IEEE 2001 Workshop on RTL ATPG & DFT, pp. 28-37, Nov. 2001.

Design for Consecutive Testability of Systems-on-a-Chip with Built-In Self Testable Cores

Tomokazu Yoneda and Hideo Fujiwara
Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-0101, Japan
Tel : +81-743-72-5226 Fax : +81-743-72-5229
{tomoka-y, fujiwara}@is.aist-nara.ac.jp

Abstract

This paper introduces a new concept of testability called consecutive testability and proposes a design-for-testability 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 test pattern sources (implemented either off-chip or on-chip) consecutively at speed of system clock. Similarly the test responses are propagated to test response sinks (implemented either off-chip or on-chip) 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, it is possible to 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, systems-on-a-chip, design for testability, built-in self test*

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 observes the test responses. TPS as well as TRS can be implemented either off-chip or on-chip. (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*.

A major difficulty to make SoC testable 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, 4]. The second approach is based on *boundary scan architectures*[5, 6]. The third approach uses *transparency*[8, 9, 10] or *bypass mode*[7] 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 capable of applying test patterns and observing the responses consecutively at speed of 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 [11], assuming that TPS and TRS are implemented only off-chip (i.e., embedded cores are tested by using external automatic test equipment), we proposed a new testability of SoCs called *consecutive testability*. A consecutively testable SoC consists of consecutively transparent cores only and can achieve consecutive test access to all cores and all interconnects.

In this paper, we consider SoCs that include *BISTed* (Built-In Self Tested) cores and *opaque* cores as well as non-BISTed cores and consecutively transparent cores, and extend the concept of *consecutive testability* of SoCs so that TPS and TRS implemented both on-chip and off-chip can be dealt with. Then, we present a DFT method to make

a given SoC consecutively testable. Consecutive testability of an SoC guarantees that, for each core (for each interconnect), by using consecutive transparencies of other cores and interconnects, test patterns can be fed into the core (the interconnect, respectively) from TPS and the responses can be propagated to TRS consecutively at speed of system clock. Therefore, consecutively testable SoCs can achieve high quality of test since any test sequence for a core can be applied to the core from TPS and any response sequence can be observed at TRS consecutively at speed of system clock.

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 graph model for an SoC. In section 5, we present a DFT method for consecutive testability. Section 6 concludes this paper.

2 Systems-on-a-Chip

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 by either external test or built-in self test. In case a core is testable by external test, a precomputed test set is available for the 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 Systems-on-a-Chip 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 to Core 3. 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*. The types of configurations are consecutive transparencies and functions as a TPS and a TRS. Core 1 works as a TPS for Core 3. Core 2 realizes a consecutive transparency of shaded output port and Core 4 realizes a consecutive transparency of shaded input port. Consecutive

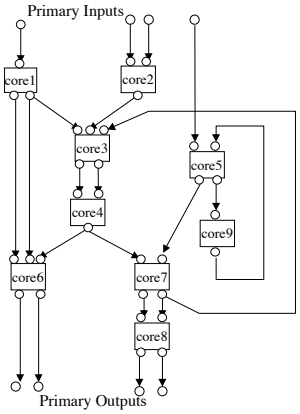

Figure 1. Systems-on-a-Chip

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 (for each interconnect) in the SoC, by selecting configurations of other cores, any test sequence can be consecutively fed into the core (the interconnect, respectively) from TPSs and any response sequence can be consecutively propagated to TRSs 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 of a Core

Definition 1: Consecutive transparency of a core
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 port are realized and the I/O port can be made consecutively transparent. For each configuration, all consecutively trans-

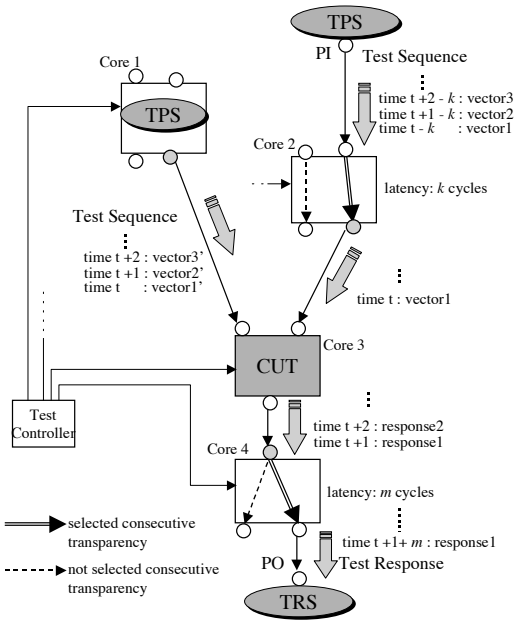

Figure 2. Consecutive Test Access

parent 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).

3.2 Test Pattern Source and Test Response Sink

The test pattern source (TPS) generates test patterns for cores and interconnects, and the test response sink (TRS) observe the test responses. TPS and TRS can be implemented either off-chip or on-chip. In this paper, we classify TPS and TRS into the following three types (Figure 4).

1. S_{BIST}
   S_{BIST} is a type of TPS and TRS implemented inside of a core (i.e.,on-chip) and used for testing the core itself (Figure 4(c)). A core which has this type of TPS and TRS can be self-testable.

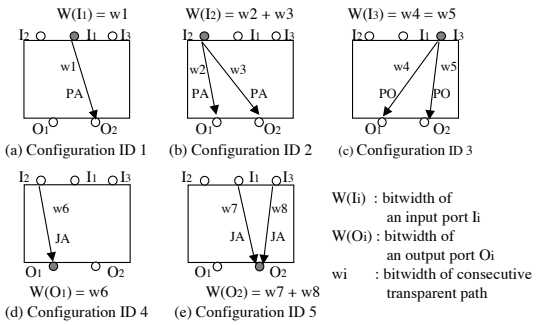

Figure 3. Various Configurations of a Consecutively Transparent Core

2. S_{off}
   S_{off} is a type of TPS and TRS implemented off-chip by external automatic test equipment (Figure 4(a)). TPS of type S_{off} can generate any test sequence of any length, and TRS of type S_{off} can observe any response sequence of any length consecutively at speed of system clock.

3. S_{on}
   S_{on} is a type of TPS and TRS implemented inside of a core (i.e.,on-chip) and used for testing other cores (Figure 4(b)). Since TPS and TRS of type S_{on} are implemented on-chip, memory spaces for them are limited. Therefore, TPS and TRS of type S_{on} cannot deal with arbitrary long sequences like TPS and TRS of type S_{off}. However, within the limited memory spaces, TPS of type S_{on} can generate any test sequence and TRS of type S_{on} can observe any response sequence consecutively at speed of system clock. A core which can be tested by TPS and TRS of type S_{on} can be also tested by TPS and TRS of type S_{off}. A core which has TPS and TRS of type S_{on} has several configurations (Figure 5), and each configuration can be identified by an ID number. By selecting a configuration of the core, the core can realize functions as a TPS and a TRS.

3.3 Consecutive Testability of a System-on-a-Chip

In this subsection, we introduce a new testability of an SoC called *consecutive testability*. In this paper, we assume that the following informations are given as an SoC.

- Connectivity information between cores
- Test informations of each core

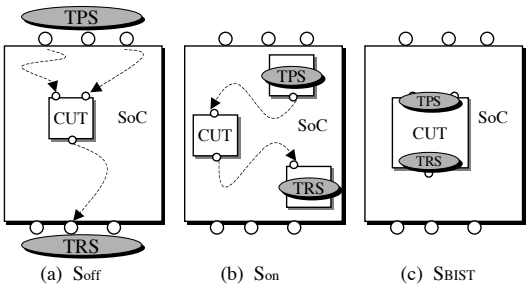

Figure 4. Types of TPS and TRS

- type of TPS/TRS that can test the core (S_{BIST} or S_{off} or S_{on})
- configurations if the core is consecutively transparent
- configurations if the core has TPS/TRS of type S_{on}

The length of a test sequence required to test an interconnect is usually much shorter than that required to test a core. Hence, we assume all interconnect can be tested by TPS/TRS of type S_{on}. In order to test a core, it is necessary to apply test patterns consecutively to all input ports of the core simultaneously. On the other hand, it is not necessary to observe all output ports of the core simultaneously. It is sufficient only to observe one output port at a time. Therefore, we define the consecutive test accessibility of a core and the consecutive test accessibility of an interconnect as follows.

Definition 2: Consecutive test accessibility of a core
A core C is said to be *consecutively test accessible* if the following two conditions are satisfied at the same time for each output port O of C.

1. Any test sequence generated by the TPS required to test C can be applied to all input ports of C consecutively at normal system clock (**consecutive controllability of C for TPS**).
2. Any response sequence appeared at O can be propagated to the TRS required to test C consecutively at normal system clock (**consecutive observability of O for TRS**). ■

Definition 3: Consecutive test accessibility of an interconnect
For an interconnect E that connects an output port O with an input port I, E is said to be *consecutively test accessible* if O and I satisfies the following two conditions at the same time.

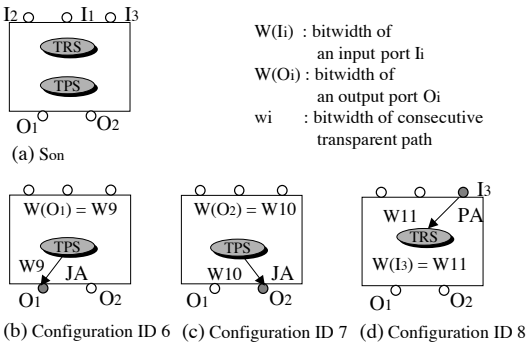

Figure 5. Various Configurations of a Core that has TPS and TRS of type S_{on}

1. Any test sequence generated by the TPS required to test E can be applied to O consecutively at normal system clock (**consecutive controllability of E for TPS**).
2. Any response sequence appeared at I can be propagated to the TRS required to test E consecutively at normal system clock (**consecutive observability of I for TRS**). ■

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

Definition 4: Consecutive testability of an SoC
An SoC is said to be *consecutively testable* if all cores and all interconnects in the SoC are consecutively test accessible. ■

4 Graph Modeling

In this section, we define a core connectivity graph to represent an SoC, and consider the consecutive testability on the graph.

Definition 5: Core connectivity graph
We define a *core connectivity graph* $G = (V, E, \lambda)$ as a following directed graph to represent an SoC.

- $V = V_{PI} \cup V_{PO} \cup V_{in} \cup V_{out}$ 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$\}$,

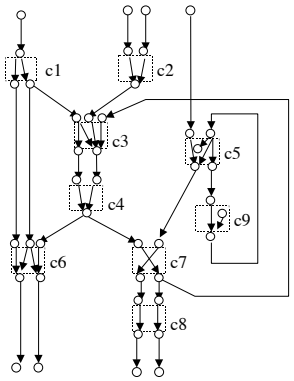

Figure 6. Core Connectivity Graph

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 $\lambda : E \rightarrow 2^{C \times I \times T \times W}$ 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 6 illustrates a core connectivity graph G which corresponds to the SoC of Figure 1. Figure 7 illustrates edges labeled by λ which correspond to the core of Figures 3 and 5.

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.

Then, we define a justification subgraph of a core, a justification subgraph of an interconnect and a propagation subgraph of a port as subgraphs of a core connectivity graph.

Definition 6: Justification subgraph of a core
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* if G_J satisfies all the following conditions.

1. All input ports of c are sinks in G_J and there exists no sink except for all input ports of c in G_J.
2. For each edge $u \in E_J$, u has a label of either JO or JA.

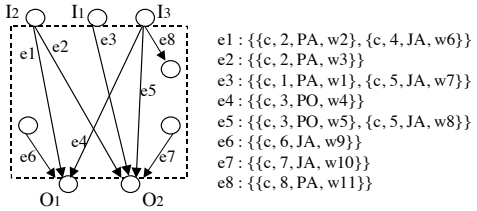

Figure 7. Label by λ

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'. ■

Lemma 1: Let V_S be the set of all source vertices in G_J of core c. Then c is consecutively controllable for V_S
proof: By Definition 5 and condition 2 of Definition 6, all edges in G_J can be used to apply test patterns consecutively at speed of system clock since each edge in G_J represents either a consecutively transparent path or an interconnect, and has a label of either JO or JA. By condition 1 of Definition 6, there exist simple paths from more than one element in V_S to each input port of c. By condition 3 of Definition 6, all edges in the same core have the same ID number of configuration since only one configuration is selected for each core. Let v be the vertex in V_{out} (i.e., v is an output port of a core). If a configuration to realize a consecutive transparency of v is selected, all consecutively transparent paths for v exist in G_J (condition 3(a) of Definition 6). If a configuration to realize a consecutive transparency of v is not selected, v is a source vertex in G_J. By condition 3(b) of Definition 6, it is possible to apply any test sequence for all simple paths at the same time.

Therefore, we can see that any test sequence generated at V_S can be applied to all input ports of c along all simple paths in G_J consecutively at speed of system clock. Hence, the Lemma is proved. ■

Definition 7: Justification subgraph of an interconnect
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* if G_J satisfies all the following conditions.

1. Only y is a sink 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'. ■

Lemma 2: Let V_S be the set of all source vertices in G_J of interconnect e. Then e is consecutively controllable for V_S
proof: The Lemma is proved similarly to Lemma 1. ■

Definition 8: Propagation subgraph of a port
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 if G_P satisfies all the following conditions.

1. 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 more than one output edge of u in G' when the output edges have labels of PO in G'.

■

Lemma 3: Let V_E be the set of all sink vertices in G_P of vertex v. Then v is consecutively observable for V_E
proof: By Definition 8 and condition 2 of Definition 6, all edges in G_P can be used to propagate test responses consecutively at speed of system clock since each edge in G_P represents either a consecutively transparent path or an interconnect, and has a label of either PO or PA. By condition 1 of Definition 8, there exist simple paths from v to each element in V_E. By condition 3 of Definition 8, all edges in the same core have the same ID number of configuration since only one configuration is selected for each core. Let v' be the vertex in V_{in} (i.e., v' is an input port of a core). If a configuration to realize a consecutive transparency of v' is selected and the consecutively transparent paths for v' are type PA, all consecutively transparent paths for v' exist in G_P (condition 3(a) of Definition 8). If a configuration to realize a consecutive transparency of v' is selected and the consecutively transparent paths for v' are type PO, there exist at least one consecutively transparent path for v' in G_P (condition 3(b) of Definition 8). If a configuration to realize a consecutive transparency of v' is not selected, v' is a sink vertex in G_P.

Therefor, we conclude that any response sequence appeared at v can be propagate to V_E along all simple paths in G_P consecutively at speed of system clock. Hence, the Lemma is proved. ■

Theorem 1: 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 and satisfy the following conditions.
   - if TPS/TRS type required to test c is S_{BIST}
     $G_J = G_P = \phi$
   - if TPS/TRS type required to test c is S_{off}
     $V_S \subseteq V_{PI}$, $V_E \subseteq V_{PO}$
   - if TPS/TRS type required to test c is S_{on}
     $V_S \subseteq (V_{PI} \cup V_{source})$, $V_E \subseteq (V_{PO} \cup V_{sink})$
2. For each interconnect $e = (y, x) \in E_{net}$, there exist one *justification subgraph* G_J of e and one *propagation subgraph* G_P of x where G_J and G_P are disjoint and satisfy the following conditions.
   - $V_S \subseteq (V_{PI} \cup V_{source})$, $V_E \subseteq (V_{PO} \cup V_{sink})$

proof: The Theorem is proved by Definitions 2, 3, 4 and Lemmas 1, 2, 3. ■

5 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 by either external test or built-in self test. In case a core is testable by external test, a precomputed test set is available for the core which, if applied to the core, will result in a very high fault coverage. Additionally, we assume that the internal design of the cores cannot be modified by DFT due to IP (Intellectual Property) protection and control signals for configurations can be controlled independently of normal operations.

In the rest of this paper, we consider the DFT under such assumptions.

5.1 Problem Formulation

Each core (interconnect) in a consecutively testable SoC is consecutively controllable for the required TPS and consecutively observable for the required TRS. In other words, for each output port v of each core $c \in C$, a core connectivity graph G that represents a consecutively testable SoC has one justification subgraph G_J of c and one propagation subgraph G_P of v where G_J and G_P are disjoint and satisfy the condition 1 of Theorem 1. Similarly, for each interconnect

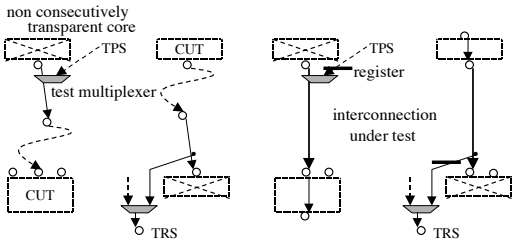

Figure 8. DFT elements

$e = (y,x) \in E_{net}$, there exist one justification subgraph G_J of e and one propagation subgraph G_P of x where G_J and G_P are disjoint and satisfy the condition 2 of Theorem 1.

When a core (an interconnect) in a given SoC is not consecutively controllable for the required TPS, paths from the TPS are added by using *test multiplexers* (MUXs) in the proposed DFT (Figure 8(a)). Similarly, when a core (an interconnect) in a given SoC is not consecutively observable for the required TRS, paths to the TRS are added by using test MUXs (Figure 8(a)). When an interconnect-under-test connects non-consecutively transparent core, it is necessary to isolate the interconnect from the core in order to make the interconnect consecutively test accessible. This isolation is implemented by using test MUXs and *registers* (Figure 8(b)). Assuming that any SoC includes enough number of TPSs and TRSs to make each core (each interconnect) consecutively controllable and observable, we formalize a DFT for making the SoC consecutively testable as the following optimization problem.

Definition 9: DFT for consecutive testability
Input: An SoC (a core connectivity graph)
Output: A consecutively testable SoC
Optimization: Minimizing hardware overhead (i.e., total bit width of added MUXs and registers) ■

5.2 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 for the required TPS.
Stage 2: Augment a given SoC so that all cores are consecutively observable for the required TRS.
Stage 3: Augment a given SoC so that all interconnects are consecutively controllable for the required TPS.
Stage 4: Augment a given SoC so that all interconnects are consecutively observable for the required TRS.

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.

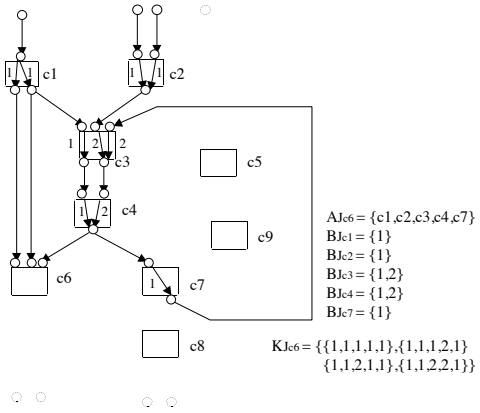

Figure 9. Control Initial Graph $G_{J_{c6}}$

5.2.1 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 for the required TPS (i.e., each core $c \in C$ has a justification subgraph G_J of c where G_J satisfies the condition 1 of Theorem 1). 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 the control middle graph satisfies the Definition 6 and the core is consecutively controllable for the required TPS. Then it formalizes the DFT as *integer programming problem*. All cores are made consecutively controllable 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.
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 9 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

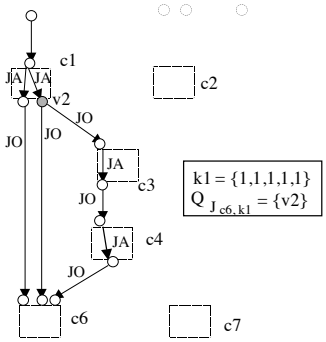

Figure 10. Control Middle Graph $G_{J_{c6,k1}}$

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 10 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 $G_{J_{c,k}}$, let $Q^1_{J_{c,k}}$, $Q^2_{J_{c,k}}$ and $Q^3_{J_{c,k}}$ be the sets of all vertices $q \in G_{J_{c,k}}$ that satisfies the following conditions respectively (Figure 11).

1. $Q^1_{J_{c,k}}$: q is a source.
2. $Q^2_{J_{c,k}}$: q has more than two output edges which have labels of JO.
3. $Q^3_{J_{c,k}}$: There exist cycles which contain q.

We define $Q_{J_{c,k}}$ as follows.

$$Q_{J_{c,k}} = Q^1_{J_{c,k}} \cup Q^2_{J_{c,k}} \cup Q^3_{J_{c,k}}$$

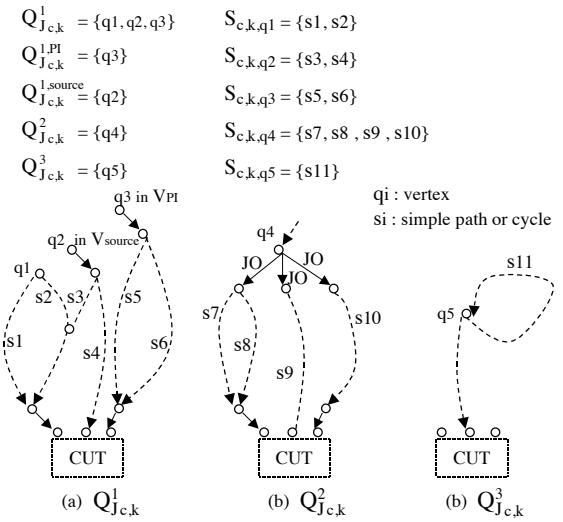

Figure 11. $Q^1_{J_{c,k}}, Q^2_{J_{c,k}}, Q^3_{J_{c,k}}$

$Q^2_{J_{c,k}}$ and $Q^3_{J_{c,k}}$ are the sets of all vertices that do not satisfy the Definition 6. Moreover, we define $Q^{1,PI}_{J_{c,k}}$ and $Q^{1,source}_{J_{c,k}}$ as follows.

$$Q^{1,PI}_{J_{c,k}} = Q^1_{J_{c,k}} \cap V_{PI},\ \ Q^{1,source}_{J_{c,k}} = Q^1_{J_{c,k}} \cap V_{source}$$

Then, let $S_{c,k,q}$ be the set of all simple paths from q in $Q_{J_{c,k}}$ to each sink vertex in $G_{J_{c,k}}$.

Step 3: Integer Programming Formulation

We define the following variables as integer programming variables.

$$y_c = \begin{cases} 1 & \text{core } c \text{ is consecutively controllable for TPS} \\ 0 & \text{otherwise} \end{cases}$$

$$a_{c,k} = \begin{cases} 1 & G_{J_{c,k}} \text{ is consecutively controllable for TPS} \\ 0 & \text{otherwise} \end{cases}$$

$$d_{c,k,q} = \begin{cases} 1 & G_{J_{c,k}} \text{ is consecutively controllable for vertex } q \\ 0 & \text{otherwise} \end{cases}$$

$$z_{q,r} = \begin{cases} 1 & \text{output edge } r \text{ of vertex } q \text{ is consecutively controllable for } q \\ 0 & \text{otherwise} \end{cases}$$

$$m_s = \begin{cases} 1 & \text{if MUX is inserted to simple path } s \\ 0 & \text{otherwise} \end{cases}$$

$$x_e = \begin{cases} 1 & \text{if MUX is inserted to interconnect } e \\ 0 & \text{otherwise} \end{cases}$$

The following integer programming formulation minimizes the test overhead (i.e., total bit width of MUXs)

while making all cores consecutively controllable.

Minimize

$$\sum_{e_i \in E_N} x_{e_i} \cdot width(e_i) \tag{1}$$

Subject to:

1. for each core $c \in C$,

$$y_c \geq 1 \tag{2}$$

2. For each core $c \in C$ which can be tested by either S_{off} or S_{on},

$$\sum_{k \in K_{J_c}} a_{c,k} \geq y_c \tag{3}$$

3. For each element $k \in K_{J_c}$,

$$\sum_{q \in Q_{J_{c,k}}} d_{c,k,q} \geq |Q_{J_{c,k}}| \cdot a_{c,k} \tag{4}$$

$|Q_{J_{c,k}}|$ is a constant value which represents the number of elements in $Q_{J_{c,k}}$.

4. (a) In case TPS type required to test c is S_{off} for each vertex $q \in (Q^1_{J_{c,k}} - Q^{1,PI}_{J_{c,k}})$,

$$\sum_{s \in S_{c,k,q}} m_s \geq |S_{c,k,q}| \cdot d_{c,k,q} \tag{5}$$

   (b) In case TPS type required to test c is S_{on}, for each vertex $q \in (Q^1_{J_{c,k}} - (Q^{1,PI}_{J_{c,k}} \cup Q^{1,source}_{J_{c,k}}))$,

$$\sum_{s \in S_{c,k,q}} m_s \geq |S_{c,k,q}| \cdot d_{c,k,q} \tag{6}$$

5. For each vertex $q \in Q^2_{J_{c,k}}$, let $R_{c,k,q}$ be the set of all output edges of q. For each element $r \in R_{c,k,q}$, let $S^r_{c,k,q}$ be the set of all simple paths between r and all sink vertices in $G_{J_{c,k}}$. Then, for each vertex $q \in Q^2_{J_{c,k}}$,

$$\sum_{r \in R_{c,k,q}} z_{q,r} \geq d_{c,k,q} \tag{7}$$

$$\sum_{s \in (S_{c,k,q} - S^r_{c,k,q})} m_s \geq |S_{c,k,q} - S^r_{c,k,q}| \cdot z_{q,r} \tag{8}$$

6. For each vertex $q \in Q^3_{J_{c,k}}$,

$$\sum_{s \in S_{c,k,q}} m_s \geq |S_{c,k,q}| \cdot d_{c,k,q} \tag{9}$$

7. For each simple path $s \in S_{c,k,q}$,

$$\sum_{e \in E_s} x_e \geq m_s \tag{10}$$

E_s represents the set of all edges which correspond to interconnects in simple path s.

Equation (2) guarantees that all cores are consecutively controllable for the required TPS. If TPS type required to test a core is either S_{off} or S_{on}, more than one $G_{J_{c,k}}$ must be consecutively controllable for the TPS. This is guaranteed by equation (3). In order to make $G_{J_{c,k}}$ consecutively controllable for the TPS, all vertices in $Q_{J_{c,k}}$ must be consecutively controllable for the TPS. This is guaranteed by equation (4). In order to make q in $Q^1_{J_{c,k}}$ consecutively controllable for the TPS, all simple paths in $S_{c,k,q}$ must be inserted MUXs and paths from TPS must be added. However, if q is a vertex that represents the TPS required to test the core, q is already consecutively controllable for the TPS. This is guaranteed by equation (5) and (6). Each vertex q in $Q^2_{J_{c,k}}$ has more than two output edges which have label of JO, and all the edges propagate only the same sequence. In order to make q in $Q^2_{J_{c,k}}$ consecutively controllable for the TPS, each edges of q must propagate any test sequence at the same time. This is guaranteed by equation (7) and (8). In order to make q in $Q^3_{J_{c,k}}$ consecutively controllable for the TPS, all cycles which contain q must be broken by MUXs and paths from TPS must be added. This is guaranteed by equation (9). Equation (10) and guarantees that, let s be a simple path and let E_s be the set of all edges which represent interconnects, insertion of MUX to s means that MUX is inserted to more than one element in E_s.

Test MUXs are inserted to the edges obtained by solving the above integer programming problem (Figure 12), and all cores can be made consecutively controllable with minimum hardware overhead. However, in case TPS type required to test core c is S_{off}, it is necessary to add TPSs of type S_{off} (i.e., add vertices to V_{PI}) if the sum bit width of edges that must be inserted MUXs to make c consecutive controllable is larger than that of available TPSs (i.e.,vertices in $V_{PI} - Q^{1,PI}_{J_{c,k}}$). Similarly, in case TPS type required to test core c is S_{on}, it is necessary to add TPSs of type S_{on} (i.e., add vertices to V_{source}) if the same condition as above is satisfied.

Procedures for the other stages can be presented in a similar fashion.

6 Conclusions

In the paper, we introduced a new testability called consecutive testability. For a consecutively testable SoC, testing can be performed as follows. Test patterns of a core are propagated to all input ports of the core from TPS, and

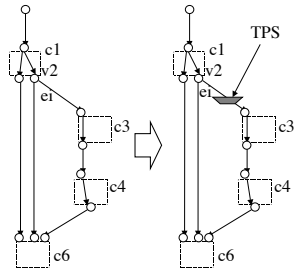

Figure 12. Insertion of a MUX to an edge e_i for Consecutive Controllability

the test responses appeared at an output port of the core are propagated to TRS consecutively at speed of system clock. The propagation of test patterns and responses is achieved by using interconnect and consecutive transparencies of surrounding cores. All interconnects can be tested in a similar fashion. Therefore, it is possible to 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 method for making a given SoC consecutively testable based on integer programming problem. Our future work is to propose a DFT method for making cores consecutively transparent with minimum hardware overhead.

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. 1998 Int. Test Conf., pp.130-143, Oct. 1998.

[2] S.Bhatia, T.Gheewala and P.Varma, "A unifying methodology for intellectual property and custom logic testing," Proc. 1996 Int. Test Conf., 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. 6th Asian Test Symp., pp.122-125, Nov. 1997.

[4] P.Varma and S.Bhatia, "A structured test re-use methodology for core-based system chips," Proc. 1996 Int. Test Conf., pp.294-302, Oct. 1998.

[5] N.A.Touba and B.Pouya, "Testing embedded cores using partial isolation rings," Proc. 15th VLSI Test Symp., pp.10-16, May 1997.

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

[7] 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.

[8] 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.1661-1676, Nov. 1999.

[9] I.Ghosh, S.Dey, and N.K.Jha, " A fast and low cost testing technique for core-based system-chips," IEEE Trans. on CAD, vol.19, no.8, pp.863-877, Aug. 2000.

[10] S.Ravi, G.Lakshminarayana, and N.K.Jha, " Testing of Core-Based Systems-on-a-Chip," IEEE Trans. on CAD, vol.20, no.3, pp.426-439, Mar. 2001.

[11] T.Yoneda and H.Fujiwara, "A DFT method for core-based systems-on-a-chip based on consecutive testability," Proc. 10th Asian Test Symp., Nov. 2001(to appear).