Test Scheduling of NoC-Based SoCs Using Multiple Test Clocks

Jin-Ho Ahn and Sungho Kang

Network-on-chip (NoC) is an emerging design paradigm intended to cope with future systems-on-chips (SoCs) containing numerous built-in cores. Since NoCs have some outstanding features regarding design complexity, timing, scalability, power dissipation and so on, widespread interest in this novel paradigm is likely to grow. The test strategy is a significant factor in the practicality and feasibility of NoC-based SoCs. Among the existing test issues for NoC-based SoCs, test access mechanism architecture and test scheduling particularly dominate the overall test performance. In this paper, we propose an efficient NoC-based SoC test scheduling algorithm based on a rectangle packing approach used for current SoC tests. In order to adopt the rectangle packing solution, we designed specific methods and configurations for testing NoC-based SoCs, such as test packet routing, test pattern generation, and absorption. Furthermore, we extended and improved the proposed algorithm using multiple test clocks. Experimental results using some ITC'02 benchmark circuits show that the proposed algorithm can reduce the overall test time by up to 55%, and 20% on average compared with previous works. In addition, the computation time of the algorithm is less than one second in most cases. Consequently, we expect the proposed scheduling algorithm to be a promising and competitive method for testing NoC-based SoCs.

Keywords: NoC test, test scheduling, rectangle packing, network-based TAM.

Manuscript received Dec. 23, 2005; revised May 29, 2006.

Jin-Ho Ahn (phone: + 82 2 2123 2775, email: sominaby@soc.yonsei.ac.kr) and Sungho Kang (shkang@yonsei.ac.kr) are with Department of Electrical & Electronic Engineering, Yonsei University, Seoul, Korea.

I. Introduction

It is expected that systems-on-chips (SoCs) including hundreds of embedded cores will appear in the near future through the development of a design methodology based on intellectual properties and deep submicron manufacturing technology. In such a highly dense SoC design, a communication scheme among built-in cores will be a main design constraint and will dominate the issues of system architecture, performance, robustness, power consumption, and cost. Until now, the shared bus has been generally exploited for the interconnection architecture within systems. However, due to increases in the number of embedded cores, system frequency, and deep submicron technologies, designers are contending with difficulties related to signal and power integrity within the shared bus architecture, and they will require new models and templates suitable for future SoCs. One such emerging approach is the network-on-chip (NoC)-based architecture and platform [1]. An NoC can be defined as an interconnection model implemented on a chip in the form of a micro-network [2]. An on-chip implementation of a network-based interconnection paradigm provides many advantages such as scalability and configurability [3]. For example, it is easier to add or delete built-in cores on an NoC structure. Moreover, the network operation clock can be arbitrarily determined since an on-chip network does not require strict clock synchronization with embedded cores like computer networks do. This is an important feature of any NoC-based SoC that includes multiple operation clocks. The basic structure of NoC-based SoCs consists of routers, functional and storage cores, routing channel connecting cores, and network interfaces (NIs) bridging between a core and a router [4]. The cores communicate with each other by sending and receiving packets composed of a header, payload, and trailer. Packet-based communication schemes can effectively utilize the full resources and bandwidth of networks.

Like all other SoCs, NoC-based SoCs must be tested for manufacturing defects. The general issues of SoC tests include the design of test wrappers, and test access mechanism (TAM) architectures, and test scheduling methods. The test wrapper is the logic added around an embedded core to isolate it from the surrounding logic and to provide test access to the core through a TAM. A TAM is the physical mechanism connecting cores from test sources or sinks, and it determines how efficiently test stimuli and test results can be transported. Test scheduling is applied to find the test organization that minimizes the overall test time while considering the test power and TAM architecture. NoC-based SoCs have nearly the same traditional test methodologies as common SoCs. However, some test issues incorporating NoC characteristics remain as unresolved problems under investigation. In particular, TAM research is one of the most active research areas in testing NoC-based SoCs. Among earlier TAM architectures for SoCs, an on-chip test bus was the most efficient form. Figure 1 shows a basic bus-based TAM architecture. However, it may be impractical to have a TAM solely for the purpose of testing in NoC-based SoCs because test costs such as those for the silicon area and pin count will be much higher. Thus, the reuse of NoCs as TAMs is a very attractive and logical goal. In an NoC-based TAM, all test data should come in a packet type. For this reason, it is difficult to assign TAM pins to cores by a bit scale. This constraint renders many test scheduling ideas based on a common SoC structure not directly applicable to testing NoC-based SoCs. Therefore, an efficient test scheduling method using an NoC-based TAM is important to minimize the overall system test time. In this paper, we propose a novel NoC-based SoC test scheduling, reusing the NoC for data communications as a TAM with variable test clocks. First, we designed a new NoC-based test platform, including test packet generation and routing. The proposed test platform helped us utilize previous test scheduling algorithms designed to test common SoCs [5]-[14]. Among the SoC test scheduling algorithms, we chose a rectangle packing approach to solve the NoC-based SoCs test scheduling problem. The rectangle packing heuristic was first introduced in [6]. The SoC test scheduling problem was formulated as a 2-dimensional bin packing problem, and each core was represented by a rectangle, the width of which was the number of SoC pins allocated, and the height of which was the core test time given the number of SoC pins [6]. A technique based on rectangle packing has been used for wrapper/TAM co-optimization and test scheduling for SoCs [7]. A restricted 3-dimensional bin-packing model [8] and techniques based on rectangle packing [9] have been used for power constrained SoC test scheduling. In this paper, we extended the rectangle packing procedure proposed in [7] for its simplicity and feasibility. In addition, we improved the proposed scheduling algorithm by factoring in multiple test clocks.

In the next section, we review prior studies and present the new contribution of our work. In section III, an NoC-based test platform for the proposed scheduling algorithm is introduced. We elucidate why the use of multiple test clocks is significant and how it can be implemented in section IV. The proposed scheduling method, which efficiently optimizes core assignment and schedules test order based on a rectangle packing approach, is presented with a pseudo-code in section V. The experimental results using ITC'02 benchmark circuits and comparisons with previous results are given in section VI. Finally, conclusions are presented in section VII.

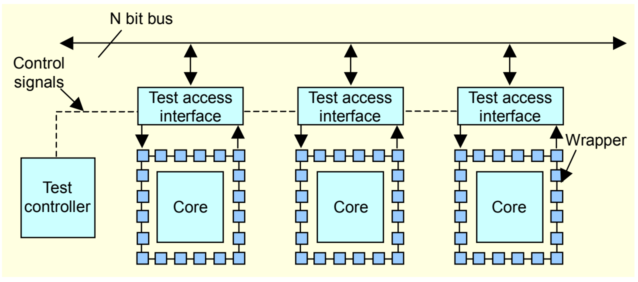


Fig. 1. Bus-based TAM architecture for SoC test.

II. Related Research and Summary of Contributions

The general concept of testing NoC-based SoCs was first shown in [4]. Before the built-in core test, the communication infrastructure of an NoC should be tested. The primary issues and methodologies regarding testing communication resources are introduced in [15]. After verifying the communication resource, we can advance to the standard core test using an on-chip network as a TAM. An early approach to the subject of test architectures utilizing an on-chip network was shown in [16]. The proposed network-oriented test architecture, novel indirect and modular architecture (NIMA), laid the groundwork for realizing a test architecture that benefits from the reuse of an NoC interconnect template.

Test scheduling algorithms for NoC-based SoCs can be grouped roughly into two main categories: packet-based and core-based scheduling. Packet-based scheduling determines the order of generation and transmission of test packets for cores according to the priority of each core. Cota and others have proposed test scheduling based on a packet-switching protocol [17]. Test vectors and responses per core are represented as a set of packets to be transmitted throughout the network, and the packets are scheduled to minimize the total test time using test parallelism [17]. 'Test parallelism' denotes that several cores are tested simultaneously, improving test efficiency through fully utilizing the network bandwidth. Enhanced versions of this algorithm have been reported that include the addition of power constraints [18]. Embedded processors have been used for test sources and sinks to increase test parallelism [19].

Core-based scheduling determines the test order of each core [20]-[21]. In this approach, the scheduler assigns each core a routing path, including an input port, an output port, and corresponding channels that transport test vectors from the input to the core and test responses from the core to the output. Once a core is scheduled on a path, all resources on that path are reserved for the core test until the entire test is completed. Because the proposed idea maintains test pipelining from a test vector input to a test response output for a core under test (CUT), it shows fairly good scheduling results. The term 'test pipelining' means the continuous, concurrent operation of scan input of a test vector and output of a response. As collision among the routed test data causes test pipelining to be broken, the corresponding routing path for each CUT has to be reserved. However, the reservation of a path reduces test parallelism. Evidently, packet-based scheduling is suitable for globally asynchronous and locally synchronous (GALS) architecture, and promises to fully exploit NoC characteristics. Even though the packet-based approach has merit, its test scheduling results are inferior to those of core-based methods so far.

Recently, a new test scheduling idea was proposed improving the limitation of the NoC-based TAM. In [21], the idle channel width of a TAM that cannot contribute to reducing the overall system test time is efficiently utilized through a combination of an on-chip clocking and parallel-serial conversion of test data. In order to preserve test pipelining, however, the test clock generated from an on-chip PLL for a CUT should increase in proportion to the number of test vectors transmitted together through a test packet. For example, let the size of a test packet for a CUT be 32 and the number of wrapper scan chains of the CUT be 2. In this case, a test packet can contain 16 test vectors. Thus, the test clock frequency of the CUT supplied from the PLL should be 16 times as fast as the operation frequency of the NoC-based TAM for test pipelining of the CUT. Considering the difficulty of an on-chip PLL design, we cannot help but limit the frequency and number of test clocks generated by the PLL, and therefore idle channel widths still remain.

Consequently, the test scheduling heuristics proposed so far for the testing of NoC-based SoCs are different from the SoC tests in previous studies, owing to the limits of NoC-based TAMs. Therefore, it is necessary to integrate both methods to enhance test efficiency. The major technical contributions made in this paper are the following:

(1) We exploit test scheduling methods based on common SoC architectures to test NoC-based SoCs. Though any heuristic approach for SoC test scheduling can be applied to NoC-based SoCs, we chose and extened the rectangle packing heuristic presented in [7] on the basis of its simplicity and feasibility.

(2) In order to apply the rectangle packing algorithm to NoC-based SoC test scheduling, we designed a new NoC-based test platform defined as specific methods and configurations for testing NoC-based SoCs such as test packet routing, test pattern generation, and absorption. Its details are presented in section III.

(3) Most SoCs consist of cores in variable clock domains. Some of those should be tested at-speed, and some should not. Thus, we consider test scheduling using multiple test clocks for a practical application. Particulars are presented in section IV.

III. NoC-Based SoC Test Platform

1. NoC Basics

An NoC can be characterized by several parameters such as topology, network protocol, structure, and control of a router. In this study, we assumed an NoC which uses a 2-D mesh topology, XY routing, and wormhole switching. A mesh structure is one of the most practical and widespread NoC topologies. Each router in a mesh is connected to its four neighboring routers via a bi-directional channel and an embedded core is attached to the router. The core communicates with the router through NI parsing or by making packet headers. The header contains information like the destination or origin of the packet. In wormhole switching, a packet is broken up into flits and they are transported in a pipelined manner. The flit is a unit of flow control. As common wormhole switching requires very small buffers and is implemented in hardware, it is suitable for multi-processor systems. XY routing, also known as dimension-order routing, is very popular for its simplicity and its capability of routing without deadlock. In XY routing, a packet is first routed on the X direction and then on the Y direction before reaching its destination. The simple NoC structure mentioned above is shown in Fig. 2. All routers have flit buffers at input ports and the flit size is the same as the channel width of the NoC.

2. Test Resources and Configuration for Testing NoC-Based SoC

A test resource denotes any specific logic required for a test operation. In this paper, a test source, sink, and controller are mainly used for test resources. A test source can generate test vectors at a rate of up to one packet per network time step, and

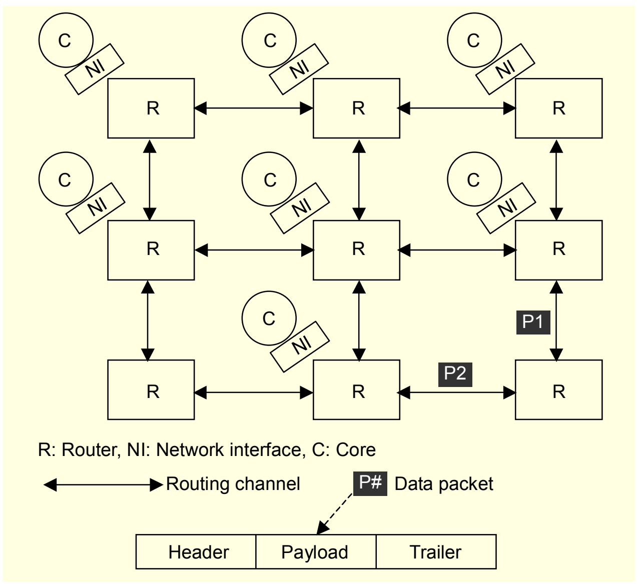


Fig. 2. Basic NoC structure.

a test sink can absorb test responses at the same rate as the test source. While a multi-source/sink mode has been used in previous studies [17]-[21], we adopt a single test source and sink attached directly to a router. Since test data should move on a network in a packet type, we assume the test source and sink include an NI internally for network routing. A test controller ensures that test sources and sinks satisfy test pipelining and parallelism by means of a predetermined test schedule. The number of test sources and sinks in a multi-source/sink mode corresponds to the difference of clock rate between an on-chip network and CUTs in a single source and sink mode. For example, if an on-chip network operates two times faster than the CUT, it produces the same effect as if there were two test sources and two sinks.

3. Generation and Transmission of Test Packets

Common SoC test scheduling achieves minimal test time by assigning the proper TAM width to each core. However, because NoC-based TAMs are not reconfigurable, we use the idea found in [21] to vary the size of TAM width available for scheduling. A packet can transfer multiple test vectors if the packet size is multiple times larger than the vector size [21]. The test vector size is identical to the TAM width assigned to a core. For example, let us assume that a test vector is 8-bit and a packet is 32-bit. In this case, the packet can transfer 4 vectors at a time. Such a packet generation scheme can reduce the overall test time by efficiently utilizing the network bandwidth in the spatial domain. Next, we propose a test packet transmission method including multiple vectors without the test clock multiplications used in [21].

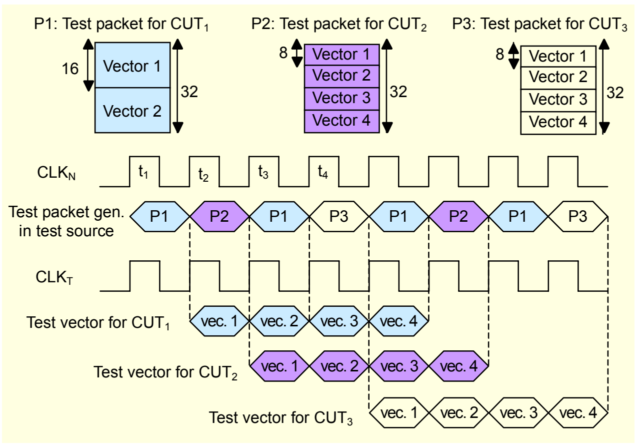


Fig. 3. Example of test packet generation and transmission.

An example of the proposed packet transmission is shown in Fig. 3. We assumed that the channel width of the network is 32-bit and CUT_1, CUT_2, and CUT_3 are assigned to 16-, 8-, and 8-bit TAM widths, respectively. CLK_N is a network clock, CLK_T is a CUT test clock and all clocks have the same frequency. First, $P1$ for CUT_1 is transmitted at cycle t_1. Since $P1$ has two test vectors, the test source can deliver $P2$ at the next t_2 cycle. In t_3, $P1$ should be sent again for test pipelining of CUT_1. $P3$ is delivered at t_4. The above process is repeated until the test of any one of the CUTs is done. In this paper, we confine the TAM width to the power of 2, that is, 1, 2,⋯, 2^k ($2^k \leq W$, W is the channel width of a network) for the simplicity of calculating transmission cycles. Because this time-division packet transmission allows a test scheduler to fully exploit the network bandwidth in the time domain, additional reduction of test application time can be realized.

4. Routing Strategy of Test Packets

As stated earlier, test packet collisions on a network should be avoided to preserve test pipelining. Instead of the path reservation that has been used in the past, a new test packet routing method using its routing characteristics is presented in this section.

Test vector packets from the test source to the CUTs show a 'one-to-many' communication pattern. In a one-to-many pattern, just one node is identified as a sender and the other nodes are receivers. If one sender transmits test packets in order and they move with XY routing, there will be no collision between test vector packets. However, the test response packets show a 'many-to-one' pattern because there is one test sink receiver, but there are many CUT senders. Since the transmission time of each CUT is not uniform, response packet collision should be expected. In order to solve this problem, we use the 'global combining' method [22]. Global combining

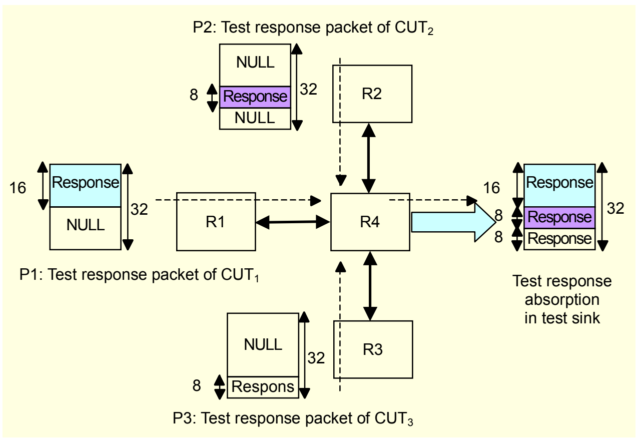

P2: Test response packet of CUT2
NULL
Response
NULL
8
32
R2
16
Response
NULL
32
R1
R4
Response
Response
Response
16
8
8
32
P1: Test response packet of CUT1
Test response absorption in test sink
NULL
Respons
32
8
R3
P3: Test response packet of CUT3

Fig. 4. Test packet routing with global combining.

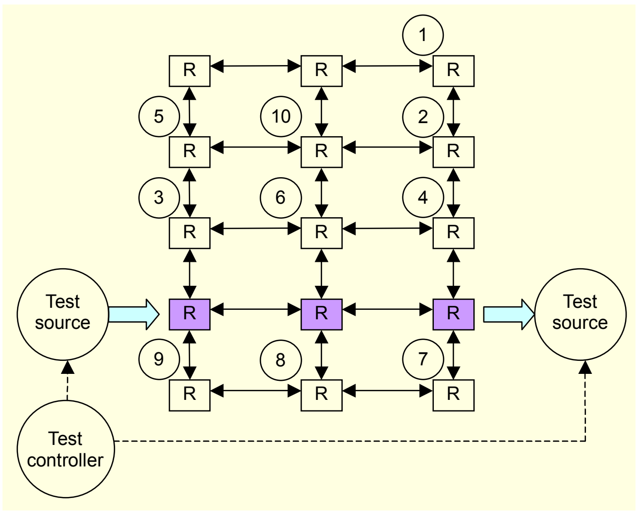


Fig. 5. Test configuration example of an NoC-based d695 circuit.

removes packet collision by combining different packets into one packet. XY routing is still valid even when global combining is used.

Figure 4 illustrates the process of packet routing using global combining. All of the basic conditions in Fig. 4 are equal to those of Fig. 3. When CUT_1 generates a response packet, $P1$, the response value will be located on the upper 16 bits of $P1$. In the case of $P2$, the response is located on the upper 8 bits within the lower 16 bits. $P3$ will use the other 8 bits for response data. Thus, the packet header must also include a piece of information, which indicates the relevant bits of the response. If $P1$, $P2$, and $P3$ arrive at the router, $R4$, simultaneously, $R4$ combines the input packets into one output packet, and forwards it. Since a test scheduler makes the total sum of TAM widths of CUTs tested at the same time less than the channel width of the network, it is very efficient to use global combining to solve the packet collision problem.

In spite of this routing scheme, it is possible that a test vector packet will collide with a response packet. This problem can be solved by placing a test source and sink on the opposite side of a row or column in an NoC and not installing another core in the row or column.

From the presentations given so far, we show as an example the overall configuration to test an NoC-based D695 benchmark circuit in Fig. 5.

IV. Test Scheduling Using Multiple Test Clocks

For a given core, the test time varies with the TAM width in a staircase pattern. For the sake of simplicity, this means that an increase of the TAM width of a core cannot always decrease the test time of the core. However, the increase in speed of the core test clock can always reduce the core test time as much as the test clock is increased; but the decrease of test parallelism due to the additional usage of network bandwidth is the downside of the test clock increase. Therefore, in order to maximize the effect using multiple test clocks for practicality as well as test efficiency, we must efficiently determine the test clock rates and schedule the test organization.

For example, let the clock rate of an on-chip network be 30 MHz and let the channel width be 10 bits (Fig. 6). Also, we assume the test times of core 1, core 2, and core 3 to be 200, 100, and 100 cycles, respectively, if a 10 MHz test clock is applied. In Fig. 6(a), since all cores can be tested at the same time, we simply see that the required time to test all cores is 200. Next, in Fig. 6(b), if we apply a 20 MHz test clock to test

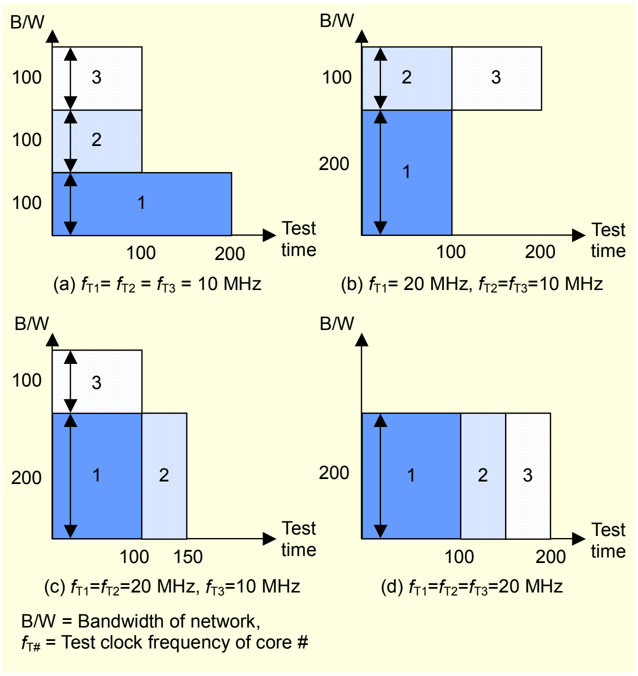


Fig. 6. Test scheduling example using multiple test clocks.

core 1, the overall test time will be unchanged as core 2 and core 3 should be tested sequentially. However, when the core 2 test clock is also doubled, as shown in Fig. 6(c), the overall test time can be reduced to 150. In the final example (Fig. 6(d)), if all cores are tested with a 20 MHz clock, the test time becomes 200 because cores are tested one by one to preserve test pipelining. As we can see in this example, under variable test clock domains, we should determine the clock rate and the core test order in consideration of several constraints, such as the extent of network bandwidth usage, power consumption, and precedence rules between cores. Note that no test clock should be faster than the network clock for test pipelining on the test platform. A detailed procedure to determine the test clock rate for each core will be presented in section V.2.

V. Test Scheduling Based on a Rectangle Packing

1. Problem Formulation

NoC-based SoC test scheduling can be formulated in terms of a rectangle packing problem if the test platform described in section III is used. For example, Fig. 7 illustrates the relationship between the test time and TAM width for a core. The test time varies with the TAM width in a staircase pattern, and the testing of a core is represented as a rectangle whose height indicates the TAM width assigned to that core; width denotes the test time of the core for the corresponding value of the TAM width. Thus we can obtain a number for the TAM width and the test time combinations for the same core. Taken as a whole, a test scheduler picks up just one rectangle from the candidate rectangle set of each core, and then packs it into a bin of a fixed height and an unlimited width until the bin is filled with rectangles of all cores embedded in an SoC, while minimizing the overall width of the bin without overflowing the bin's height.

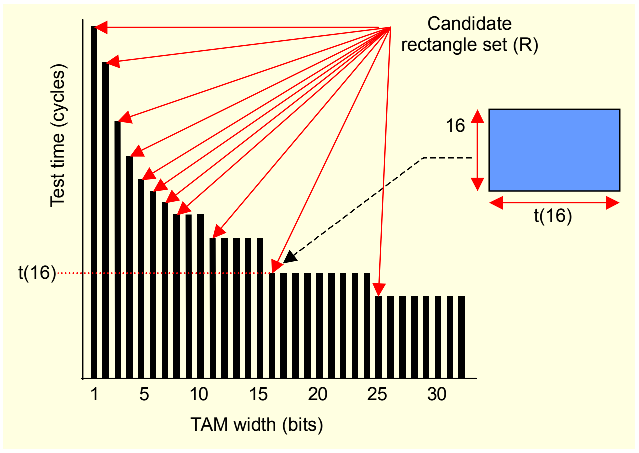


Fig. 7. Relationship between test time and TAM width of a core.

We now take up the NoC-based SoC scheduling problem in terms of a rectangle packing modeling. First, we assume that an NoC-based SoC N includes m cores and has the channel width W. Also, let w_i be the TAM width assigned to core i embedded to N and t_i be the test time of core i for w_i. All rectangles of core i, $\boldsymbol{R}_i$, are represented as an ordered pair such that

$$\boldsymbol{R}_i = (w_i(k),\ t_i(k)),\quad 1 \le i \le m,\quad 1 \le 2^k \le W,$$

where k denotes the number of TAM widths available for a time-division transmission. Note that $\boldsymbol{R}_i$ is 'Pareto-optimal.' The scheduling problem can be summarized as selecting one rectangle r_i from $\boldsymbol{R}_i$, $(1 \le i \le m)$, packing the selected rectangles into a bin of height W and unbounded width without overlapping between the r_i rectangles and minimizing the width of the bin.

Up to this point, the scheduling problem has had the same clock rate for the on-chip network and the core test clocks. However, as remarked in section III.2, a single source/sink mode is used in this paper instead of a multi-source/sink mode used in previous works. Therefore, we extend the aforementioned problem definition to a multiple bin packing problem. Let the network clock frequency be f_N and the test clock frequency of the cores be f_T. Further, let f_N be n times faster than f_T. Note that n is an integer value. Since a test scheduler can operate n times more within a period of test clock and the overall test time is measured by the test clock, we regard n as the number of bins to be packed by the core rectangles. In this multiple bin packing problem, we determine that a rectangle cannot be separated into multiple bins for a simple calculation of a time-division transmission time.

2. Test Scheduling Procedure

Before beginning the scheduling, we designed test wrappers for embedded cores and found $\boldsymbol{R}_i$. Though any wrapper design procedure can be applied, we use the 'one-element exchange' algorithm [14]. Additionally, the one-element exchange can optimize the result of partitioning scan chains initially derived from applying the 'largest processing time' (LPT), as the results from the LPT are not always best-optimized. Then, the heuristic procedure to solve the scheduling problem modeled in section V.1 is advanced. While many ideas have been proposed, we chose and extended the TAM_optimizer [7] for its simplicity and feasibility. In this section, all basic notations are identical to the comments in section V.1.

In the TAM_optimizer, first a scheduler calculates a preferred TAM width w_{prefer} for each core through heuristic ideas. Then, cores assigned to preferred TAM widths are scheduled in succession on the basis of their test time. The scheduler also supplements cores to idle room or assigns

```
1. $w_{prefer}$       // Preferred TAM width for the core
2. $w_{assigned}$     // Assigned TAM width for the core
3. begin_time         // Test start time
4. end_time           // Test end time
5. scheduled          // Indicates the core has been scheduled
6. completed          // Indicates testing has completed
7. bin_pos            // Position of bin that the core is scheduled
8. clk_mode           // Test clock rate
```

Fig. 8. Data structure of a core.

```
1. for ($i = 1; i \le m$; i++) {//m is the number of embedded cores.
       $A_i(k) = w_i(k) * T(w_i(k))$, $(1 \le 2^k \le W)$;// W is the channel width.
       Set $w_{prefer}(i) = w_i(k)$,
       where $w_i(k)$ is the highest Pareto-optimal width,
           such that $(A_i(k) - A_i(0)) \le (p * A_i(0))$;
       $T_{sum} = T_{sum} + T(w_{prefer}(i))$;
   }
2. for ($i = 1; i \le m$; i++) {
       Update $w_{prefer}(i) = w_i(k)$,
       where $w_i(k)$ is the highest Pareto-optimal width in $R_i$,
           if $(T(w_{prefer}(i)) / T_{sum}) \ge q$;
   }
```

Fig. 9. Preferred TAM width calculation.

additional TAM widths available to cores scheduled earlier to reduce the overall test time. While the basic scheduling methods are nearly the same as the TAM_optimizer, some parts are added or modified. We now explain them in detail.

A. Data Structure

The data structure of a core is shown in Fig. 8. In the data structure, *bpos*(*i*) denotes the position of the bin including core *i*, and *clk_mode*(*i*) is the test clock rate of core *i* relative to the normal test clock. For example, if *clk_mode*(*i*) is 2, the current test clock frequency of core *i* becomes two times faster than the test clock frequency set up at the beginning of the scheduling. Note that we assumed that test clock frequencies of cores are uniform at the start-up.

B. Calculation of Preferred TAM Width

The calculation procedure of w_{prefer} is presented in Fig. 9. In Fig. 9, let $T(w_i)$ be the test time of core *i* where a TAM width w_i and $A_i(k)$ indicate the result of $w_i(k) \cdot T(w_i(k))$, $(1 \le 2^k \le W)$. In our algorithm, $w_{prefer}(i)$ is set to the highest w_i in $\boldsymbol{R}_i$ such that the difference between $A_i(k)$ and $A_i(0)$ is less than the p% of $A_i(0)$. Furthermore, we assign a core, the TAM width providing the corresponding core with the best test time, if $T(w_{prefer})$ of the core is more than the q% of the total sum of $T(w_{prefer})$ of all embedded cores. The search space to find the best p and q is selected on an experimental basis. These heuristic approaches can make the proposed method applicable to various test circuits.

C. Multiple Bin Packing

The clock rate of a network is a factor n of the core test clock rate and n corresponds to the number of input and output pairs (see III.2 and V.1). In order to solve a multiple bin packing problem, we use an incremental packing on the basis of a 'cur_time.' The cur_time indicates the time trying to put a new rectangle into the bin with available space. First, the bin having the minimum cur_time among n bins is chosen to pack rectangles. If no rectangle is available to fill the bin in the cur_time, the cur_time of the current bin moves to the shortest end time in cores scheduled earlier within the bin. In this way, n bins are packed step by step.

D. Consideration of Multi-clock

In the domain of a bin packing problem, the increase of a test clock causes the width of a rectangle to decrease, but the height to increase in proportion to the decrease in width. For example, let the rectangle r_i operate f_T initially. If we assign $2f_T$ to r_i as a test clock, then the width and height of r_i becomes $t_i/2$ and $2w_i$, respectively. In this paper, we restrict the maximum test clock rate to $2f_T$ and consider two multi-clock modes similar to [21]. In one case, the test clock is predetermined. Only the cores fulfilling the condition that $T(W) = T(W/2)$ can be tested by the faster clock $2f_T$. In the other case, we assign the clock $2f_T$ to a core, as long as there are enough TAM lines available during the scheduling of the core. If TAM lines are short for the faster clock, the normal test clock f_T is tried instead. The detailed pseudo-code of the proposed scheduling procedure is shown in Fig.10.

VI. Experimental Results

We simulated four ITC'02 benchmark circuits [23] to evaluate the proposed scheduling algorithm. All of our simulations were conducted on a SUN UltraSPARC III with 1.2 GHz processors. Table 1 displays the experimental results where a common clock was used for the core test. Column 2 of the table shows the test configuration method related to the number of test sources and sinks as stated in section III.2. In order to make comparisons with previous results, we assumed that the network clock frequency f_N could increase up to four times as fast as the core test clock frequency f_T, that is, $n = f_N / f_T$ and n is an integer. Column 3 lists the results of packet-based test scheduling by Cota and others [18], and Columns 4 and 6 show the results of core-based test scheduling without on-chip clocking by Liu and others [20]-[21]. The network channel width is either 32 or 16 bits. Table 2 contains the scheduling results derived from two multi-test clock modes. We confined

```
1. Compute $\boldsymbol{R}_i$ from wrapper design of cores ;
2. while $p \le 50\%$   {
3.   while $q \le 30\%$ {
4.      Initialize C[m], cur_time[bin_pos] = 0, next_time[bin_pos] = 0, $w_{avail}$[bin_pos] = W;
5.      Calculate $w_{prefer}$ of C[m];
6.      while there exists a core not tested {
          Select TAM position, p, that has the lowest cur_time;
          // Start of rectangle packing heuristics for a bin p
          If ($w_{avail}$[p]  $\ge$  0) {
            If core i can be found, such that ($w_{prefer}$(i)*2  $\le$  $w_{avail}$[p]    AND T($w_{prefer}$(i))/2 is maximum) OR
                                            ($w_{prefer}$(i)  $\le$  $w_{avail}$[p]    AND T($w_{prefer}$(i)) is maximum)   {
              set bin_pos(i) AND clk_mode(i);
              Rectangle packing process with assigning $w_{prefer}$(i)*clk_mode(i) to cores; // refer to [7] about details
            }
            Else {
              If core i can be found, such that (bin_pos(i) =   p) AND (scheduled(i) =   YES) AND (completed(i) =   NO)
                                            AND (end_time(i) > cur_time[p])    AND end_time(i) is minimum    {
                next_time[p] = end_time(i);
              }
                If core i can be found, such that (scheduled(i) =   NO) AND
                                            ((T($w_{avail}$[p]/2) + cur_time[p])  $\le$  next_time[p] AND (T($w_{avail}$[p]/2) + cur_time[p]) is maximum) OR
                                            (T($w_{avail}$[p]) + cur_time[p])  $\le$  next_time[p] AND (T($w_{avail}$[p]) + cur_time[p]) is maximum)) {
                  set bin_pos(i) AND clk_mode(i);
                  find w(i), where w(i) is the highest Pareto-optimal width, such that w(i)  $\le$  $w_{avail}$[p]/clk_mode(i);
                  Rectangle packing process with rectangle insertion in idle time; // refer to [7] about details
                }
                Else {
                  If core i can be found, such that (bin_pos(i) = p) AND (scheduled(i) = YES) AND (completed(i) = NO) AND
                                            (begin_time(i) = cur_time[p]) AND ($w_{assigned}$(i)*clk_mode(i)  $\le$  $w_{avail}$[p])    AND
                                            (T($w_{assigned}$(i)) - T($w_{assigned}$(i)+$w_{avail}$[p]/clk_mode(i))) / clk_mode(i)) is maximum {
                    find w(i), where w (i) is the highest Pareto-optimal width, such that w (i)  $\le$  $w_{assigned}$(i) + $w_{avail}$[p]/ clk_mode(i);
                    Rectangle packing process with increasing TAM widths to fill idle time; // refer to [7] about details
                  }
                  Else {
                    set $w_{avail}$[p] = 0;
                  }
                }
            }
          }
          Else {
            move to next and update information; // refer to [7] about details
          }
        }   // End of rectangle packing heuristics for a bin p
        If max. cur_time[bin_pos] is less than current Best_test_time,
        Update Best_test_time = cur_time[bin_pos];
        Increase q by 5%;
     }
     Increase p by 5%;
  }
7. return Best_test_time;
```

Fig. 10. Procedure of test scheduling.

the test clock of all cores to f_T or $2f_T$ under the multi-clock mode. In case 1 in Table 2, we assigned a $2f_T$ clock to core i if $T_i(W) = T_i(W/2)$. On the other hand, the test clock f_T or $2f_T$ was dynamically chosen according to the TAM width available in case 2. In Table 2, we compared the test times of the proposed method with those of [21] using on-chip clocking.

Our algorithm significantly improves on earlier methods in all cases regardless of circuit sizes, channel width W, and network speed n (Tables 1 and 2). However, there are some cases in which the overall test time is not decreased despite the increase of the network clock rate. This is because the test time reaches the lower bound due to bottleneck cores. In order to

Table 1. Experimental results under single test clock mode.

Benchmark	No. of IO or n	$W = 32$			$W = 16$	
		[18]	[21]	Proposed	[21]	Proposed
D695 10 cores	2/2 or 2	26012	18869	13732	26200	23193
	3/3 or 3	20753	13412	9869	17807	16197
	4/4 or 4	14785	10705	9869	16197	12192
G1023 14 cores	2/2 or 2	31898	25062	14794	28635	17798
	3/3 or 3	22648	17925	14794	19620	14794
	4/4 or 4	18851	16489	14794	17925	14794
P22810 30 cores	2/2 or 2	315708	271384	138990	331672	249164
	3/3 or 3	222432	180905	102965	221134	177009
	4/4 or 4	170999	150921	102965	166800	145417
P93791 32 cores	2/2 or 2	-	611991	470011	935987	932380
	3/3 or 3	-	482352	341858	734390	623715
	4/4 or 4	435787	333091	259263	502876	470011

Table 2. Experimental results under multi-test clock mode.

Benchmark	No. of IO or n	Case 1				Case 2			
		$W = 32$		$W = 16$		$W = 32$		$W = 16$	
		[21]	Proposed	[21]	Proposed	[21]	Proposed	[21]	Proposed
D695 10 cores	2/2 or 2	15570	11978	22779	21571	14176	11013	24848	21460
	3/3 or 3	10705	9869	15223	15196	9073	8082	16226	15100
	4/4 or 4	10705	9869	14945	12192	9073	6096	15813	12081
G1023 14 cores	2/2 or 2	16286	9822	19888	17461	15662	8899	20482	17056
	3/3 or 3	10941	7550	13338	11462	11546	7397	14986	11756
	4/4 or 4	9163	7397	11172	10248	9788	7397	12581	9735
P22810 30 cores	2/2 or 2	187061	127480	270722	248382	177915	124463	261185	227127
	3/3 or 3	124787	93102	180977	170999	120840	91409	187663	152873
	4/4 or 4	99345	72981	150594	145417	98842	72981	150091	139320
P93791 32 cores	2/2 or 2	608809	470011	933119	932380	488171	457274	935343	912193
	3/3 or 3	479403	341858	732888	623715	391339	328248	786818	608504
	4/4 or 4	333091	259263	502876	470011	281518	238078	553860	468846

illustrate this point, we give an example of the scheduling result of P22810 in Fig. 11 when $W = 32$ and $n = 3$. In Fig. 11(a), the test time of core 1 does not decrease any more, even though the TAM width for core 1 is increased. Thus, core 1 becomes a bottleneck and dominates the overall test time of the system. In Fig. 11(b), since *clk_mode* of core 1 increases, the test time of core 1 is cut in half, and the system test time is diminished. In addition, the computation time of our algorithm is no more than one second in most cases. This is a very encouraging indicator for the practicality and feasibility of the proposed algorithm.

VII. Conclusions

In this paper, we proposed a new test scheduling algorithm for NoC-based SoCs using the rectangle packing solution used in testing common SoCs. In order to implement the rectangle packing approach on an NoC structure, we designed test resource configuration and test methods, such as test packet generation and routing. Furthermore, we extended the algorithm and improved test efficiency with multiple test clocks. Experimental results using some ITC'02 benchmark circuits showed that the proposed algorithm can provide

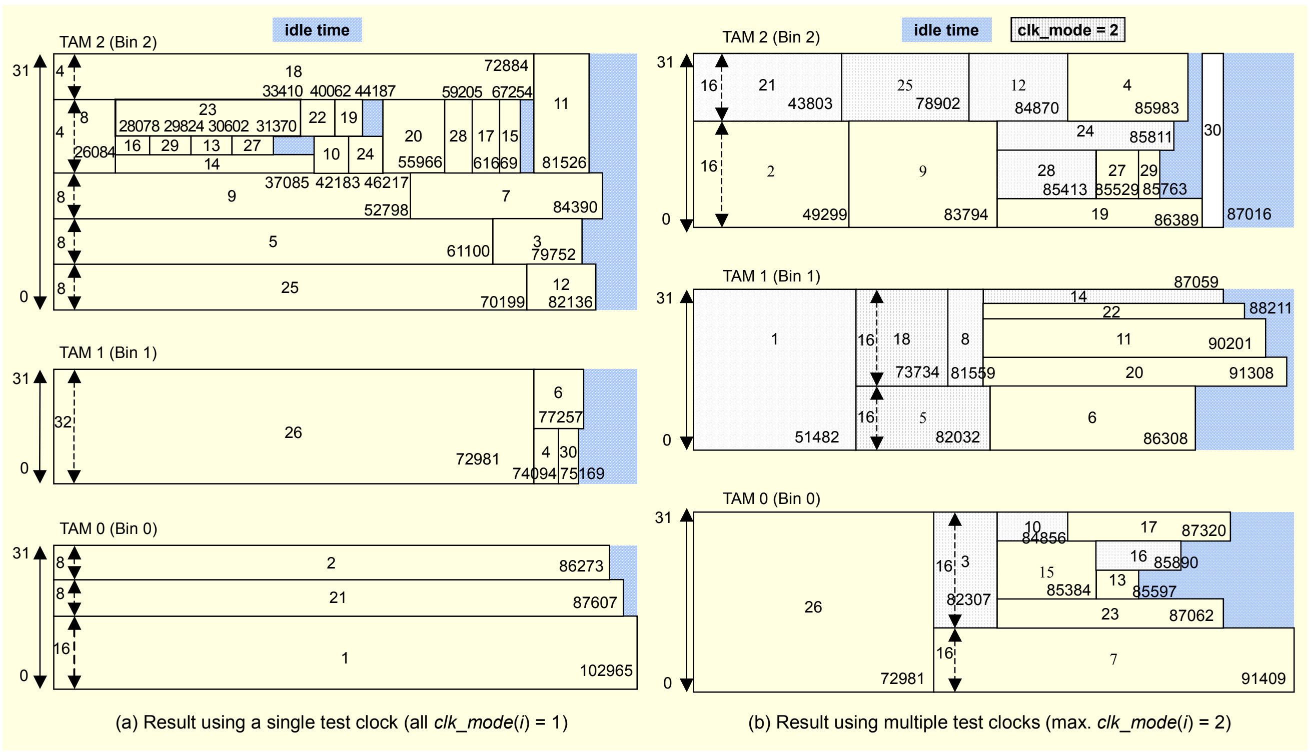


Fig. 11. Test scheduling results of P22810 ($W = 32$, $n = 3$).

superior results under all conditions.

Consequently, as the proposed scheduling algorithm and test platform seem to be very efficient and feasible for testing NoC-based SoCs, we expect further related research will be stimulated.

References

[1] L. Benini and G. D. Micheli, "Networks on Chips: A New SoC Paradigm," *IEEE Computer*, vol 35, Jan. 2002, pp. 70-78.

[2] A. Ivanov and G. D. Micheli, "The Network-on-Chip Paradigm in Practice and Research," *IEEE Design and Test of Computers*, Sep.-Oct. 2005, pp. 399-403.

[3] P. Guerrier and A. Greiner, "A Generic Architecture for On-Chip Packet-Switched Interconnections," *Proc. DATE,* Mar. 2000*,* pp. 250-256.

[4] B. Vermeulen, J. Dielissen, K. Goossens, and C. Ciordas, "Bringing Communication Networks on a Chip: Test and Verification Implications," *IEEE Commun. Magazine*, vol. 41, Sep. 2003, pp. 74-81.

[5] K. Chakrabarty, "Test Scheduling for Core-Based Systems Using Mixed-Integer Linear Programming," *IEEE Trans. on CAD*, Oct. 2000, pp. 1163-1174.

[6] Y. Huang, W.-T. Cheng, C.-C. Tsai, N. Mukherjee, O. Samman, Y. Zaidan, and S. M. Reddy, "Resource Allocation and Test Scheduling for Concurrent Test of Core-Based SOC Design," *Proc. ATS*, 2001, pp. 265-270.

[7] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "On Using Rectangle Packing for SOC Wrapper/TAM Co-optimization," *Proc. VTS*, 2002, pp.253-258.

[8] Y. Huang, S. M. Reddy, W.-T. Cheng, P. Reuter, N. Mukherjee, C.-C. Tsai, O. Samman, and Y. Zaidan, "Optimal Core Wrapper Width Selection and SOC Test Scheduling Based on 3-D Bin Packing Algorithm," *Proc. ITC*, 2002, pp. 74-82.

[9] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "Wrapper/TAM Co-optimization, Constraint-Driven Test Scheduling, and Test Data Volume Reduction for SOCs," *Proc. DAC*, 2002, pp. 685-690.

[10] E. Larsson and Z. Peng, "A Reconfigurable Power-Conscious Core Wrapper and its Application to SOC Test Scheduling," *Proc. ITC*, Sep. 2003, pp. 1135-1144.

[11] S. Koranne and V. Iyengar, "On the Use of k-tuples for SoC Test Schedule Representation," *Proc. ITC*, 2002, pp. 539-548.

[12] W. Zou, S. R. Reddy, I. Pomeranz, and Y. Huang, "SOC Test Scheduling Using Simulated Annealing," *Proc. VTS*, Apr. 2003, pp. 325-330.

[13] Y. Xia, M. Chrzanowska-Jeske, B. Wang, and M. Jeske, "Using a Distributed Rectangle Bin-Packing Approach for Core-based SoC Test Scheduling with Power Constraints," *Proc. ICCAD*, Nov. 2003, pp. 100-105.

[14] J. Im, S. Chun, G. Kim, J.-H. Ahn, and S. Kang, "RAIN (RAndom INsertion) Scheduling Algorithm for SoC Test," *Proc.*

ATS, Nov. 2004, pp 242-247.

[15] P. P. Pande, G. D. Micheli, C. Grecu, A. Ivanov, and R. Saleh, "Design, Synthesis, and Test of Networks on Chips," *IEEE Design and Test of Computers*, Sep.-Oct. 2005, pp. 404-413.

[16] M. Nahvi and A. Ivanov, "Indirect Test Architecture for SoC Testing," *IEEE Trans. on CAD*, vol. 23, no. 7, July 2004, pp. 1128-1142.

[17] E. Cota, M. Kreutz, C. A. Zeferino, L. Carro, M. Lubaszewski, and A. Susin, "The Impact of NoC Reuse on the Testing of Core-based Systems," *Proc. VTS*, Apr. 2003, pp. 128-133.

[18] E. Cota, L. Carro, F. Wagner, and M. Lubaszewski, "Power-Aware NoC Reuse on the Testing of Core-Based Systems," *Proc. ITC*, vol. 1, Sep. 2003, pp. 612-621.

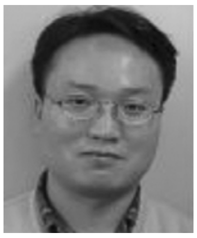
[19] A. M. Amory, E. Cota, F. Wagner, L. Carro, M. Lubaszewski, and F. G. Moraes, "Reducing Test Time with Processor Reuse in Network-on-Chip Based System," *Proc. SBCCI '04*, Sep. 2004, pp. 111-116.

[20] C. Liu, E. Cota, H. Sharif, and D. K. Pradhan, "Test Scheduling for Network-on-Chip with BIST and Precedence Constraints," *Proc. ITC*, Oct. 2004, pp. 1369-1378.

[21] C. Liu, V. Iyengar, J. Shi, and E. Cota, "Power-Aware Test Scheduling in Network-on-Chip Using Variable-Rate On-Chip Clocking," *Proc. VTS*, May 2005, pp. 349-354.

[22] J. Duato, *Interconnection Networks: An Engineering Approach*, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2003.

[23] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, ITC'02 SoC Test Benchmarks, http://www.hitech-projects.com/itc02socbenchm



Jin-Ho Ahn received the BS and MS degrees in electrical engineering from Yonsei University, Seoul, Korea, in 1997. He was a Research Engineer with DTV Lab in LG Electronics. He is currently working toward the PhD degree in electrical and electronic engineering at Yonsei University. His research interests focus on design and testing of NoC-based SoCs.

Sungho Kang received the BS degree from Seoul National University, Seoul, Korea, and the MS and PhD degrees in electrical and computer engineering from the University of Texas at Austin in 1992. He was a Research Scientist with the Schlumberger Laboratory for Computer Science, Schlumberger Inc. and a Senior Staff Engineer with the Semiconductor Systems Design Technology, Motorola Inc. Since 1994, he has been a Professor with the Department of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea. His main research interests include VLSI design and testing, design for testability, BIST, defect diagnosis and design for manufacturability.