st Century Center of Excellence Program "Ubiquitous 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), for Young Scientists (B) (No.17700062) and in part by the grant of JSPS Research Fellowship (No.L04509).

References

[1] H. Fujiwara, Logic Testing and Design for Testability, The MIT press, Cambridge, 1985.
[2] T. Niermann and J. Patel, "HITEC: A test generation package for sequential circuits," Proc. European Design Automation Conference, pp.214–218, 1991.
[3] H. Cho, G.D. Hachtel, and F. Somenzi, "Redundancy identification/removal and test generation for sequential circuits using implicit state enumeration," IEEE Trans. Computer-Aided Design, vol.12, pp.935–945, July 1993.
[4] M. Samiha and Z. Yervant, Principles of testing electronic systems, Wiley-Interscience, 2000.
[5] H.K. Ma, S. Devadas, A.R. Newton, and A. Sangiovanni-vincentelli, "Test generation for sequential circuit," IEEE Trans. Computer-Aided Design, vol.7, no.10, pp.1081–1093, Oct. 1988.
[6] T.P. Kelsey and K.K. Saluja, "Fast test generation for sequential circuits," Int. Conf. on Computer-Aided Design 1989, pp.354–357, Nov. 1989.
[7] J. Gross and J. Yellen, Graph Theory and Its Applications, CRC press, 1991.
[8] C.W. Cha, W.E. Donath, and F. Özgüner, "9-v algorithm for test pattern generation of combinational digital circuits," IEEE Trans. Comput., vol.C-27, pp.193–200, March 1978.
[9] P. Muth, "A nine-valued circuit model for test generation," IEEE Trans. Comput., vol.C-25, no.6, pp.630–636, June 1976.

[10] The CAD benchmarks at North Carolina State University, http://www.cbl.ncsu.edu/www/.

Masato Nakazato received the B.E. degree in Photonics from Ritsumeikan University, Shiga, Japan, in 2001, and the M.E. degree in information science from Nara Institute of Science and Technology, Nara, Japan, in 2005. Presently he is a doctoral student at Graduate School of Information Science, Nara Institute of Science and Technology. His research interests are VLSI CAD, test generation for logic circuits and synthesis for testability.



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 here 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 Information and Systems Society 2001 Year Paper Award in 2002. He is a member of IEEE Computer Society, and a member of the Information Processing Society of Japan.

Kewal K. Saluja obtained his Bachelor of Engineering (BE) degree in Electrical Engineering from the University of Roorkee, India in 1967, MS and Ph.D. degrees in Electrical and Computer Engineering from the University of Iowa, Iowa City in 1972 and 1973 respectively. He is currently with the Department of Electrical and Computer Engineering at the University of Wisconsin-Madison as a Professor, where he teaches courses in logic design, computer architecture, microprocessor based systems, VLSI design and testing, and fault-tolerant computing. Prior to this he was at the University of Newcastle, Australia. Professor Saluja has held visiting and consulting positions at various national and international institutions including University of Southern California, Hiroshima University, Nara Institute of Science and Technology, and the University of Roorkee. He has also served as a consultant to the United Nations Development Program. He was the general chair of the 29th FTCS and he served as an Editor of the IEEE Transactions on Computers (1997–2001). He is currently the Associate Editor for the letters section of the Journal of Electronic Testing: Theory and Applications (JETTA). His research focus is in the areas of Digital Systems Testing, Fault-Tolerant Computing, and Sensor Networks. Professor Saluja has authored and co-authored over 300 technical papers that have appeared in conference proceedings and journals. Professor Saluja is a member of Eta Kappa Nu, Tau Beta Pi, a fellow of the JSPS and a Fellow of the IEEE.

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 1933, 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 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 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.