An ILP Formulation for Consecutive Testability of System-on-a-Chip

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 called consecutive testability and proposes a design-for-testability method that makes a given SoC consecutively testable using integer linear programming (ILP). A consecutively testable SoC can achieve consecutive application of arbitrary test sequence at the speed of system clock. The advantage is that it is possible to test not only logic faults but also timing faults that require consecutive application of test patterns at the speed of the system clock.

keywords: *consecutive testability, consecutive transparency, test access mechanism, system-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 by making it possible to design an entire system, containing millions of transistors, on a single chip. In order to cope with the growing complexity of such 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, it is difficult to test SoCs after fabrication[1]. In order to make an SoC testable, the following three conditions have to be satisfied. (1)*There exist a test pattern source (TPS) and a test response sink (TRS) for each core.* The TPS generates the test patterns for the embedded core and the TRS observes the test responses. The TPS as well as the TRS can be implemented either off-chip or on-chip. (2)*There exists a test access mechanism for each core.* The test access mechanism propagates test patterns and test 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 problem to make an SoC testable concerns accessibility of embedded cores. Several design-for-testability (DFT) techniques have been proposed. There are three main approaches to achieve accessibility of embedded cores. The first approach is based on *test bus architectures* by which the cores are isolated from each other in test mode using a dedicated bus [4, 5, 6, 7, 8] or flexible *TESTRAIL* [9] around the cores to propagate test data. Test time reduction is the main objective in the majority of these methods. For example, [7] used an integer linear programming (ILP) formulation to find the best test assignment and to optimize the bandwidth distribution among various test buses minimizing test time. The second approach uses *boundary scan architectures* [10, 11] to isolate the core during test. The third approach uses *transparency* [13, 14, 15] or *bypass* [12] mode for embedded cores to reduce the problem to one of finding paths from TPS to core inputs and from core outputs to TRS.

Under the design environment for SoCs, pre-computed test sets are provided for each core. These test sets may contain functional vectors, scan vectors or ordered test sequences for non-scan sequential circuits. They may be for logic faults such as stuck-at faults or timing faults such as delay faults. Moreover, some cores may be able to be at-speed testable in order to increase the coverage of non-modeled and performance-related defects. For that reason, it is necessary to apply arbitrary test sequence to each core and observe its response sequence from the core consecutively at the speed of the system clock. We call such a test access *consecutive test access*. Although the test bus approach allows consecutively test access for cores, it does not guarantee consecutive test access for interconnects. On the other hand, boundary scan, transparency, and bypass mode approaches allow test access for both cores and interconnect. However, they do not support consecutive test access.

There are two works [16, 17] realizing the consecutive test access for both cores and interconnects. In [16], 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 testability of SoCs called *consecutive testability*. A consecutively testable SoC consists of *consecutively transparent* cores and can achieve consecutive test access to all cores and all interconnects. Consecutive transparency of a core guarantees consecutive propagation of arbitrary test/response sequence from the core input to the core output with some latency. In [17], a synthesis-for-transparency approach was presented to make cores *single-cycle transparent* by embedding multiplexers. This single-cycle transparency is a special case of consecutive transparency of [16] such that the latency of the consecutive transparency is restricted to zero, i.e., single-cycle transparency is the consecutive transparency with zero latency. Therefore, area overhead for making cores consecutively transparent with some latency is generally lower than that for making cores single-cycle transparent (i.e., transparent with zero latency).

In this paper, we consider SoCs that include *BISTed* (Built-In Self Tested) cores and *opaque* cores (which are not consecutively transparent) 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. We also present a DFT method to make a given SoC consecutively testable. Consecutive testability of an SoC guarantees that, for each core and each interconnect, by using interconnects and consecutive transparency properties of surrounding cores, test patterns can be fed into the core (the interconnect, respectively) from TPS and the responses can be propagated to TRS consecutively at the speed of the system clock. Therefore, consecutively testable SoCs can achieve high quality of test since arbitrary test sequence can be applied to a core from TPS and its response sequence can be observed at TRS consecutively at the speed of the system clock.

The rest of this paper is organized as follows. We introduce an SoC model in section 2. In section 3, we introduce the consecutive transparency, the consecutive testability, and present a new test methodology for testing SoCs. We present a graph model for an SoC in section 4. In section 5, we present a DFT method for consecutive testability. The experimental results are discussed in section 6. Finally, section 7 concludes this paper.

2 System-on-a-Chip Modeling

An SoC consists of *cores*, *primary inputs*, *primary outputs* and *interconnects*. For the sake of uniformity, user-defined logic can be considered as another core. Each individual core is testable by either external test or BIST. In case a core is testable by external test, a pre-computed 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 core as interface points in a natural fashion: signals enter into a core through its *input ports*, and exit through its *output ports*. An interconnect connects an output port with an input port, a primary input with an input port, or an output ports with a primary output. 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 System-on-a-Chip Based on Consecutive Testability

We present a new test methodology for SoCs based on *consecutive testability*. Figure 1 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

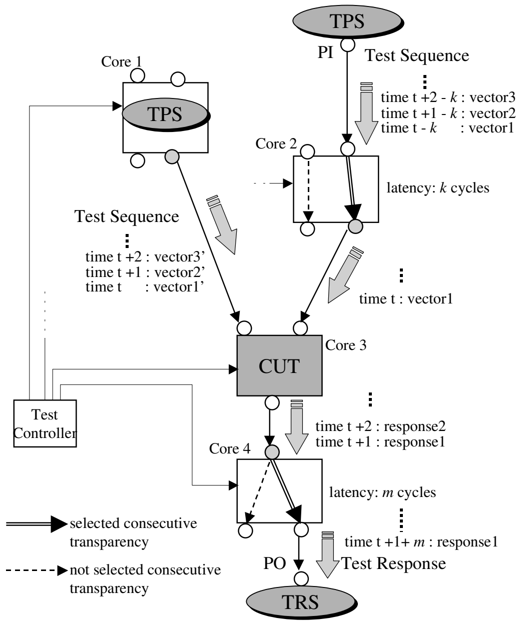


Figure 1. Consecutive Test Access

(either off-chip or on-chip). Each of the control signal 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 transparency of an input port of a core guarantees that arbitrary input sequence applied to the input port can propagate to some output ports of the core consecutively at the speed of system clock. Consecutive transparency of an output port of a core guarantees that arbitrary output sequence that appears at the output port can propagate from some input ports of the core consecutively at the speed of the system clock. Consecutive testability of an SoC guarantees that, for each core (for each interconnect) in the SoC, by selecting configurations of other cores, arbitrary test sequence can be consecutively fed into the core (the interconnect, respectively) from TPSs and its 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 2 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 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* is a type of consecutively transparent path for an input port that propagates part of the bit-width of test responses applied to the input port. On the other hand, *PO* is a type of a consecutively transparent path for an input port that propagates all bit-width of test response applied to the input port. For an input port, all consecutively transparent paths of type *PA* are necessary to make it consecutively transparent. However, only one consecutively transparent path of type *PO* is sufficient to make the input port consecutively transparent. *JA* is a type for a consecutively transparent path of an output port that propagates all or part of bit-width of test sequence which appears at the output port. For an output port, all consecutively transparent paths are necessary to make it consecutively transparent.

Figure 2(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 2(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 2(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). Figure 2(d) illustrates type *JA* such that any output sequence that appears at the output port O_1 is propagated from only one input port I_2. Figure 2(e) illustrates type *JA* such that any output sequence that appears at the output port O_2 is propagated from two input ports (I_1 and I_3), where any output sequence of bit width $W(O_2)$ is constructed by the two input sequences ($W(O_2) = w7 + w8$).

3.2 Test Pattern Source and Test Response Sink

The TPS generates test patterns for cores and interconnects, and the 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 3).

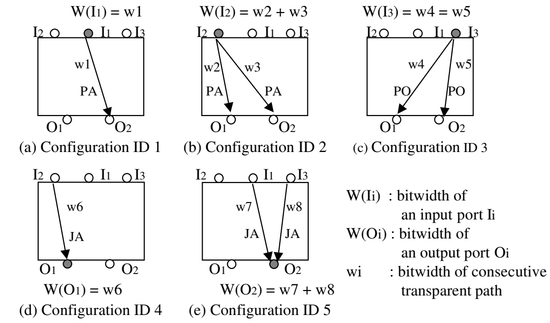


Figure 2. Various Configurations of a Consecutively Transparent Core

1. S_{BIST}
   S_{BIST} is a type for TPS and TRS implemented inside of a core (i.e.,on-chip) and used for testing the core itself (Figure 3(c)). A core with this type of TPS and TRS is self-testable.

2. S_{off}
   S_{off} is a type for TPS and TRS implemented off-chip by external automatic test equipment (ATE)(Figure 3(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 the speed of ATE system clock, which is usually slower than SoC system clock.

3. S_{on}
   S_{on} is a type for TPS and TRS implemented inside of a core (i.e.,on-chip) and used for testing other cores and interconnects (Figure 3(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 the speed of the 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}. Furthermore, a core, which has TPS and TRS of type S_{on}, has several configurations (Figure 4) where each configuration is identified by an ID number. For example, Figure 4(b) shows a configuration of the core identified by ID 6 where the core works as a TPS and generates test patterns from output port O_1.

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

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

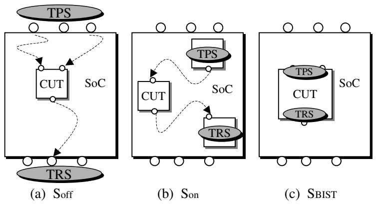


Figure 3. Types of TPS and TRS

- Connectivity information between cores, which is how the cores are connected
- Test informations for each core
  - type of TPS/TRS that can test the core (S_{BIST} or S_{off} or S_{on})
  - configurations where the core is consecutively transparent
  - configurations where 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 to 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 to only 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 TPSs required to test C can be applied to all input ports of C consecutively at the speed of system clock (**consecutive controllability of C with respect to TPSs**).
2. Any response sequence appeared at O can be propagated to TRSs required to test C consecutively at the speed of system clock (**consecutive observability of O with respect to TRSs**). ■

Definition 3: Consecutive test accessibility of an interconnect
For an interconnect E connecting 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.

1. Any test sequence generated by TPSs required to test E can be applied to O consecutively at the speed of system clock (**consecutive controllability of E with respect to TPSs**).
2. Any response sequence appeared at I can be propagated to TRSs required to test E consecutively at the speed of the system clock (**consecutive observability of I with respect to TRSs**). ■

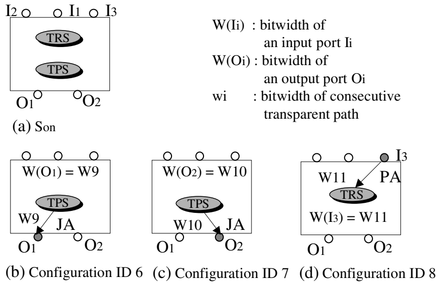


Figure 4. Various Configurations of a Core that has TPS and TRS of type S_{on}

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 core connectivity graph that can represent an SoC, and consider the consecutive testability on the graph. Proofs are omitted due to lack of space.

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

- $V = V_{PI} \cup V_{PO} \cup V_{in} \cup V_{out} \cup V_{source} \cup V_{sink}$ 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.
  V_{source} is the set of all TPSs of type S_{on} in the SoC.
  V_{sink} is the set of all TRSs of type S_{on} 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,

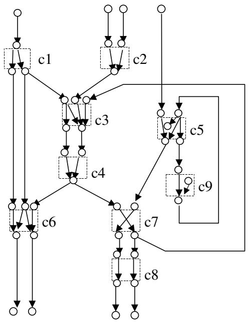


Figure 5. Core Connectivity Graph

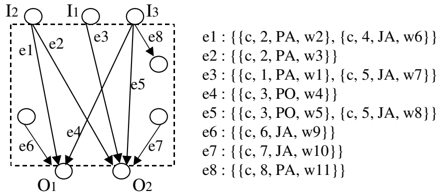


Figure 6. Label by λ

$T = \{JA, JO, PA, PO \mid$ 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 5 illustrates an example of core connectivity graph. Figure 6 illustrates edges labeled by λ of the core that has the five configurations for consecutive transparency (explained in Figure 2) and the three configurations for function as a TPS and a TRS (explained in Figure 4).

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 label 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 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.
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) G_J contains all input edges of u in G', and
   (b) G_J contains only one output edge of u in G' when output edges of u have a label 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 with respect to V_S. ■

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) G_J contains all input edges of u in G', and
   (b) G_J contains only one output edge of u in G' when output edges of u have a label 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 with respect to V_S. ■

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) G_P contains all output edges of u in G' when the output edges have labels of PA, and
   (b) G_P contains more than one output edge of u in G' when output edges of u have a label 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 with respect to V_E. ■

Theorem 1: Let $G = (V, E, \lambda)$ be a core connectivity graph of an SoC. An SoC is *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.

   - $V_S \subseteq (V_{PI} \cup V_{source})$, $V_E \subseteq (V_{PO} \cup V_{sink})$ ■

5 DFT for Consecutive Testability

In this section, we present a *design-for-testability* (DFT) method 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 pre-computed test set is available for the core which, if applied to the core, will result in a very high fault coverage. Additionally, we assume (i) the internal design of the cores cannot be modified by DFT due to IP (Intellectual Property) protection and (ii) 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 with respect to required TPSs and consecutively observable with respect to required TRSs. 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 $e = (y,x) \in E_{net}$, 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 with respect to required TPSs, paths from the TPSs are added by inserting *test multiplexers* (MUXes) in the proposed DFT (Figure 7(a)). Similarly, when a core (an interconnect) in a given SoC is not consecutively observable with respect to required TRSs, paths to the TRSs are added by inserting test MUXes (Figure 7(a)). When an interconnect-under-test is directly connected to an input or output port of a opaque core , it is necessary to isolate the interconnect from the core in order to test timing faults as well as logic faults of the interconnect. This isolation is implemented by adding test MUXes and *registers* (Figure 7(b)). Assuming that any SoC includes enough number of TPSs and TRSs to make each core (each interconnect) consecutively controllable and observable, we formulate a DFT for making the SoC consecutively testable as the following optimization problem.

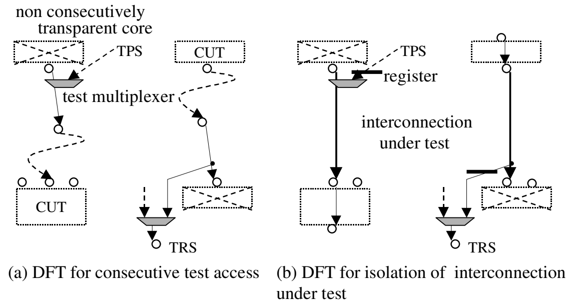


Figure 7. DFT elements

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 MUXes and registers) ■

Lemma 4: Let W_{opaque} be the summation of bit widths of all ports of cores that are not consecutively transparent, excluding ports connected to TPS or TRS. The summation of bit widths of added registers in order to make the SoC consecutively testable is equal to W_{opaque}. ■

5.2 DFT algorithm

In this subsection, we describe a DFT algorithm for consecutive testability using an example system S_1 composed of four consecutively transparent cores (Figure 8). Configurations of each core and consecutively transparent paths of each configuration are also shown in Figure 8. The strategy of the algorithm is that, for each core and interconnect in a given SoC, it first creates *control-observation graph* (**Step 1**), and then, it selects a configuration of each core (**Step 2**). Finally, the algorithm induces constraints such that the given SoC satisfies Theorem 1 and formulates the DFT as *an ILP problem* to minimize hardware overhead (**Step 3**). Any given SoC can be made consecutively testable by solving the ILP problem and adding DFT elements such as MUXes and registers.

Step 1: Creation of Control-Observation Graph

For each element $t \in (C \cup E_{net})$, the *control-observation graph* G_t is created from a core connectivity graph G as follows.

1. (a) if $t \in C$ (t is a core),
   remove the edges with label t and let the vertices corresponding to the input ports of t be sinks and

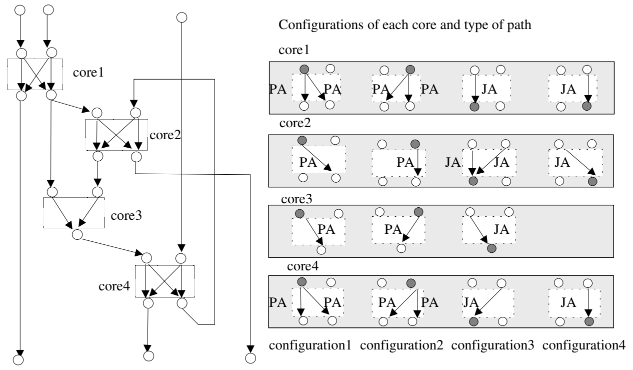


Figure 8. An Example SoC S_1

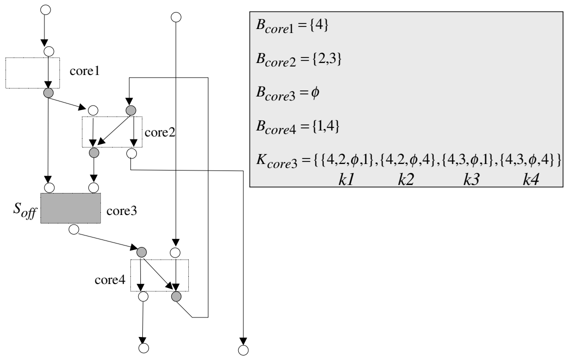


Figure 9. Control-Observation Graph G_{core3}

let the vertices corresponding to the output ports of t be sources.

(b) if $t \in E_{net}$ (t is an interconnect and connects between two vertices),
remove the edge t and let the vertices connected by t be sink and source, respectively.

2. We define G_t as the set of vertices and edges which are
   - reachable to sinks only by edges with label JO or JA, and
   - reachable from sources only by edges with label either PO or PA.

Figure 9 illustrates G_{core3} (a control-observation graph of core3) whose vertices and edges are reachable to/from core3 using configurations for shaded ports. For each $c \in C$, let B_t be the set of all configuration IDs that exist in G_t . We define the configuration set K_t of the SoC with respect to t as the following equation and define each element $k \in K_t$ as the configuration of the SoC with respect to t.

$$
\begin{aligned}
K_t &= \prod_{c \in C} B_c \\
&= B_{c1} \times B_{c2} \times B_{c3} \times \ldots\ldots
\end{aligned}
$$

Step 2: Selection of a configuration of each core

For each element $k \in K_t$, the *control-observation graph of t with respect to k* ($G_{t,k}$) is created from a control-observation graph G_t as follows.

1. For each core $c \in C$, select a configuration that corresponds to k.
2. We define $G_{t,k}$ as the set of vertices and edges which are
   - reachable to sinks only by edges with label JO or JA, and
   - reachable from sources only by edges with label PO or PA.

Figure 10(a) illustrates $G_{core3,k3}$ (a control-observation graph of $core3$ with respect to $k3$). $G_{t,k}$ contains only one configuration (for shaded ports) in each core, and consecutive transparency of each core is realized.

Here, we define $G_{J_{t,k}}$ as the set of vertices and edges reachable to sinks only by edges with label JO or JA, and $G_{Pv_{t,k}}$ as the set of vertices and edges reachable from a source vertex v only by edges with label PO or PA. Figure 10(b) and (c) shows $G_{J_{core,k3}}$ and $G_{Pv1_{core3,k3}}$, respectively. Let $E_{J_{t,k}}$ and $E_{Pv_{t,k}}$ be the set of all edges in $G_{J_{t,k}}$ and $G_{Pv_{t,k}}$ respectively, we define $E_{Dv_{t,k}}$ as the following equation.

$$E_{Dv_{t,k}} = E_{J_{t,k}} \cap E_{Pv_{t,k}}$$

For $G_{J_{t,k}}$, let $Q^1_{J_{t,k}}$, $Q^2_{J_{t,k}}$ and $Q^3_{J_{t,k}}$ be the sets of all vertices $q \in G_{J_{t,k}}$ that satisfies the following conditions respectively.

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

$Q^2_{J_{t,k}}$ and $Q^3_{J_{t,k}}$ are the sets of all vertices that do not satisfy the Definition 6 and 7. Moreover, we define $Q_{J_{t,k}}$, $Q^{1,PI}_{J_{t,k}}$ and $Q^{1,source}_{J_{t,k}}$ as follows.

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

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

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

Similarly, for $G_{Pv_{t,k}}$, let $Q^1_{Pv_{t,k}}$ and $Q^2_{Pv_{t,k}}$ be the sets of all vertices $q \in G_{Pv_{t,k}}$ that satisfies the following conditions respectively.

1. $Q^1_{Pv_{t,k}}$: q is a sink.
2. $Q^2_{Pv_{t,k}}$: There exist cycles which contain q.

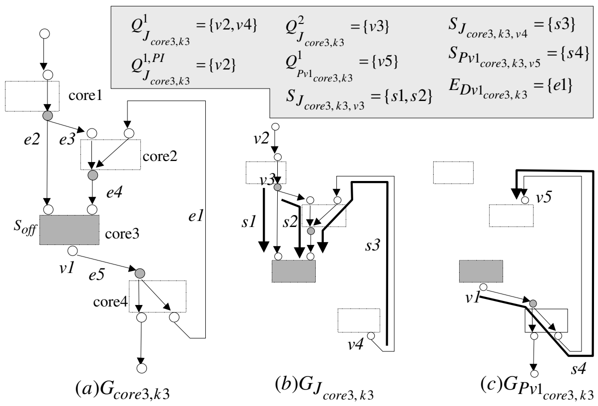


Figure 10. $G_{core3,k3}$, $G_{J_{core3,k3}}$ **and** $G_{Pv_{core3,k3}}$

$Q^2_{Pv_{t,k}}$ is the sets of all vertices that do not satisfy the Definition 8. Moreover, we define $Q_{Pv_{t,k}}$, $Q^{1,PO}_{Pv_{t,k}}$ and $Q^{1,sink}_{Pv_{t,k}}$ as follows.

$$Q_{Pv_{t,k}} = Q^1_{Pv_{t,k}} \cup Q^2_{Pv_{t,k}}$$

$$Q^{1,PO}_{Pv_{t,k}} = Q^1_{vP_{t,k}} \cap V_{PO}, \;\; Q^{1,sink}_{Pv_{t,k}} = Q^1_{Pv_{t,k}} \cap V_{sink}$$

Then, let $S_{Pv_{t,k,q}}$ be the set of all simple paths from a source vertex to $q \in Q_{Pv_{t,k}}$ in $G_{Pv_{t,k}}$.

Step 3: An ILP Formulation

We define the following 0-1 variables as ILP variables where a variable is equal to one if its condition is satisfied, otherwise is equal to zero.

y_t	:	element $t \in (C \cup E_{net})$ is consecutively test accessible
$y_{t,v}$	:	t is consecutively test accessible with respect to port v
$y_{t,v,k}$	:	t is consecutively test accessible with respect to port v by configuration k
$j_{t,k}$	:	$G_{J_{t,k}}$ is consecutively controllable
$o_{t,v,k}$	:	$G_{Pv_{t,k}}$ is consecutively observable
$d_{t,v,k}$	:	$G_{J_{t,k}}$ and $G_{Pv_{t,k}}$ are disjoint
$h_{t,k,q}$	:	$G_{J_{t,k}}$ is consecutively controllable with respect to vertex q
$a_{t,k,q,r}$	:	output edge r of vertex q in $G_{J_{t,k}}$ is consecutively controllable with respect to q
$m_{t,k,s}$	:	MUX is inserted at the simple path s in $G_{J_{c,k}}$ for controllability
$x_{t,k,e}$	:	MUX is inserted at the interconnect e in $G_{J_{t,k}}$ for controllability
x_e	:	MUX is inserted at the interconnect e in G for controllability
$x_{e,reg}$	:	register is inserted at the interconnect e in G for controllability
$f_{t,v,k,q}$	:	$G_{Pv_{t,k}}$ is consecutively observable with respect to vertex q
$n_{t,v,k,s}$	:	MUX is inserted at the simple path s in $G_{Pv_{t,k}}$ for observability
$z_{t,v,k,e}$	:	MUX is inserted at the interconnect e in $G_{Pv_{t,k}}$ for observability
z_e	:	MUX is inserted at the interconnect e in G for observability
$z_{e,reg}$	:	register is inserted at the interconnect e in G for observability

The following ILP formulation minimizes the required test overhead (i.e., total bit width of MUXes and registers) while making all cores consecutively testable. Figure 11 shows an example of constraints for core3 in S_1.

Minimize

$$\sum_{e \in E_{net}} (x_e + z_e) \cdot width(e) \tag{1}$$

Subject to:

1. for each element $t \in (C \cup E_{net})$,

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

2. For each port v, which is required for observation in order to test t,

$$y_{t,v} \geq y_t \tag{3}$$

3. For each t which can be tested by either S_{off} or S_{on},

   (a) if $K_t = \phi$,

$$x_t + x_{t,reg} + z_t + z_{t,reg} \geq 4 \cdot y_{t,v} \tag{4}$$

   (b) otherwise,

$$\sum_{k \in K_t} y_{t,v,k} \geq y_{t,v} \tag{5}$$

4. For each element $k \in K_t$,

$$j_{t,k} + o_{t,v,k} + d_{t,v,k} \geq 3 \cdot y_{t,v,k} \tag{6}$$

5. For each $G_{J_{t,k}}$,

   (a) if $G_{J_{t,k}} = \phi$,

$$x_t + x_{t,reg} \geq 2 \cdot j_{t,v} \tag{7}$$

   (b) otherwise,

$$\sum_{q \in Q_{J_{t,k}}} h_{t,k,q} \geq |Q_{J_{t,k}}| \cdot j_{t,k} \tag{8}$$

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

6. For each $G_{Pv_{t,k}}$,

(a) if $G_{Pv_{t,k}} = \phi$,

$$z_t + z_{t,reg} \geq 2 \cdot o_{t,v,k} \tag{9}$$

(b) otherwise,

$$\sum_{q \in Q_{Pv_{t,k}}} f_{t,v,k,q} \geq |Q_{Pv_{t,k}}| \cdot o_{t,v,k} \tag{10}$$

7. For each element $e \in E_{Dv_{t,k}}$,

$$x_{t,k,e} + z_{t,v,k,e} \leq d_{t,v,k} \tag{11}$$

8. For each vertex $q \in (Q_{J_{t,k}} \cup Q_{Pv_{t,k}})$,

   (a) if $q \in (Q^1_{J_{t,k}} - Q^{1,PI}_{J_{t,k}})$ and the TPS/TRS type required to test c is S_{off}, or $q \in (Q^1_{J_{t,k}} - (Q^{1,PI}_{J_{t,k}} \cup Q^{1,source}_{J_{t,k}}))$ and the TPS/TRS type required to test c is S_{on},or $q \in Q^3_{J_{t,k}}$,

$$\sum_{s \in S_{J_{t,k,q}}} m_{t,k,s} \geq |S_{J_{t,k,q}}| \cdot d_{t,k,q} \tag{12}$$

   (b) if $q \in Q^2_{J_{t,k}}$, let $R_{t,k,q}$ be the set of all output edges of q. For each element $r \in R_{t,k,q}$, let $S^r_{J_{t,k,q}}$ be the set of all simple paths which contain r in $S_{J_{t,k,q}}$,

$$\sum_{r \in R_{t,k,q}} a_{t,k,q,r} \geq d_{t,k,q} \tag{13}$$

$$\sum_{s \in (S_{J_{c,k,q}} - S^r_{J_{c,k,q}})} m_{t,k,s} \geq |S_{J_{c,k,q}} - S^r_{J_{c,k,q}}| \cdot a_{t,k,q,r} \tag{14}$$

   (c) if $q \in (Q^1_{Pv_{t,k}} - Q^{1,PO}_{Pv_{t,k}})$ and the TPS type required to test c is S_{off}, or $q \in (Q^1_{Pv_{t,k}} - (Q^{1,PO}_{Pv_{t,k}} \cup Q^{1,sink}_{Pv_{t,k}}))$ and the TPS type required to test c is S_{on},or $q \in Q^2_{Pv_{t,k}}$,

$$\sum_{s \in S_{Pv_{t,k,q}}} n_{t,v,k,s} \geq |S_{Pv_{t,k,q}}| \cdot f_{t,v,k,q} \tag{15}$$

9. For each simple path $s \in S_{J_{t,k,q}} \cup S_{Pv_{t,k,q}}$,

   (a) if $s \in S_{J_{t,k,q}}$,

$$\sum_{e \in E_s} x_{t,k,e} \geq m_{t,k,s} \tag{16}$$

   (b) if $s \in S_{Pv_{t,k,q}}$,

$$\sum_{e \in E_s} z_{t,v,k,e} \geq n_{t,v,k,s} \tag{17}$$

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

10. For each edge $e \in E_{net}$,

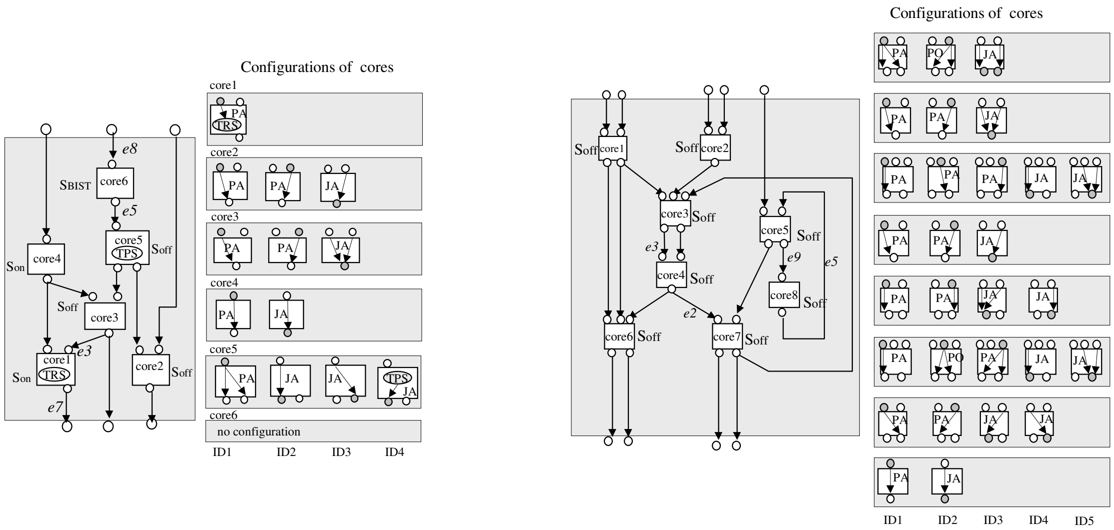
$$x_e \geq x_{t,k,e} \tag{18}$$

$$z_e \geq z_{t,v,k,e} \tag{19}$$

1.	$y_{core3} \geq 1$
2.	$y_{core3,v1} \geq y_{core3}$
3.	$y_{core3,v1,k1} + y_{core3,v1,k2} + y_{core3,v1,k3} + y_{core3,v1,k4} \geq y_{core3,v1}$
	$\vdots$
4.	$j_{core3,k3} + o_{core3,v1,k3} + d_{core3,v1,k3} \geq 3 \cdot y_{core3,v1,k3}$
	$\vdots$
5.	$h_{core3,k3,v3} + h_{core3,k3,v4} \geq 2 \cdot j_{core3,k3}$
6.	$f_{core3,v1,k3,v4} \geq o_{core3,v1,k3}$
7.	$x_{core3,k3,e1} + z_{core3,v1,k3,e1} \leq d_{core3,v1,k3}$
8.(a)	$m_{core3,k3,s3} \geq h_{core3,k3,v4}$
(b)	$a_{core3,k3,v3,e2} + a_{core3,k3,v4,e3} \geq h_{core3,k3,v3}$
	$m_{core3,k3,s2} \geq a_{core3,k3,v3,e2}$
	$m_{core3,k3,s1} \geq a_{core3,k3,v3,e3}$
(c)	$n_{core3,v1,k3,s4} \geq f_{core3,v1,k3,v4}$
9.	$x_{core3,k3,e2} \geq m_{core3,k3,s1}$
	$x_{core3,k3,e3} + x_{core3,k3,e4} \geq m_{core3,k3,s2}$
	$x_{core3,k3,e1} + x_{core3,k3,e4} \geq m_{core3,k3,s3}$
	$z_{core3,v1,k3,e1} + z_{core3,v1,k3,e5} \geq n_{core3,v1,k3,s4}$

Figure 11. Constraints for core3 in S_1

Equation (2) guarantees that all cores and all interconnects are consecutively testable. By Definition 2, each core is consecutively test accessible if, for each output port of the core, the core satisfies consecutive controllability and consecutive observability at the same time. This is guaranteed by equation (3). If TPS/TRS type required to test t is either S_{off} or S_{on}, t must be consecutively test accessible with respect to the TPS/TRS by more than one configuration k in K_t. This is guaranteed by equation (5). If $K_t = \phi$ (i.e., t is an interconnect between two opaque cores), t must be isolated by adding test MUXes and registers. This is guaranteed by equation (4). If TPS/TRS type required to test t is S_{BIST}, there is no constraint. This means that the t is already consecutively test accessible. By Definition 2, each core is consecutively test accessible if, for each output port of the core, the core satisfies consecutive controllability and consecutive observability at the same time. This is guaranteed by equation (6). In order to make $G_{J_{t,k}}$ consecutively controllable with respect to required TPSs, all vertices in $Q_{J_{t,k}}$ must be consecutively controllable with respect to the TPSs. This is guaranteed by equation (8). If $G_{J_{t,k}} = \phi$ (i.e., t is an interconnect and output edge from opaque core), t must be isolated by adding test MUX and register. This is guaranteed by equation (7). Similarly, equation (9) and (10) guarantees consecutive observability of $G_{Pv_{t,k}}$. In order to satisfy consecutive controllability and consecutive observability at the same time, $G_{J_{t,k}}$ and $G_{Pv_{t,k}}$ must be disjoint. This is guaranteed by equation (11). In order to make q in $Q^1_{J_{t,k}}$ consecutively controllable with respect to required TPSs, all simple paths in $S_{J_{t,k,q}}$ must be inserted MUXes and paths from TPSs must be added. However, if q is a vertex that represents the TPSs required to test t, q is already consecutively controllable with respect to the TPSs. In order to make q in $Q^3_{J_{t,k}}$ consecutively controllable with respect to required TPSs, all cycles which contain q must be broken by MUXes and paths from TPSs must be added. These are guaranteed by equation (12). Similarly, equation (15) guarantees con-



(a) System S_2

(b) System S_3

Figure 12. SoC Examples

secutive observability of q in $Q^1_{PV_{t,k}}$ and $Q^3_{PV_{t,k}}$. Each vertex q in $Q^2_{J_{t,k}}$ has more than two output edges with label JO, and all the edges propagate only the same sequence. In order to make q in $Q^2_{J_{t,k}}$ consecutively controllable with respect to required TPSs, MUXes must be inserted at all simple paths which include all the output edges except one output edge. This is guaranteed by equation (13) and (14). Let s be a simple path and let E_s be the set of all edges corresponding to interconnects in s, insertion of MUX at s means that MUX is inserted at more than one element in E_s. This is guaranteed by equation (16) and (17).

Test MUXes and registers are inserted at the edges obtained by solving the above ILP problem (Figure 7). By Lemma 4 and equation (1), a given SoC can be made consecutively testable with minimum hardware overhead. However, in the case when TPS/TRS type required to test t is S_{off}, it is necessary to add TPS/TRSs of type S_{off} (i.e., add vertices to V_{PI} or V_{PO}) if the summation of bit widths of edges that must be inserted MUXes to make t consecutively test accessible is larger than that of available TPS/TRSs. Similarly, in the case when TPS/TRS type required to test t is S_{on}, it is necessary to add TPS/TRSs of type S_{on} (i.e., add vertices to V_{source} or V_{sink}) if the same condition as above is satisfied.

6 Experimental Results

In this section, we present the experimental results obtained by applying the proposed method. We applied the method to three SoC examples shown in Figure 8 and Figure 12. System S_1 consists of four consecutively transparent cores and has no on-chip TPS/TRS. System S_2 consists of four consecutively transparent cores and two opaque cores, and it contains one on-chip TPS inside of core5 and one on-chip TRS inside of core1. System S_3 consists of eight consecutively transparent cores and has no on-chip TPS/TRS.

We used the *lp_solve* package from Eindhoven University of Technology [18]. Assuming that all interconnects are of the same bit-width, the running time is negligible (less than 0.1 second) for all three examples on a SUN Ultra 5 workstation. The results of the running SoC examples are shown in Table 2. Column "systems"denotes system name. Columns "#MUX(controllability)" and "#MUX(observability)" and "#Register(interconnect)" denote the number of MUXes added for consecutive controllability, MUXes added for consecutive observability and registers added for interconnect isolation, respectively. Each e_i shows an edge to which a DFT element is added.

Table 1 shows the estimations of area overhead. Column "interconnects" denotes the bit width of interconnects in each system. Columns "0.1" and "0.5", "1" and "10" of each system denote the area overhead when we assume that the system contains 0.1 million gates, 0.5 million gates, 1 million gates and 10 million gates, respectively. For example, "0.19" in the first row of the column "0.1" of system S_1 shows the area overhead of the DFT elements (2 MUXes and 0 registers) when we assume that S_1 contains 0.1 million gates and all interconnects in S_1 are of 32 bit width.

In our proposed method, the delay overhead is negligible since at most only one multiplexer is inserted to each interconnect.

7 Conclusions

In the paper, we proposed 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 TPSs, and the test responses appeared at an output port of the core

Table 1. Estimation of Area Overhead(%)

	system S_1 (million gates)				system S_2 (million gates)				system S_3 (million gates)			
interconnects (bit)	0.1	0.5	1	10	0.1	0.5	1	10	0.1	0.5	1	10
32	0.19	0.04	0.02	0.00	0.61	0.12	0.06	0.01	0.58	0.12	0.06	0.01
64	0.39	0.08	0.04	0.00	1.22	0.24	0.12	0.01	1.16	0.23	0.12	0.01
128	0.77	0.15	0.08	0.01	2.44	0.48	0.24	0.02	2.31	0.46	0.23	0.02
256	1.54	0.31	0.15	0.02	4.87	0.97	0.49	0.05	4.61	0.92	0.46	0.05
512	3.07	0.61	0.31	0.03	9.73	1.94	0.97	0.10	9.22	1.85	0.92	0.10
1024	6.15	1.23	0.62	0.06	19.46	3.89	1.95	0.20	18.44	3.69	1.84	0.18

Table 2. Results of the running SoC examples

	# DFT elements		
systems	#MUX (controllability)	#MUX (observability)	#Register (interconnect)
S_1	1 (e_4)	1 (e_5)	0
S_2	3 (e_3, e_5, e_7)	2 (e_3, e_8)	4 (e_3, e_5, e_7, e_8)
S_3	4 (e_2, e_3, e_5, e_9)	2 (e_5, e_9)	0

are propagated to TRSs consecutively at the speed of system clock. The propagation of test patterns and responses is achieved by using interconnects and consecutively transparent paths of surrounding cores. All interconnects can be tested in a similar fashion. Therefore, it is possible to apply arbitrary test sequence and observe its response sequence consecutively at the speed of the system clock. We also proposed a design-for-testability method that makes a given SoC consecutively testable using an ILP problem. Our future work is to propose a DFT method 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] N.A.Touba and B.Pouya, "Testing embedded cores using partial isolation rings," Proc. 15th VLSI Test Symp., pp.10-16, May 1997.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] K.Chakrabarty, "Design of System-on-a-Chip Test Access Architectures Using Integer Linear Programming," Proc. 18th VLSI Test Symp., pp.127-134, May 2000.

[8] K.Chakrabarty, "Design of System-on-a-Chip Test Access Architectures under Place-and-Route and Power Constraints," Proc. 37th Design Automation Conf., pp.432-437, June 2000.

[9] E.Marinissen, R.Arendsen, G.Bos, H.Dingemanse, M.Lousberg and C.Wouters, "A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores," Proc. 1998 Int. Test Conf., pp.284-293, Nov. 1998.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] T.Yoneda and H.Fujiwara, “A DFT method for core-based systems-on-a-chip based on consecutive testability,” Proc. 10th Asian Test Symp., pp.193-198, Nov. 2001.

[17] K.Chakrabarty, R.Mukherjee and A.Exnicios, “Synthesis of Transparent Circuits for Hierarchical and System-on-a-Chip Test,” Proc. IEEE International Conference on VLSI Design, pp.431-436, Jan. 2001.

[18] M.Berkelaar, *lp_solve*, version 3.2, Eindhoven University of Technology, The Netherlans, *ftp://ftp.ics.ele.tue.nl/pub/lp_solve*.