Area and Time Co-Optimization for System-on-a-Chip based on Consecutive Testability

Tomokazu Yoneda†, Tetsuo Uchiyama‡ and Hideo Fujiwara†
†Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-0101, Japan
{yoneda, fujiwara}@is.aist-nara.ac.jp
‡SOC Design Center, CANON INC.
30-2,Shimomaruko 3-Chome,Ohta-ku,Tokyo 146-8501,Japan
uchiyama.tetsuo@canon.co.jp

Abstract

This paper presents an area overhead and test time co-optimization method for SoCs based on consecutive testability. Consecutive testability of SoCs guarantees that we can handle any test sequence that requires consecutive application of test patterns at speed of system clock such as a test sequence for timing faults. The proposed method creates a test schedule and TAM using existing interconnects as much as possible. Moreover, the method allows trade-off between area overhead and test time according to user defined ratio. Experimental results show that the proposed method can achieve lower area overhead compared to test bus architecture due to the utilization of existing interconnects as a part of TAM.

keywords: *system-on-a-chip, design for testability, test access mechanism, test scheduling, consecutive testability*

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 hundred 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 known 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.

Testing of SoCs introduces several new challenges compared to testing of conventional IC designs [1]. A major problem to make an SoC testable concerns accessibility of embedded cores. Since embedded cores are not directly accessible via chip inputs and outputs, special access mechanisms are required to test them after system integration. The development of efficient test access mechanism (TAM) is an integral part of SoC test. Several TAM architectures 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 [2, 3, 4] or flexible *TESTRAIL* [5] around the cores to propagate test data. The second approach uses *boundary scan* architectures [6, 7] to isolate the core during test. The third approach uses *transparency* [8, 9, 10] for embedded cores to reduce the problem to one of finding paths from chip inputs to core inputs and from core outputs to chip outputs. In order to reduce the time-to-market and test cost, test scheduling that minimizes the test time for SoCs is also an integral part of SoC test. Several test scheduling techniques have been proposed to minimize the test time by adopting an appropriate TAM architecture [11, 12, 13].

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 designed 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 an 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 test access *consecutive test access*. Similarly, consecutive test access mechanisms are required to test interconnects between cores.

There are two approaches [14, 15] realizing the consecutive test access for both cores and interconnects. In [14], we proposed *consecutive testability* of SoCs and *consecutive transparency* of cores. Consecutive transparency of a core guarantees that arbitrary test/response sequences can be propagated from the core inputs to the core outputs without information loss. Consecutive testability of SoCs guarantees that it is possible to apply/observe arbitrary test/response sequences to/from all embedded cores and all interconnects by using interconnects and consecutively transparent cores. Therefore, the method can handle any test sequence that requires consecutive application of test patterns at speed of system clock such as a sequence for timing faults. We proposed the method to augment a given SoC into consecutively testable one [14] as well as the method to augment a given core consecutively transparent one [16]. However, in these two approaches [14, 15], only the technique to minimize area overhead were proposed and test time reduction was not addressed. Moreover, there is no discussion about core model, and the treatment of scan chains in scan-designed cores is not considered explicitly.

In this paper, we extend the target core model so that we can handle IEEE P1500 wrapped cores [17] and scan-designed cores in addition to non-scan designed cores that we considered in [14]. In order to simplify the discussion, built-in-self-testable (BIST) cores are not considered in this paper though they can be included easily by applying the proposed approach in [14]. For SoCs that include the above core models, we propose an area and time co-optimization method based on consecutive testability. We create TAM and a test schedule by using integer linear programming (ILP), and augments a given SoC into consecutively testable one where area overhead and test time are co-optimized. In the proposed method, TAM for consecutive testability is designed based on utilization of existing circuits. When we design TAM, we use interconnects and cores' consecutive transparency as much as possible. Only when TAM cannot be designed by using only the above existing circuits, we add extra circuits (test buses). Therefore, our method achieves lower area overhead compared to conventional test bus architecture. In order to evaluate area impact of TAM, we discuss the estimation of bus area based on floor plan. Experimental results show advantages of the proposed method compared to test bus architecture.

The rest of this paper is organized as follows. Section 2 gives SoC modeling and some definitions. In section 3, we show an area and time co-optimization method that creates TAM and a test schedule by using ILP. Experimental results are discussed in section 4. Finally, section 5 concludes this paper.

2 Preliminaries

2.1 SoC Modeling

We assume that an SoC consists of *cores*, *primary inputs*, *primary outputs* and *interconnects* (Figure 1) and all cores operate using single clock frequency.

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 port with a primary output. Though any number of interconnects can connect to the same output port (i.e., fanout is allowed), only one interconnect can connect to the same input port. It is not necessary that interconnects are of the same bit width. In Figure 1, the shaded number beside each interconnect represents the bit width of the interconnect.

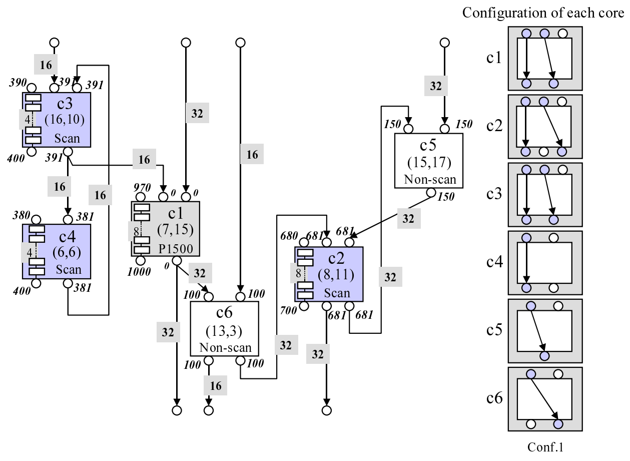


Figure 1. System-on-a-Chip S_1

We consider three types of cores; IEEE P1500 wrapped cores (P1500 cores), scan-designed cores (scan cores) and non-scan-designed cores (non-scan cores). Each individual core can be tested by external test and a pre-computed test sequence is available for the core which, if applied to the core, will result in a very high fault coverage. P1500 cores and scan cores have *scan input/output ports* which are used only for testing the cores and have no connection to other ports. P1500 cores can be tested by using only the scan port. Therefore, we consider that a test/response sequence is provided for each port. In Figure 1, the number beside each port represents the length of test/response sequence for the port.

A floor plan is provided for an SoC and each core has placement denoted by (x, y) coordinates of its center of gravity. In Figure 1, the numbers in parentheses represents the (x, y) coordinates of each core. Area overhead of a wire is estimated as the product of width and length on the floor plan. We use Manhattan distance for calculating length of wires used as a part of TAM. Moreover, if a core is consecutively transparent (defined in the next section), an information about consecutive transparency of the core is also provided. Otherwise, no information about consecutive transparency is provided.

2.2 Consecutive Transparency of a Core

We introduced a concept called consecutive transparency of cores defined as follows and proposed the method to transform a given core into consecutively transparent one [16]. Consecutive transparency guarantees that, for each port, there exists a test mode called a *configuration* which realizes consecutively transparent paths for the port (Figure 2). Here, paths are consecutively transparent in the sense that any test sequence can be propagated through them without information loss.

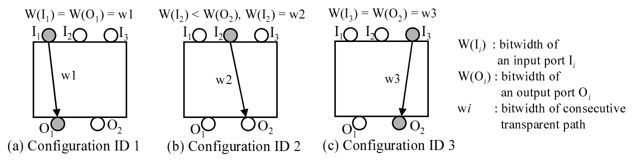

$W(I_1) = W(O_1) = w1$
$W(I_2) < W(O_2)$, $W(I_2) = w2$
$W(I_3) = W(O_2) = w3$
I_1 I_2 I_3
w1
w2
w3
O_1 O_2
(a) Configuration ID 1
(b) Configuration ID 2
(c) Configuration ID 3
$W(I_i)$: bitwidth of an input port I_i
$W(O_i)$: bitwidth of an output port O_i
wi : bitwidth of consecutive transparent path

Figure 2. Various configurations of a consecutively transparent core

Definition 1 *Consecutive transparency of a core*
Let $I(i)$ be the ith bit of a PI I, , $O(j)$ be the jth bit of a PO O and T be a PI. Suppose that there exists a configuration (test mode controlled by T) of a core that 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*. A PI is called to be *consecutively transparent* if there exists a configuration that can make all bits of the PI consecutively transparent at the same time. Similarly, A PO is called to be *consecutively transparent* if there exists a configuration that can make all bits of the PO consecutively transparent at the same time. Moreover, a core is called to be *consecutively transparent* if all PIs and POs except T are consecutively transparent. ■

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

In [14], we proposed a new test methodology based on consecutive testability of SoCs defined as follows.

Definition 2 *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. ■

Consecutive testability of SoCs guarantees that it is possible to apply/observe arbitrary test/response sequences to/from all embedded cores and all interconnects without information loss by using interconnects and consecutively transparent cores. Figure 3 illustrates a consecutively testable SoC and the consecutive test access to/from Core 3. A control signal is provided for each consecutively transparent core by a test controller (either off-chip or on-chip) and determines the configuration of the core.

3 Area and Time Co-Optimization

In this section, we present an area overhead and test time co-optimization method based on consecutive testability. The method creates TAM and a test schedule, and augments a given SoC into consecutively testable one by adding extra circuits (design-for-testability (DFT) elements) where area overhead and test time are co-optimized. When we create consecutively test accessible TAM, we consider test bus (Figure 4(a)), consecutive transparency (Figure 4(b)), direct path from a PI to a core (from a core to a PO) with multiplexer (Figure 4(c)) and existing interconnect as compo-

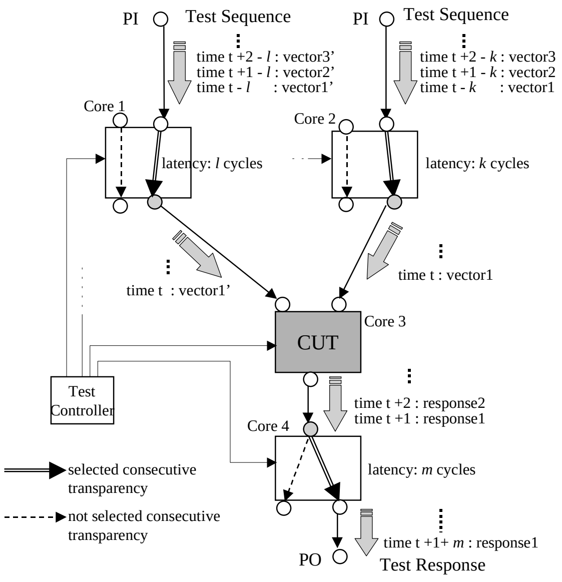


Figure 3. Consecutive test access for core3

nents of TAM. We try to utilize existing interconnects and consecutive transparency of cores as much as possible to minimize hardware overhead. Only when a core is not consecutively test accessible by using only existing interconnects and consecutive transparency of cores, we add direct paths to the core (Figure 4(c)) or make other cores consecutively transparent (Figure 4(b)) with multiplexers. For scan ports, we add test buses since scan ports have no connection to other ports and cannot utilize existing interconnects (Figure 4(a)) . The more direct paths and test buses we add, the shorter test time we can achieve. There is a trade-off between hardware overhead and test time. Then, we formulate area overhead and test time co-optimization based on consecutive testability according to user objective as the following optimization problem.

Definition 3 *Area and time co-optimization problem based on consecutive testability*

- Input : An SoC, co-optimization ratio α
- Output : A consecutively testable SoC and a test schedule
- Optimization : Minimizing hardware overhead (i.e., MUXes and area of buses) and test time(eq.(1))

$$\alpha \cdot (areaoverhead) + (1-\alpha) \cdot (testtime) \qquad (1)$$
$$0 \leq \alpha \leq 1$$ ■

3.1 Area and Time Co-Optimization Algorithm

In this subsection, we describe the proposed algorithm. The algorithm consists of the following three stages.

Stage1: TAM design for scan ports
Stage2: Design for consecutive transparency of all cores
Stage3: TAM design and test scheduling co-optimization

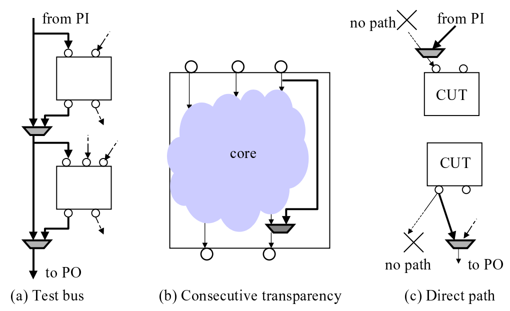


Figure 4. DFT elements

3.1.1 TAM Design for Scan Ports (Stage 1)

We cannot utilize existing interconnects as a part of TAM for scan ports since scan ports have no connection to other ports. In Stage 1, we consider the following TAM design problem for scan ports using test buses as DFT elements.

Definition 4 *TAM design problem for scan ports (P_{scan})*

- Input :
  - Cores with scan ports (bit width of each port, length of test sequence and coordinates)
  - Co-optimization ratio α
  - maximum bit width of I/O pins $W_{soc,in}$, $W_{soc,out}$
- Output : TAM
  - the number of test buses
  - width of each test bus
  - an assignment of cores to the test buses
- Optimization : Minimizing hardware overhead (i.e., MUXes and area of test bus) and test time for scan ports (eq.(1)) ■

The algorithm of this stage consists of the following two steps.

Step 1: *Estimate TAM area and test time*

Let C_s be the set of P1500 cores and scan cores, let $\mathcal{P}(C_s)$ be the power set of C_s. For each set of cores $C_p \in \mathcal{P}(C_s)$, we estimate TAM area ($Cost(C_p)$) and test time ($Time(C_p)$) in the case of connecting the cores by one test bus as follows.

$$Cost(C_p) = bus\_length(C_p) \times \max_{c \in C_p}(port\_width(c)) \quad (2)$$

Here, $bus\_length(C_p)$ denotes the Manhattan distance in the case of connecting all cores in C_p by one test bus, and $port\_width(c)$ denotes the bit width of scan port of c

$$Time(C_p) = \sum_{c \in C_p} sequence(c) \quad (3)$$

Here, $sequence(c)$ denotes the length of test sequence for c.

Figure 5 shows two examples of routing. One is a routing that connects all cores in $C_{13} = \{c1, c2, c4\}$ by a test bus.

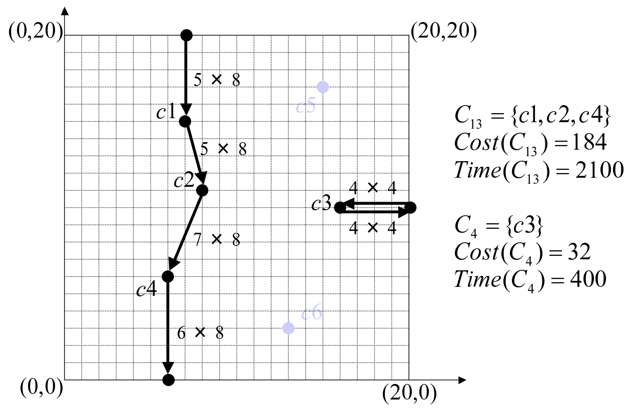


Figure 5. routing examples

The other is a routing that connects only $c3$ by a test bus.

Step 2: *Determine the number of test buses and assignment of cores to the test buses*

In this step, we design TAM where area and test time are co-optimized according to user defined ratio α. In order to design consecutively test accessible TAM with test buses for all cores in C_s, we should find a subset M of $\mathcal{P}(C_s)$ such that M satisfies the following equation.

$$\bigcup_{C_p \in M} C_p = C_s \quad (4)$$

Here, $|M|$ denotes the number of test buses, and $C_p \in M$ denotes the set of cores assigned to a test bus. Once the number of test buses and assignment of cores to the test buses are determined, area overhead and test time is also determined by the estimation in Step 1. We formulate the above subset selection as the following ILP problem.

0-1 variables

(*1 if each condition is satisfied, otherwise 0*)

x_{i,C_p} : core i is assigned to a test bus with cores in $C_p \in \mathcal{P}(C_s)$

y_{C_p} : C_p is a element of M

Minimize

$$\alpha \cdot \left(\sum_{C_p \in \mathcal{P}(C_s)} Cost(C_p) \cdot y_{C_p} \right) + (1-\alpha) \cdot \left(\max_{C_p \in \mathcal{P}(C_s)} (Time(C_p) \cdot y_{C_p}) \right) \quad (5)$$

Subject to:

1. core assignment to test bus
   - for each $i \in C_s$ and $C_p \in \mathcal{P}(C_s)$,

$$\sum_{C_p \in \mathcal{P}(C_s)} x_{i,C_p} = 1 \quad (6)$$

$$\sum_{i \in C_p} x_{i,C_p} = |C_p| \cdot y_{C_p} \quad (7)$$

2. I/O bit width limitation,

$$W_{soc,in} \geq \sum_{C_p \in \mathcal{P}(C_s)} W_{in}(C_p) \cdot y_{C_p} \quad (8)$$

$$W_{soc,out} \geq \sum_{C_p \in \mathcal{P}(C_s)} W_{out}(C_p) \cdot y_{C_p} \quad (9)$$

Here, $W_{in}(C_p)$ and $W_{out}(C_p)$ are constant values which denote the summation of bit width of cores' input ports and output ports in C_p, respectively.

We can determine the number of test buses and assignment of cores to the test buses by solving above ILP problem, and design TAM for scan ports where area and test time are co-optimized according to user defined ratio α.

3.1.2 Design for Consecutive Transparency of All Cores (Stage 2)

In order to satisfy consecutive test accessibility of interconnects, all input/output ports of all embedded cores should be consecutively observable/controllable. Therefore, all cores should be consecutively transparent. In this stage, we consider the following design for consecutive transparency problem (defined in [16]) for all cores.

Definition 5 *Design for consecutive transparency problem (P_{bypass}).*

- Input : A core
  - bit width of each port
  - consecutively transparent paths if they have
- Output : A consecutively transparent core
- Optimization : Minimizing hardware overhead (i.e., hardware of added MUXes) ■

The algorithm for this stage have been proposed in our previous work [16] and we have used it in this paper.

3.1.3 TAM Design and Test Scheduling Co-Optimization (Stage 3)

In the Stage 1, we designed TAM for scan ports using test buses where area and test time are co-optimized according to user defined ratio α. In the Stage 2, we made all cores consecutively transparent for consecutive test accessibility of all interconnect. Figure 6 shows an example SoC (corresponding to the SoC shown in Figure 1) after Stage 1 and Stage2 in the case of $\alpha = 1$ where test buses are added for scan ports and all cores are made consecutively transparent. This figure represents only the assignment of cores and do not shows the routing of test buses exactly. In this Stage 3, we consider the following TAM design and test scheduling co-optimization problem based on consecutive testability. We create a test schedule by determining the combinations of cores tested simultaneously (in the same configuration) and design TAM for consecutive test accessibility of the cores according to user defined co-optimization ratio α.

Definition 6 *TAM design and test scheduling co-optimization problem (P_{select}).*

- Input : An SoC with TAM for scan ports and all consecutively transparent cores
- Output : A consecutively testable SoC and a test schedule

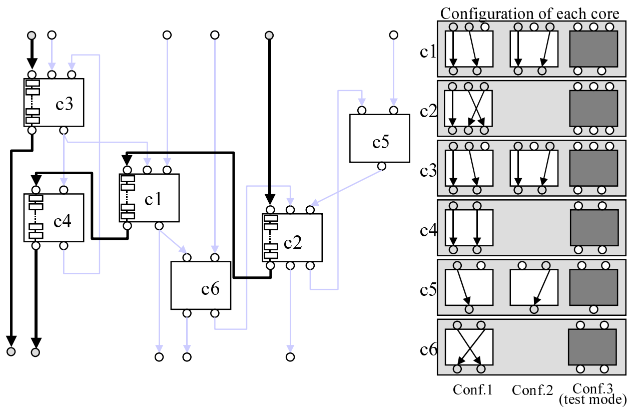


Figure 6. An example SoC after State 2

- Optimization : Minimizing hardware overhead (i.e., MUXes and wire area) and test application time of the SoC (Equation 1) ■

The algorithm of this stage consists of the following two steps. In the Step 1, it adds direct paths realized by multiplexers from PIs to core inputs and from core outputs to POs. Then, it determines TAM for all cores and all interconnects by selecting paths added in Step 1.

Step 1: *Add all direct paths from PIs to core inputs and from core outputs to POs*

For input/output ports which are not controllable/observable directly at PIs/POs of a SoC, we add direct paths from PIs to the input ports (from the output ports to POs) with multiplexers in order to guarantee consecutive accessibility for all ports.

Step 2: *Create TAM and a test schedule*

In this step, we determine the combinations of cores tested simultaneously and create TAM for each combination by selecting configurations of other cores and direct paths added in Step 1 so that area overhead and test time are co-optimized according to user defined ratio α. We formulate the above decision and selection problems as an ILP problem using the following notations.

Notations for an ILP Formulation

Sets

C: all cores
$V_{in,c}$: all input ports of core c
$V_{out,c}$: all output ports of core c
E: all wires including all interconnects, all consecutively transparent paths, all direct paths and all test buses
E_{net}: all interconnects ($E_{net} \subset E$)
K: all configurations of a SoC ($K = \prod_{c \in C} T_c$)
here, T_c is the set of all configurations of core c
K_c: all configurations in which core c is tested ($K_c \subset K$)
C_k: all cores which are tested in configuration $k \in K$
V_k: all ports of all cores in C_k
$G_{k,v}$: all possible TAM for port $v \in V_k$

$E_{k,v,g}$: all wires in the TAM $g \in G_{v,k}$

Constant values

$S(e)$: hardware cost in order to realize $e \in E$

$L(v,g)$: maximum sequential depth of TAM g for port v

$R(v)$: length of test sequence for $v \in V$

0-1 variables:

(*1 if each condition is satisfied, otherwise 0*)

y_c: core c is consecutively test accessible

$y_{c,k}$: core c is consecutively test accessible in configuration k

y_k: configuration k is used to test the SoC

$y_{k,v}$: port v is consecutively test accessible in configuration k

$y_{k,v,g}$: TAM g is used for port v in configuration k

$x_{e,k,v}$: wire e is used for port v in configuration k

x_e : wire e is used to test the SoC

Integer variables

$in\_time_{k,c}$: test application time of core c in configuration k

$out\_time_{k,c}$: test observation time of core c in configuration k

$time_k$: total test time in configuration k

Minimize:

$$\alpha \cdot \left(\sum_{e \in E} S(e) \cdot x_e \right) + (1-\alpha) \cdot \left(\sum_{k \in K} time(k) \cdot y_k \right) \quad (10)$$

Subject to:

1. for each $c \in C$,

$$y_c \geq 1 \quad (11)$$

$$\sum_{k \in K_c} y_{c,k} \geq y_c \quad (12)$$

2. for each $k \in K$,

$$|C_k| \cdot y_k = \sum_{c \in C_k} y_{c,k} \quad (13)$$

$$\sum_{v \in V_k} y_{k,v} \geq |V_k| \cdot y_k \quad (14)$$

3. for each $v \in V_k$,

$$\sum_{g \in G_{k,v}} y_{k,v,g} \geq y_{k,v} \quad (15)$$

$$\sum_{e \in E_{k,v,g}} x_{e,k,v} \geq |E_{k,v,g}| \cdot y_{k,v,g} \quad (16)$$

4. for each $e \in E$,

$$\sum_{v \in V_k} x_{e,k,v} \leq 1 \qquad for\ k \in K \quad (17)$$

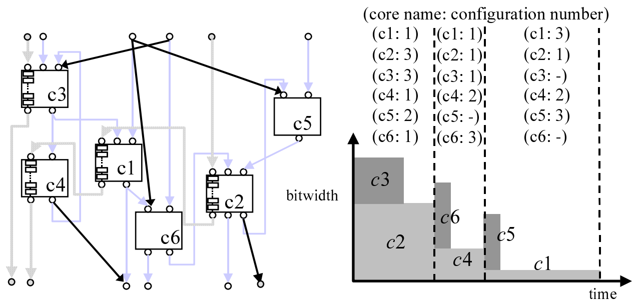
$$x_e \geq x_{e,k,v} \quad (18)$$

5. constraints for test time, for each $k \in K$,

$$in\_time_{k,c} = \max_{v \in V_{in,c}} \left(\sum_{g \in G_{k,v}} L(v,g) \cdot y_{k,v,g} \right) \quad (19)$$

$$out\_time_{k,c} = \max_{v \in V_{out,c}} \left(\sum_{g \in G_{k,v}} (L(v,g) + R(v)) \cdot y_{k,v,g} \right) \quad (20)$$

$$time_k = \max_{c \in C_k} ((in\_time_{k,c} + out\_time_{k,c}) \cdot y_k) \quad (21)$$



(a) Added direct paths (b) Test schedule

Figure 7. Result: added direct paths and a test schedule ($\alpha = 1$)

Equations (11) and (12) guarantee the consecutive test accessibility of all cores. Eqs. (13) and (14) are constraints for configuration k. These two Eqs. guarantee the accessibility of all ports of all cores tested in configuration k. Eqs. (15) and (16) guarantee the existence of TAM for all ports v in V_k. Eqs. (17) and (18) guarantee the disjointedness of TAM for all ports v in V_k. Eqs. (19), (20) and (21) calculate the test time in configuration k.

We can determine combinations of cores tested simultaneously and direct paths used as a part of TAM for each combination by solving above ILP problem. Figure 7 shows an example schedule and selected direct paths as a part of TAM in an SoC corresponding to Figure 1. Similarly testing of interconnects in addition to cores can be considered simultaneously by replacing the set C with $C \cup E_{net}$ in the notations and ILP formulation.

Through these three stages, we can augments a given SoC into consecutively testable one where area overhead and test application time are co-optimized according to user defined ratio α.

4 Experimental Results

In this section, we present experimental results obtained by the proposed method and make a comparison between the proposed method and test bus architecture. Since our approach cannot apply the SoCs that have no information about connectivity between cores, it is impossible to make experiments by using ITC'02 SoC benchmarks. Therefore, in this section, we present experimental results for a randomly created SoC System S_1 (shown in Figure 1). This SoC has three scan cores, two non-scan cores and one P1500 core. Please keep in mind that the main purpose of this work is: (1) to present the design methodology (TAM design and test scheduling) based on consecutive testability that allows trade-off between area overhead and test application time and (2) to show the advantage of using existing interconnect to achieve test access by comparing with conventional test bus architecture which always inserts additional paths. In this experiments, we considered testing of cores only since it is difficult to perform consecutive test ac-

Table 1. Results of our approach for S_1

	stage1(P_{scan})				stage2(P_{bypass})		stage3(P_{select})				Total			
α	Time	Area		CPU(s)	Area	CPU(s)	Time	Area		CPU(m)	Time	Area		CPU(m)
		wire	MUX		MUX			wire	MUX			wire	MUX	
1	2100	216	24	0.1	144	0.0	2260 (2125)	1536 (1216)	124 (92)	4320* (40)	2260 (2125)	1752 (1432)	292 (260)	4320* (40)
0.5	1700	224	24	0.3			- (1830)	- (1920)	- (148)	4320* (0.2)	- (1830)	- (2144)	- (316)	4320* (0.2)
0	1000	304	24	1.5			2100 (1500)	2816 (2432)	244 (228)	4320* (22)	2100 (1500)	3120 (2736)	412 (396)	4320* (22)

*:*lp_solve* was halted after 4320 minutes.

cess for interconnects by test bus architecture. We used the *lp_solve* package from Eindhoven University of Technology [18] on a SunBlade 1000, 900 MHz with 1GB RAM for the experiments.

Table 1 shows experimental results of the proposed method in the case of $\alpha = 0$ (area), $\alpha = 0.5$ (co-optimize) and $\alpha = 1$ (time). Figure 7 shows the result of a test schedule and direct paths added in Stage 3 in the case of $\alpha = 1$. "stage1(P_{scan})", "stage2(P_{bypass})" and "stage3(P_{select})" denote the results of Stage 1, Stage 2 and Stage 3, respectively. "Time", "Area" and "CPU" denotes the test time, the area overhead and running time of *lp_solve*, respectively. "wire" and "MUX" at the column "Area" denote the wire area estimated from a given floor plan and the bit width of multiplexer added in each stage,respectively. "Time" at the column "Total" denotes the total test time which is equal to "Time" at the column "stage3". "Area" and "CPU" at the column "Total" denote the total area overhead and computational time which is the summation of all stages. In Stage 3, we halted *lp_solve* after 4320 minutes (72 hours) and "Time" and "Area" denote the intermediate solutions after that time. In the case of $\alpha = 0.5$, we obtained no solution at that time.

In order to shorten the running time of *lp_solve*, we made experiments with reduced configuration set K' which is the subset of K (the set of all configurations of SoC) and created as follows. After Stage 1, for cores with scan ports, we can obtain the number of test buses and assignment of the cores to the test buses. Here, let N be the number of test buses and C_N be the set of cores assigned to test bus N. For each C_N, we first order the cores in the descending order of their lenght of test sequence to build a list L_{C_N}. For each L_{C_N}, one core is picked following the order in the list (i.e. a core which have longest test sequence in each test bus is picked). We only select the configurations from K such that all the cores picked from each L_{C_N} are test mode simultaneously and add the configurations to K'. Similarly, next core is pick following the order from each L_{C_N}, and we select configurations from K and add them to K'. In the above way, we create the reduced configuration set K'.

The results using the reduced configuration set K' are shown as the numbers in parentheses. From the results, we observe that the reduction of the configuration set K improves not only running time but also test application time and area overhead in all cases. From Table 1, we observe that the proposed method can allow trade-off between area overhead and test time according to user defined ration α.

Table 2. Results of test bus approach for S_1

	Test Bus(P_{scan})			
α	Time	Area		CPU(m)
		wire	MUX	
1	1650	2774	528	0.0
0.5	1250	2984	528	20
0	1000	3248	516	243

Table 2 shows results of the test bus architecture. Please notice that when we creates TAM, our approach uses existing interconnects, consecutive transparent cores and adds test buses. Therefore, by restricting TAM components to test buses in our approach, we can create TAM as test bus architecture (i.e. test bus approach is a special case of our approach). The results shown in Table 2 are obtained by applying our proposed method in Stage 1 assuming that all input/output ports of cores are scan ports and restricting that only test buses are added as TAM.

From Table 1 and Table 2, we observe that the proposed method achieves lower area overhead compared to test bus architecture in all three co-optimization ratio. Especially for $\alpha = 1$ (area has high priority), the proposed method achieves 50% reduction of area overhead compared to test bus architecture. This is because the proposed method utilizes existing interconnects and consecutively transparent cores as a part of TAM. On the other hand, the proposed method introduces longer test time. The proposed method is based on configuration-dependent scheduling which means that no new test are allowed to start until all tests in a configuration are completed. Therefore, test time depends on a core which has longest test sequence in a configuration. On the other hand, in the test bus approach, we can schedule tests for each test bus independently. We can consider that this disadvantage will be removed by adopting preemptive scheduling where Iyengar and Chakrabarty proposed in [19] while the advantages of the proposed method are kept. .

5 Conclusions

In this paper, we proposed an area and time co-optimization method for SoCs based on consecutive testability. The proposed method creates TAM and a test schedule by using integer linear programming, and augments a given SoC into consecutively testable one where area overhead and test time are co-optimized. The proposed method achieves lower area overhead compared to test bus architecture. Especially for the case where objective is to minimize area overhead, the proposed method achieves 50% area overhead reduction compared to test bus architecture. This is because the proposed method utilizes existing interconnects and consecutively transparent cores as a part of TAM. Consecutive testability of SoCs guarantees that arbitrary test/response sequences including timing information can be propagated to/from all embedded cores and all interconnects without information loss. Therefore, the method can handle any test sequence that requires consecutive application of test patterns at speed of system clock such as a sequence for timing faults. One of our future works is to improve the test scheduling method in order to shorten the test time. Another future work is to propose heuristic algorithms instead of current ILP based approach in order to shorten the computational time.

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 Japan Society for the Promotion of Science (JSPS) under Grants-in-Aid for Scientific Research B(2)(No. 15300018). The authors would like to thank Dr. Michiko Inoue, Dr. Satoshi Ohtake (Nara Institute of Science and Technology), Dr. Erik Larsson (Linöpings Universitet) , Dr. Tomoo Inoue and Dr. Hideyuki Ichihara (Hiroshima City University) 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] 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.

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

[7] 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.

[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.

[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] 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.

[12] 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.

[13] Erik Larsson, Klas Avidsson, Hideo Fujiwara and Zebo Peng, “Integrated Test Scheduling, Test Parallelization and TAM Design,” Proc. 11th Asian Test Symp., pp.397-404, Nov. 2002.

[14] Tomokazu Yoneda and Hideo Fujiwara, ”Design for Consecutive Testability of System-on-a-Chip with Built-In Self Testable Cores,” Journal of Electronic Testing: Theory and Applications (JETTA) Special Issue on Plug-and-Play Test Automation for System-on-a-Chip, Vol. 18, No. 4/5, pp.487-501, Aug./Oct. 2002.

[15] 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.

[16] Tomokazu Yoneda and Hideo Fujiwara, ”Design for Consecutive Transparency of Cores in System-on-a-Chip,” proc. 21st VLSI Test Symp.(VTS’03), pp.287-292, Apr. 2003.

[17] IEEE P1500 web site, *http://grouper.ieee.org/groups/1500/*.

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

[19] V.Iyengar and K.Chakrabarty, “Precedence-based, preemptive, and power-constrained test schduling for system-on-a-chip,” Proc. IEEE VLSI Test Symposium (VTS’01), pp.42-47, April 2001.