IEEE the 10th Asian Test Symposium (ATS 2001), pp. 193-198, Nov. 2001.

A DFT Method for Core-Based Systems-on-a-Chip based on Consecutive Testability

Tomokazu Yoneda and Hideo Fujiwara

Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-0101, Japan
{tomoka-y, fujiwara}@is.aist-nara.ac.jp

Abstract

This paper introduces a new concept of testability of core-based systems-on-a-chip (SoCs) called consecutive testability and proposes a design-for-testability (DFT) method for making a given SoC consecutively testable based on integer programming problem. For a consecutively testable SoC, testing can be performed as follows. Test patterns of a core are propagated to the core inputs from the SoC inputs consecutively at speed of system clock. Similarly the test responses are propagated to the SoC outputs from the core outputs consecutively at speed of system clock. The propagation of test patterns and responses is achieved by using the consecutive transparency properties of surrounding cores and interconnects between cores. All interconnects can be tested in a similar fashion. Therefore, the method can test not only logic faults such as stuck-at faults, but also timing faults such as delay faults that require consecutive application of test patterns at speed of system clock.

keywords: *consecutive testability, consecutive transparency, test access mechanism, core-based systems-on-a-chip, design for testability*

1 Introduction

A fundamental change has taken place in the way digital systems are designed. It has become possible to design an entire system, containing millions of transistors, on a single chip. In order to cope with the growing complexity of such modern systems, designers often use pre-designed, reusable megacells knows as cores. Core-based systems-on-a-chip (SoC) design strategies help companies significantly reduce the time-to-market and design cost for their new products.

However, SoCs are difficult to test after fabrication[1]. In order to make SoC testable, the following three conditions have to be satisfied. (1)*There exist test pattern source (TPS) and test response sink (TRS) for each core.* The TPS generates the test patterns for the embedded core, and the TRS compares the test responses to the expected responses. (2)*There exists test access mechanism for each core.* The test access mechanism propagates test patterns and responses. It can be used for on-chip propagation of test patterns from a TPS to the core-under-test, and for on-chip propagation of test responses from the core-under-test to a TRS. (3)*Interconnects that exist between cores are testable.* In this paper, we assume that TPS and TRS are implemented off-chip (i.e., embedded cores are tested by using external primary inputs (PIs) / primary outputs (POs)). Under this assumption, a major difficulty concerns accessibility of embedded cores. Several techniques of design for testability (DFT) have been proposed. There are three main approaches to achieve accessibility of embedded cores. The first approach is based on test bus architectures[2, 3]. The second approach is based on boundary scan architectures[4, 5]. The third approach uses transparency[6, 7] or bypass mode[8] for embedded cores to propagate test patterns and responses.

Under the design environment for SoCs, it is also important to test timing faults such as delay faults as well as logic faults such as stuck-at faults. For that reason, it is necessary to be able to apply test patterns consecutively by using normal system clock and observe the responses consecutively by using normal system clock. We call such a test access *consecutive test access*. Although test bus approach is consecutively test accessible, it is difficult to test interconnects. On the other hand, boundary scan, transparency, and bypass mode approaches are able to test interconnects, they are not consecutively test accessible.

In this paper, we propose new concepts,*consecutive transparency* for cores and *consecutive testability* for SoCs, as the properties that enable both above-mentioned consecutive test access and test for interconnects. Then we present a DFT method to make a given SoC consecutively testable. Consecutive transparency of a core guarantees that any input sequence applied to an input port of the core can be propagated to some output ports of the core, and any output sequence that appears at an output port of the core can be propagated from some input ports of the core *consecutively* at speed of system clock. Consecutive testability of an SoC guarantees that, for each core (for each interconnect), by using consecutive transparencies of other cores, test patterns can be fed into the core (the interconnect, respectively) from PIs and the responses can be propagated to POs consecutively at speed of system clocks. Therefore, consecutive testability guarantees high quality of test since any test sequence for a core can be applied to the core from PIs and any response sequence can be observed at POs consecutively at speed of system clock (*at-speed test*).

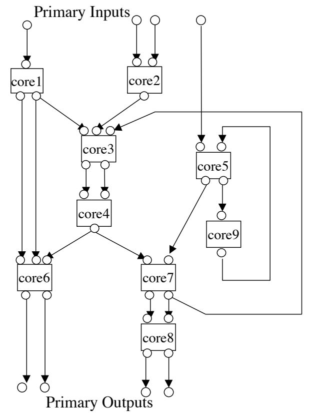

Figure 1. Core-Based Systems-on-a-Chip

In this paper, we assume that TPS and TRS are implemented off-chip. However it is easy to extend the method so that TPS and TRS implemented on-chip by Built-In-Self-Test can be dealt with.

This paper is organized as follows. In section 2, we introduce an SoC model. In section 3, we introduce the consecutive transparency, the consecutive testability, and present a new test methodology for testing SoCs. In section 4, we present a DFT method for consecutive testability. Section 5 concludes this paper.

2 SoC Modeling

An SoC consists of *hardware elements* and *interconnects*. A hardware element is a primary input (PI), a primary output (PO), or a core. For the sake of uniformity, user-defined logic can be considered as another core. Each individual core is testable and a precomputed test set is available for each core which, if applied to the core, will result in a very high fault coverage. We introduce *ports* of each hardware element as interface points in a natural fashion: signals enter into a hardware element through its *input ports*, and exit through its *output ports*. For convenience, we regard a PI as an output port and a PO as an input port. An interconnect connects an output port with an input port. Any number of interconnects can connect to the same output port (i.e., fanout is allowed), but only one interconnect can connect to the same input port. It is not necessary that interconnects are of the same bit width.

3 A Test Methodology for SoCs Based on Consecutive Testability

We present a new test methodology for SoCs based on *consecutive testability*. Figure 2 illustrates a consecutively testable SoC and the consecutive test access. A control signal is provided for each core by a test controller (either off-chip or on-chip). Each control signal of a core determines the current test mode of the core called a *configuration*. In Figure 2, a configuration of each core is determined and *consecutive transparencies* of shaded ports are

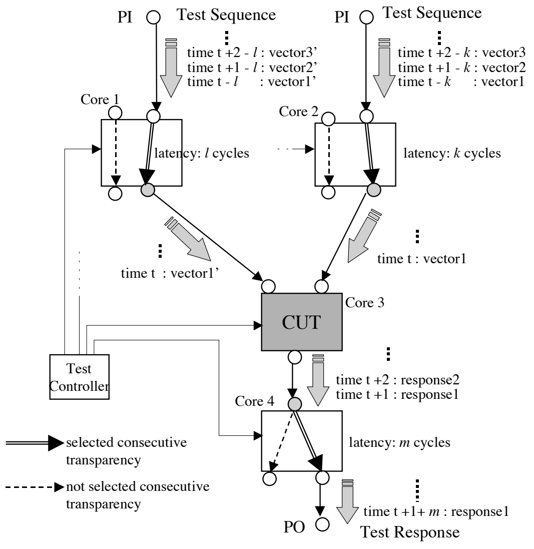

Figure 2. Consecutive Test Access

realized. Consecutive transparency of an input port guarantees that any input sequence applied to the input port can propagate to some output ports consecutively at speed of system clock , and consecutive transparency of an output port guarantees that any output sequence that appears at the output port can propagate from some input ports consecutively at speed of system clock. Consecutive testability of an SoC guarantees that, for each core (interconnect) in the SoC, by selecting *configurations* of other cores, test patterns can be consecutively fed into the core (interconnect) from PIs and the responses can be consecutively propagated to POs through *consecutive transparencies* of other cores and interconnects. We define the *consecutive transparency* of a core and the *consecutive testability* of an SoC in the following subsections.

3.1 Consecutive Transparency

In this subsection, we define a new testability of a core called *consecutive transparency* as follows.

Definition 1: Let $I(i)$ be the ith bit of an input port I, and $O(j)$ be the jth bit of an output port O. Suppose that there exists a configuration of a core which can realize a path P between $I(i)$ and $O(j)$. P is called a *consecutively transparent path* if any input sequence applied to $I(i)$ can be consecutively observed at $O(j)$ after some latency, and then $I(i)$ and $O(j)$ are said to be *consecutively transparent*. Moreover, a core is called to be *consecutively transparent* if, for each port of the core, there exists a configuration that can make all bits of the port consecutively transparent. ■

Figure 3 illustrates various configurations of a consecutively transparent core. A consecutively transparent core has generally several *configurations*, and each configuration can be identified by an ID number. By selecting a configuration of a core, *consecutively transparent paths* of an I/O

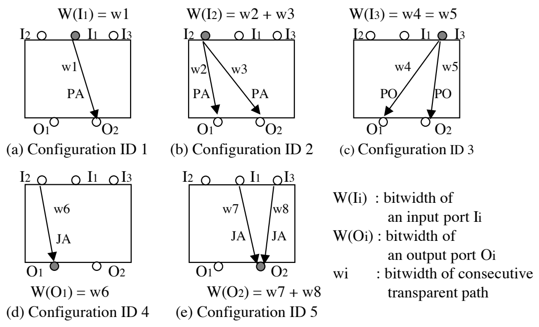

Figure 3. Various Configurations of a Consecutively Transparent Core

port are realized and the I/O port can be made consecutively transparent. For each configuration, all consecutively transparent paths between an input port and an output port are represented as one consecutively transparent path.

We classify consecutively transparent paths into three types, *PA* (Propagation AND), *PO* (Propagation OR), and *JA* (Justification AND). *PA* and *PO* are types of consecutively transparent paths for input ports to propagate test responses. *JA* is a type of consecutively transparent paths for output ports to justify test sequences. Figure 3(a) illustrates type *PA* such that any input sequence applied to an input port I_1 propagates to only one output port O_2. Figure 3(b) illustrates type *PA* such that any input sequence applied to an input port I_2 propagates to two output ports(O_1 and O_2), where any input sequence of bit width $W(I_2)$ is bit-sliced ($W(I_2) = w2 + w3$) and observed at two output ports (O_1 and O_2). Figure 3(c) illustrates type *PO* such that any input sequence applied to I_3 propagates to two output ports (O_1 and O_2), where any input sequence of bit width $W(I_3)$ is fanouted ($W(I_3) = w4 = w5$) and observed at two output ports (O_1 and O_2).

We define a *core connectivity graph* $G = (V, E, \lambda)$ to represent an SoC composed of consecutively transparent cores:

- $V = V_{PI} \cup V_{PO} \cup V_{in} \cup V_{out}$ where
  V_{PI} is the set of all PIs of the SoC,
  V_{PO} is the set of all POs of the SoC,
  V_{in} is the set of all input ports of cores in the SoC, and
  V_{out} is the set of all output ports of cores in the SoC.
- $E = E_{core} \cup E_{net}$ where
  $E_{core} = \{(x,y) \in V_{in} \times V_{out} \mid$ input port x is connected to output port y by a consecutively transparent path$\}$, and
  $E_{net} = \{(y,x) \in V_{out} \times V_{in} \mid$ output port y is connected to input port x by an interconnect$\}$.
- Labeling function $\lambda : E \to 2^{C \times I \times T \times W}$ where
  C is the set of all cores in the SoC,
  I is the set of all ID numbers of configurations,
  $T = \{JA, JO, PA, PO \mid$ types of consecutively transparent path (JO is for fanouted interconnects) $\}$, and
  W is the set of all bit widths of $e \in E$.
  Especially for $e \in E_{net}$,
  $\lambda(e) = \{\{\phi, \phi, JO,$ bit width of $e\}, \{\phi, \phi, PO,$ bit width of $e\}\}$

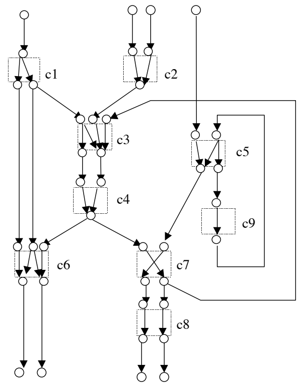

Figure 4. Core Connectivity Graph

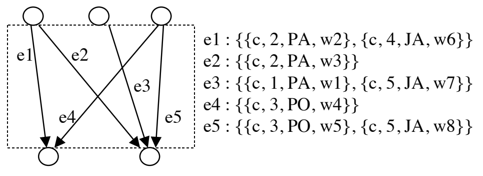
e1
e2
e3
e4
e5
e1 : {{c, 2, PA, w2}, {c, 4, JA, w6}}
e2 : {{c, 2, PA, w3}}
e3 : {{c, 1, PA, w1}, {c, 5, JA, w7}}
e4 : {{c, 3, PO, w4}}
e5 : {{c, 3, PO, w5}, {c, 5, JA, w8}}

Figure 5. Label by λ

Figure 4 illustrates a core connectivity graph G which corresponds to the SoC of Figure 1 and Figure 5 illustrates edges labeled by λ which correspond to the core of Figure 3.

We refer to a vertex that has no input edge as a *source*, and a vertex that has no output edge as a *sink*. For a core connectivity graph G, selecting a configuration of a core is to leave edges which have labels of the configuration and to remove other edges from the core.

3.2 Consecutive Testability

In this subsection, we introduce a new testability of an SoC called *consecutive testability*. For SoCs, it is important to test timing faults such as delay faults as well as logic faults such as stuck-at faults. For that reason, it is necessary to be able to apply any test sequence from PIs to each core and observe any response sequence at POs consecutively at speed of system clock. Moreover, it is also important to test interconnects between cores thoroughly. We formalize consecutive testability of an SoC as a sufficient condition that satisfies above-mentioned conditions. We first define justification subgraph G_J and propagation subgraph G_P as follows.

Definition 2: Let $G = (V, E, \lambda)$ be a core connectivity graph of an SoC and $G_J = (V_J, E_J, \lambda)$ be an acyclic subgraph of G. For a core $c \in C$, G_J is called a *justification subgraph of c* and c is said to be *consecutively controllable* if G_J satisfies all the following conditions.

1. All input ports of c are *sinks* in G_J and only PIs are *sources* in G_J.
2. For each edge $u \in E_J$, u has a label of either JO or JA.

3. Let $G' = (V', E', \lambda)$ be a subgraph of G obtained by selecting a configuration for each core. For each edge $u \in E_J$,
   (a) u contains all input edges of u in G', and
   (b) u contains only one output edge of u in G' when output edges have labels of JO in G'. ■

Definition 3: Let $G = (V, E, \lambda)$ be a core connectivity graph of an SoC and $G_J = (V_J, E_J, \lambda)$ be an acyclic subgraph of G. For an interconnect $e = (y, x) \in E_{net}$, G_J is called a *justification subgraph of e* and e is said to be *consecutively controllable* if G_J satisfies all the following conditions.

1. Only y is a *sink* in G_J and only PIs are *sources* in G_J.
2. For each edge $u \in E_J$, u has a label of either JO or JA.
3. Let $G' = (V', E', \lambda)$ be a subgraph of G obtained by selecting a configuration for each core. For each edge $u \in E_J$,
   (a) u contains all input edges of u in G', and
   (b) u contains only one output edge of u in G' when output edges have labels of JO in G'. ■

Definition 4: Let $G = (V, E, \lambda)$ be a core connectivity graph of an SoC and $G_P = (V_P, E_P, \lambda)$ be an acyclic subgraph of G. For a vertex $v \in V$, G_P is called *propagation subgraph of v* and v is said to be *consecutively observable* if G_P satisfies all the following conditions.

1. Only POs are *sinks* in G_P and only v is a *source* in G_P.
2. For each edge $u \in E_P$, u has a label of either PO or PA.
3. Let $G' = (V', E', \lambda)$ be a subgraph of G obtained by selecting a configuration for each core. For each edge $u \in E_P$,
   (a) u contains all output edges of u in G' when the output edges have labels of PA, and
   (b) u contains only one output edge of u in G' when the output edges have labels of PO in G'. ■

Then, we define the consecutive testability of an SoC as follows.

Definition 5: Let $G = (V, E, \lambda)$ be a core connectivity graph of an SoC. An SoC is said to be *consecutively testable* if the SoC satisfies the following two conditions.

1. For each output port $v \in V_{out}$ of each core $c \in C$, there exist one *justification subgraph* G_J of c and one *propagation subgraph* G_P of v where G_J and G_P are disjoint.
2. For each interconnect $e = (y, x) \in E_{net}$, there exist one *justification subgraph* G_J of y and one *propagation subgraph* G_P of x where G_J and G_P are disjoint. ■

4 DFT for Consecutive Testability

This section presents a method for *design-for-testability* (DFT) that makes a given SoC consecutively testable. We assume that each individual core is testable and a precomputed test set is available for each core which, if applied to the core, will result in a very high fault coverage, and the internal design of the cores cannot be modified by DFT due to IP (Intellectual Property) protection. Additionally, we assume that all cores are consecutively transparent and control signals for configurations can be controlled independently of normal operations. Even if a core is not consecutively transparent, we can make the core consecutively transparent by adding bypass routes outside the core. In the rest of this paper, we consider the DFT under such assumptions.

4.1 Problem Formulation

Each core (each interconnect) in a consecutively testable SoC is consecutively controllable and consecutively observable. In other words, for each output port of each core (for each interconnect), a core connectivity graph G that represents a consecutively testable SoC has one justification subgraph G_J and one propagation subgraph G_P where G_J and $G_P$are disjoint. When a given SoC does not have such disjoint subgraphs, paths from PIs and paths to POs are added using *test MUXs* (multiplexers) in the proposed DFT.

Definition 6: The DFT for the consecutive testability is formalized as the following optimization problem.
Input: An SoC (a core connectivity graph)
Output: A consecutively testable SoC
Optimization: Minimizing hardware overhead (i.e., total bit width of added MUXs) ■

4.2 DFT algorithm

We propose a DFT algorithm for consecutive testability. The algorithm consists of the following four stages.

Stage 1: Augment a given SoC so that all cores are consecutively controllable.
Stage 2: Augment a given SoC so that all cores are consecutively observable.
Stage 3: Augment a given SoC so that all interconnects are consecutively controllable.
Stage 4: Augment a given SoC so that all interconnects are consecutively observable.

Due to limitations of space, we only present a procedure of stage 1. However procedures for the other stages can be presented in a similar fashion.

4.2.1 DFT for Consecutive Controllability of Cores (Stage 1)

The objective of the first stage is to modify a given SoC with minimum hardware overhead so that all cores are consecutively controllable (i.e., all cores have justification subgraphs). The strategy of the algorithm is that, for each core, first it creates *control initial graph*, and second it creates *control middle graph*. Then it induces conditions such that each core has a justification subgraph (each core is consecutively controllable), and formalizes the DFT as *integer programming problem*. Justification subgraphs of all cores are determined with minimum hardware overhead by solving the *integer programming problem*.

Step 1: Creation of Control Initial Graph
The *control initial graph* G_{J_c} of a core $c \in C$ is created from a core connectivity graph G as follows.

1. Remove the edges which have labels of c and let the vertices which correspond to the input ports of c be *sinks*.

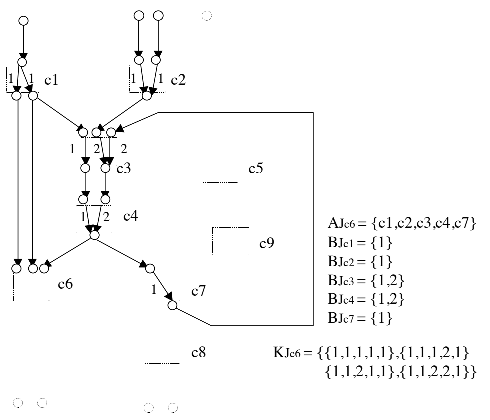

Figure 6. Control Initial Graph $G_{J_{c6}}$

2. Remove the edges which have labels of neither *JA* nor *JO*.
3. We define the control initial graph G_{J_c} as the set of vertices and edges reachable to *sinks*.

Figure 6 illustrates a control initial graph $G_{J_{c6}}$. Each edge in $G_{J_{c6}}$ has a label of either *JO* or *JA* and the number beside $e \in E_{core}$ represents a label of configuration ID.

Let A_{J_c} be the set of cores that exist in G_{J_c}. Here, a core $c' \in C$ that exists in G_{J_c} means that there exists more than one edge which has a label of c' in G_{J_c}. For each $a \in A_{J_c}$, let B_{J_a} be the set of all configuration IDs of a. We define K_{J_c} as the following equation.

$$\begin{aligned} K_{J_c} &= \prod_{a \in A_{J_c}} B_{J_a} \\ &= B_{J_{a1}} \times B_{J_{a2}} \times B_{J_{a3}} \times ..... \end{aligned}$$

A control initial graph G_{J_c} contains several configurations for each core $a \in A_{J_c}$, and consecutive transparency of each core $a \in A_{J_c}$ is not realized.

Step 2: Creation of Control Middle Graph

For each $k \in K_{J_c}$, the *control middle graph* $G_{J_{c,k}}$ is created from a control initial graph G_{J_c} as follows.

1. For each $a \in A_{J_c}$, select a configuration that corresponds to k.
2. We define the control middle graph $G_{J_{c,k}}$ as the set of vertices and edges reachable to *sinks*.

Figure 7 illustrates a control middle graph $G_{J_{c6,k1}}$. *JO* and *JA* beside $e \in E$ represent types of consecutively transparent path e. A control middle graph $G_{J_{c,k}}$ contains only one configuration for each core $a \in A_{J_c}$, and consecutive transparency of each core $a \in A_{J_c}$ is realized.

For $G_{J_{c,k}}$, we define $Q_{J_{c,k}}$ as the set that, for each $q \in Q_{J_{c,k}}$, q satisfies the following conditions.

1. q is a *source* and not an element of V_{PI}.
2. q has more than two output edges which have labels of *JO*.
3. There exist cycles which contain q.

$G_{J_{c,k}}$ is not a *justification subgraph* G_J because of vertices in $Q_{J_{c,k}}$. Thus, $G_{J_{c,k}}$ is a justification subgraph if the set $Q_{J_{c,k}}$ is empty, and we can make c consecutively controllable by

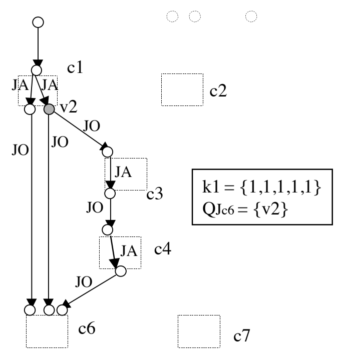

Figure 7. Control Middle Graph $G_{J_{c6,k1}}$

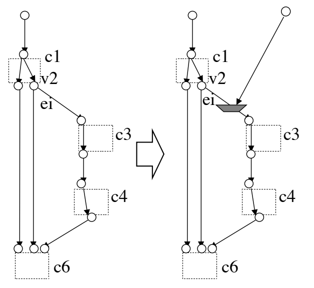

Figure 8. Insertion of a MUX for Consecutive Controllability

selecting configurations of $a \in A_{J_c}$ which correspond to k.

Step 3: Integer Programming Formulation

Let Y_c be a variable that represents consecutive controllability for $c \in C$, and let x_{e_i} be a variable defined as follows.

$$x_{e_i} = \begin{cases} 1 & \text{if a test MUX is inserted to } e_i \in E_{net} \\ 0 & \text{otherwise} \end{cases} \qquad (1)$$

Figure 8 illustrates that a test MUX is inserted to e_i. We can formalize the DFT for consecutive controllability of cores as the following *integer programming problem*:

Minimize:

$$\sum_{e_i \in E_{net}} x_{e_i} \cdot bitwidth(e_i) \qquad (2)$$

Subject to:

$$Y_c \geq 1 \quad for\ all\ c \in C \qquad (3)$$

$$x_{e_i} \in \{0,1\}\ \forall e_i \in E_{net} \qquad (4)$$

Let $Y_{c,k}$ be a variable that represents consecutive controllability for a control middle graph $G_{J_{c,k}}$, and let $Y^q_{c,k}$ be a variable that represents consecutive controllability for a vertex $q \in Q_{J_{c,k}}$. Y_c is defined as follows.

$$Y_c = \sum_{k \in K_{J_c}} Y_{c,k} \qquad (5)$$

$$Y_{c,k} = \begin{cases} \prod_{q \in Q_{J_{c,k}}} Y^q_{c,k} & (Q_{J_{c,k}} \neq \phi) \\ 1 & (Q_{J_{c,k}} = \phi) \end{cases} \qquad (6)$$

Equation(5) means that core c is consecutively controllable if c is consecutively controllable for more than one control initial graph $G_{J_{c,k}}$ by selecting configurations which correspond to $k \in K_{J_c}$. Equation(6) means that $G_{J_{c,k}}$ is a justification subgraph if $Q_{J_{c,k}}$ is empty or $G_{J_{c,k}}$ can be made consecutively controllable for all vertices in $Q_{J_{c,k}}$.

$Y^q_{c,k}$ is defined with x_{e_i} as follows.

Case 1: q is a *source* of $G_{J_{c,k}}$ and not an element of V_{PI}.

Let S be the set of all simple paths from q to *sinks* in $G_{J_{c,k}}$. In order to make $G_{J_{c,k}}$ consecutively controllable for q, it is sufficient that more than one MUX is inserted to each $s \in S$ and paths from PIs are added. Let E_s be the set of all edges which are elements of E_{net} in s. Insertion of more than one MUX to a simple path s means that m_s represented by the following equation is more than 1.

$$m_s = \sum_{e_i \in E_s} x_{e_i} \tag{7}$$

With this m_s, $Y^q_{c,k}$ is defined as follows.

$$Y^q_{c,k} = \prod_{s \in S} m_s \tag{8}$$

Case 2: q has more than two output edges which have labels of JO.

Let R be the set of all output edges of q in $G_{J_{c,k}}$. For each $r \in R$, let S_r be the set of all simple paths that contain r from q to *sinks*. In order to make $G_{J_{c,k}}$ consecutively controllable for q, it is sufficient that the following condition is satisfied for more than one element $r \in R$. The condition is that ,for each $r' \in R - \{r\}$, more than one MUX is inserted to each $s \in S_{r'}$ and paths from PIs are added. Therefore, $Y^q_{c,k}$ is defined as follows with m_s represented by equation(7).

$$Y^q_{c,k} = \sum_{r \in R} \left(\prod_{r' \in R - \{r\}} \Big(\prod_{s \in S_{r'}} m_s \Big) \right) \tag{9}$$

Case 3: There exist cycles which contain q.

Let S be the set of all cycles that contain q in $G_{J_{c,k}}$. In order to make $G_{J_{c,k}}$ consecutively controllable for q, it is sufficient that more than one MUX is inserted to each $s \in S$ and paths from PIs are added. Therefore, $Y^q_{c,k}$ is defined as follows with m_s represented by equation(7).

$$Y^q_{c,k} = \prod_{s \in S} m_s \tag{10}$$

Test MUXs are inserted to the edges obtained by solving the *integer programming problem* with equations (2), (3), and (4). Thus, justification subgraphs of all cores can be determined with minimum hardware overhead.

5 Conclusions

In this paper, we introduced a new concepts of testability called *consecutive transparency* and *consecutive testability*. For a consecutively testable SoC, testing can be performed as follows. Test patterns of a core are propagated to the core inputs from the SoC inputs consecutively at speed of system clock. Similarly the test responses are propagated to the SoC outputs from the core outputs consecutively at speed of system clock. The propagation of test patterns and responses is achieved by using the consecutive transparency properties of surrounding cores and interconnects between cores. All interconnects can be tested in a similar fashion. Therefore, the method can test not only logic faults such as stuck-at faults, but also timing faults such as delay faults that require consecutive application of test patterns at speed of system clock. We also proposed a design-for-testability (DFT) method for making a given SoC consecutively testable based on integer programming problem.

One of our future works is to propose a DFT method for making cores consecutively transparent. In this paper, we assumed that TPS and TRS are implemented off-chip, that is, external testing only. However, we also have to consider Built-In-Self-Testing (BIST). Hence, another future work is to extend the proposed SoC model to the SoC model with on-chip TPS and TRS and BISTed cores.

Acknowledgments

This work was sponsored in part by NEDO (New Energy and Industrial Technology Development Organization) through the contract with STARC (Semiconductor Technology Academic Research Center) and supported in part by Foundation of Nara Institute of Science and Technology under the Grant for Activity of Education and Research. Authors would like to thank Toshimitsu Masuzawa (Osaka University), Michiko Inoue and Satoshi Ohtake (Nara Institute of Science and Technology) for their valuable discussion.

References

[1] Y.Zorian, E.J.Marinissen and S.Dey, "Testing embedded-core based system chips," Proc. International Test Conference, pp.130-143, Oct. 1998.

[2] S.Bhatia, T.Gheewala and P.Varma, "A unifying methodology for intellectual property and custom logic testing," Proc. International Test Conference, pp.639-648, Oct. 1996.

[3] T.Ono, K.Wakui, H.Hikima, Y.Nakamura and M.Yoshida, "Integrated and automated design-for-testability implementation for cell-based ICs," Proc. Asian Test Symposium, pp.122-125, Nov. 1997.

[4] N.A.Touba and B.Pouya, "Testing embedded cores using partial isolation rings," Proc. VLSI Test Symposium, pp.10-16, May 1997.

[5] L.Whetsel, "An IEEE 1149.1 based test access architecture for ICs with embedded cores, " Proc. International Test Conference, pp.69-78, Nov. 1997.

[6] I.Ghosh, N.K.Jha and S.Dey, "A low overhead design for testability and test generation technique for core-based systems-on-a-chip," IEEE Trans. on CAD, vol.18, no.11, pp.1661, Nov. 1999.

[7] I. Ghosh, S. Dey, and N.K. Jha, " A fast and low cost testing technique for core-based system-on-chip," Proc. 35th Design Automation Conference, pp.542-547, June 1998.

[8] M.Nourani and C.A.Papachristou, "Structural fault testing of embedded cores using pipelining," Journal of Electronic Testing:Theory and Applications 15, pp.129-144 1999.