Constraint-Driven Test Scheduling for NoC-Based Systems

Érika Cota, *Member, IEEE*, and Chunsheng Liu, *Member, IEEE*

***Abstract*—On-chip integrated network, the so-called network-on-chip (NoC), is becoming a promising communication paradigm for the next-generation embedded core-based system chips. The reuse of the on-chip network as test access mechanism has been recently proposed to handle the growing complexity of testing NoC-based systems. However, the NoC reuse is limited by the on-chip routing resources and various constraints. Therefore, efficient test-scheduling methods are required to deliver feasible test time while meeting all the constraints. In this paper, the authors propose a comprehensive approach to test scheduling in NoC-based systems. The proposed scheduling algorithm is based on the use of dedicated routing path that is suitable for nonpreemptive test. The algorithm is improved by incorporating both preemptive and nonpreemptive tests. In addition, BIST, precedence, and power constraints were taken into consideration. Experimental results for the ITC'02 system-on-chip benchmarks show that the nonpreemptive scheduling based on dedicated path can efficiently reduce test application time compared to previous work, and the improved method provides a practical solution to the real-world NoC-based-system testing with both preemptive and nonpreemptive cores. It is also shown that various constraints can be incorporated to deliver a comprehensive test solution.**

***Index Terms*—Network-on-chip (NoC), system-on-chip (SoC) testing, test access mechanism (TAM), test scheduling.**

I. Introduction

SYSTEM-ON-CHIP (SoC) has become a successful very large-scale integration (VLSI) design paradigm. The reuse of predesigned embedded cores has significantly reduced overall design period and cost. However, as the number of embedded cores in the system increases, the implementation of an efficient and effective communication architecture among cores is becoming the new bottleneck of SoC performance. Traditional broadcasting or shared-bus architecture has been shown unable to supply the new-generation SoC systems with both sufficient bandwidth and low latency under a stringent power-consumption limitation [8], [12], [23], [26].

Manuscript received May 24, 2005; revised September 27, 2005. The work of É. Cota was supported in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico. The work of C. Liu was supported in part by a University of Nebraska-Lincoln (UNL) faculty research fellowship and a Layman award. This paper was presented in part at the International Test Conference (ITC), pp. 1369–1378, 2004. This paper was recommended by Associate Editor K. Chakrabarty.

É. Cota is with the Programa de Pos-Graduacao em Computação, Instituto de Informatica, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil (e-mail: erika@inf.ufrgs.br).

C. Liu is with the Department of Computer and Electronic Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588 USA (e-mail: chunshengliu@unlnotes.unl.edu).

Digital Object Identifier 10.1109/TCAD.2006.881331

As an alternative, integrated on-chip networks have been proposed as the communication platform for complex core-based system, the so-called network-on-chip (NoC) [7], [8], [12], [23], [26], [28]. This new paradigm relies on an on-chip network, e.g., packet switching, to provide high-performance interconnection to embedded cores. It has been shown that NoC is superior to other traditional architectures, especially for systems containing a large number of cores with intensive communications load [23]. In view of the increasing complexity of the new designs, NoC can potentially become the preferred interconnection scheme for the next-generation SoCs. In fact, industrial NoCs have started to appear in literature [26], [28].

For the rest of this paper, we use the term "NoC" to denote an on-chip interconnection network (in this paper, a packet-switching network) consisting of routers, channels, and other network components. We use the term "NoC-based system" to denote the entire SoC system consisting of NoC and the functional embedded cores.

As in other core-based systems, testing of embedded cores remains a challenge in NoC-based systems. In traditional SoCs, test data are transported through a dedicated test access mechanism (TAM). In an NoC-based system, however, the implementation of network components (routers, channels, etc.) has already imposed a considerable amount of area overhead to the system. Therefore, testing of cores in an NoC-based system should rely on the reuse of the existing resources without introducing new overhead.

Previous work [5], [13] has attempted to reuse the functional interconnections as TAM. However, this leads to additional hardware overhead. Reusing the on-chip network as TAM, on the other hand, does not require extra hardware, hence, it provides a cost-efficient solution to the NoC-based system testing.

The reuse of on-chip network as TAM in NoC has been first proposed in [24] and further improved in [29]. Through the reuse method, testing time can be significantly reduced, while other cost factors such as pin count and area overhead are strongly optimized. Previous work has explored the use of network to maximize the test parallelism, i.e., the available communication resources (channels, I/O ports) are utilized to maximize the test data throughput. In an NoC-based system, test data are organized into test packets, which are routed through the network. One test packet contains one test vector or one test response. Packets are scheduled to maximize the network usage and to reduce the system test application time. The impact of core placement in the network and other design characteristics has also been studied [29].

Previous work [24], [29] has assumed that core testing is preemptive, where test packets can be scheduled individually

and core-testing pipeline can be interrupted. In practice, however, testing of some cores may be nonpreemptive, particularly for BIST and sequential core test [19]. Therefore, a practical test plan should be able to schedule both preemptive and nonpreemptive of tests in a way that the overall testing time is minimized. In addition, previous work has only studied the scheduling under power constraints. A comprehensive test solution should consider a more complex scenario where more constraints and test schemes are incorporated.

In this paper, a comprehensive test-scheduling approach for NoC-based systems is proposed. The contributions of this paper are as follows.

1) We present a scheduling method based on the use of a dedicated routing path for the test of each core. Test pipeline is maintained, and preemption is not required. We proved that both the packet-based scheduling and the dedicated-path scheduling are NP-complete problems.
2) We improve the current scheduling methods by incorporating both preemptive and nonpreemptive tests to handle the NoC-based systems consisting of both types of cores.
3) We incorporate the support for BISTed cores. The BIST engines can be either dedicated to each core or shared by several cores.
4) We include power constraints as well as precedence constraints in the scheduling method.
5) We obtain optimized test application time of the NoC-based system under the above conditions.

Note that this paper does not describe the test of the NoC (channels and routers) itself; we, instead, focus on the test of the functional cores, and we assume that the network components have already been tested *a priori* as fault free. Testing the NoC itself has been studied in [10] and [26].

This paper is organized as follows. In Section II, we review some prior work. Section III presents some basics of NoC. In Section IV, we briefly explain the concepts and notations of the reuse techniques proposed in [24] and [29]. Section V introduces the scheduling method for nonpreemptive test using dedicated routing path. Then, in Section VI, we present a method that allows both nonpreemptive and preemptive tests, and includes BISTed core tests, power, and precedence constraints into the schedule. Finally, experimental results for the ITC'02 SoC Test Benchmarks are presented in Section VII. Section VIII concludes the paper.

II. Prior Work

Efficient test scheduling has been the focus of many early studies in core-based system testing. A number of scheduling algorithms have been proposed in literature [6], [17], [20], [21], [25], [31] assuming a dedicated TAM. The implementation of a cost-effective test bus is proposed in [15] and [16], where power, area, and system constraints are considered in addition to the testing-time minimization. Other works propose the approaches of using switching architectures to optimize the usage of test bandwidth and test control [11], [14].

Previous works [5], [13] have also attempted to reuse the functional interconnections as TAM for traditional SoCs. However, this leads to additional hardware, e.g., large number of multiplexers and special test registers, which are needed to implement extra functions required for reuse. Moreover, they are inflexible because any change on cores can lead to partial or overall redesign of the TAM.

Test of embedded cores through a packet-switching architecture has been proposed in [9] and [10]. Aktouf proposes in [10] a test strategy for an on-chip homogeneous multiprocessor architecture, where the processors are interconnected through an on-chip switching network. Nahvi and Ivanov propose in [9] the use of a packet-switching communication-based TAM for SoC. The proposed TAM model, the so-called NIMA, is defined to allow modularity, generality, and configurability for the test architecture. Such an architecture is very similar to a functional NoC, but it is specifically designed for testing. Hence, routing and addressing strategies are designed and optimized for test mode instead of mission mode. The NIMA network is further developed and presented in [30].

Industrial NoC-based systems have been introduced through several implementations. Vermeulen *et al.* present an NoC-based system in [26] and suggest some options for cores and on-chip network components to be tested and verified. It is shown that the NoC architecture can facilitate the access to the network components and the embedded cores during both manufacturing test and system verification. In [28], a methodology for automatically generating energy models for a versatile and parametric on-chip communication architecture (STBus) is presented. A software simulator is developed and applied to a real-world NoC-based multiprocessor system.

Reusing the on-chip network as TAM in NoC does not require extra hardware, since the functional cores are already adapted to the network by network interfaces. Therefore, the reuse approach provides a cost-efficient solution to the NoC-based system testing.

Recently, an NoC-based test method by reusing the on-chip network was proposed in [24]. In this paper, test patterns and corresponding test responses for each core are organized into sets of packets. Cores are sorted in decreasing order of test data size, and access paths to cores using the existing network channels are sorted in increasing order of path length. The algorithm schedules the packets using the available network resources (channels, I/O ports) such that the overall test application time is minimized. Since scheduling is performed on every single packet, the test pipeline for a core can be interrupted, and test preemption is allowed. This scheduling algorithm is further augmented in [29] by increasing the parallelism of test data transportation and including power constraints. However, the algorithm is not directly applicable to cores for which test preemption is either impossible or undesirable [19].

III. Background: NoC

NoCs typically use the message-passing communication model. Cores attached to the network communicate by sending request and receiving response messages. To be routed by the network, a message is typically composed by a header, a payload, and a trailer. The header and the trailer frame the packet, and the payload carries the data being transferred. The header also carries the information needed to establish the path

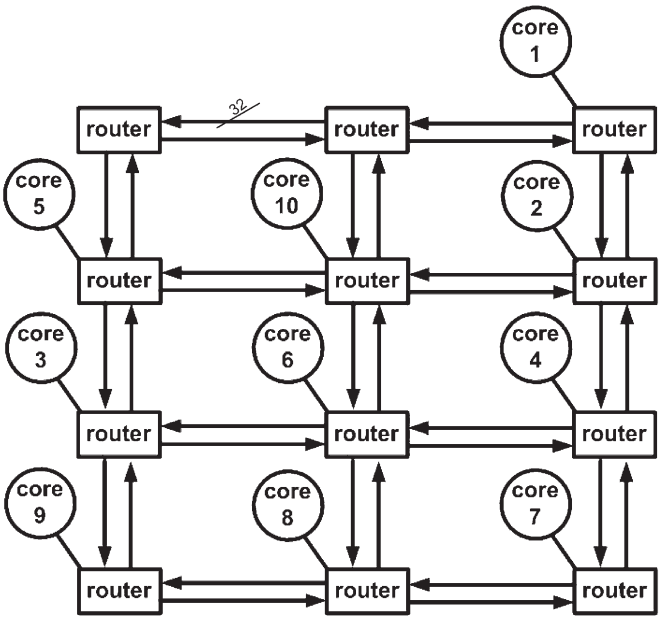


Fig. 1. System d695 implemented in grid SoCIN NoC [27].

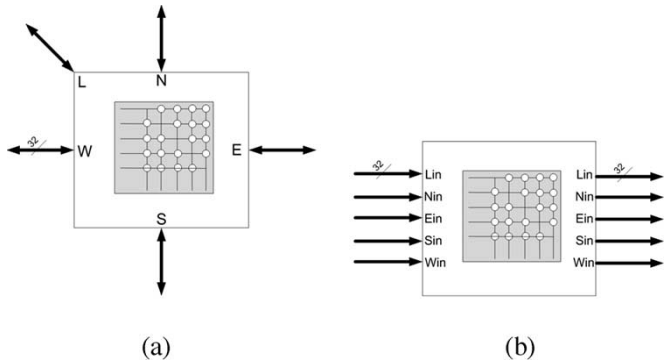


Fig. 2. Basic structure of SoCIN router: (a) interface and (b) architecture.

between the sender and the receiver. Depending on the network implementation, messages can be split into smaller structures, the so-called packets, which can be individually routed. Packet-based networks present a better resource utilization, since packets are shorter and reserve a smaller number of channels during transportation compared to a whole piece of message.

Besides its topology, an NoC can be described by the approaches used to implement the mechanisms for flow control, routing, arbitration, switching, and buffering [3]. Flow control deals with data traffic on the channels and inside the routers. Routing is the mechanism that defines the path a message takes from a sender to a receiver. The arbitration establishes priority rules when two or more messages request the same resource. Switching is the mechanism that accepts an incoming message of a router and sends it to an output port of the router. Finally, buffering is the strategy used to store messages when a requested output channel is busy. Current cores usually need to use wrappers to adapt their interfaces and protocols to the on-chip network.

In this paper, we base our analysis on a packet-switching network model, so-called SoCIN, introduced in [27]. It is implemented in a two-dimensional mesh topology. Fig. 1 shows the implementation of the system d695 from the ITC'02 SoC Test benchmarks suite [22] in this topology. The communication channels between two adjacent routers are defined to be 32-bit wide. Each router in the SoCIN network is composed of five input and five output ports, as shown in Fig. 2(a). One pair of input/output ports is dedicated to the connection between the router and the core, while the remaining four pairs connect the router with the four adjacent routers, as depicted in Fig. 2(b). An SoCIN router is implemented using from 3000 to 6000 gates, depending on the bitwidth of the network channel and depth of the input buffers [27]. For simplicity of router implementation, the channel between a router and its associated core is similarly defined. The network uses credit-based flow-control and XY routing, where a packet is first routed on the X-direction and then on the Y-direction before reaching its destination. Switching is based on the wormhole approach, where a packet is broken up into multiple flits (flow-control units). Flit is the smallest unit over which the flow control is performed, and its size equals the channel width.

IV. Reusing NoC as TAM

In order to reuse the on-chip network as the TAM for the embedded cores, the test vectors and test responses of each core must be expressed as a set of packets to be transported through the network. In addition, the wrapper that adapts the core to the network must be modified to correctly send/receive the test data to/from the test interface of the core (scan controls, scan data pins, functional pins).

NoC-based system requires additional functionalities on core wrapper besides the original P1500 compliant architecture. In order to impose a minimum effect on the original design to reduce the cost, the test packets are designed in such a way that each flit arriving from the network is unpacked in one cycle. That is, each bit in a packet flit fills exactly one bit of the wrapper scan chains of the core. Functional inputs and outputs of the core, as well as the internal scan chains, are concatenated into wrapper scan chains of similar length, such that the channel width is adequate for transporting one bit to each scan chain. Fig. 3(a) and (b) depicts a wrapper configuration during normal operation and during test, respectively. The area overhead due to the implementation of the test mode in the wrapper is comparable to the overhead of a basic P1500 compliant wrapper, and the number of wrapper scan chains is determined by the algorithm proposed in [18]. The new control signals of the NoC wrapper are included in the input and output control of the basic P1500 wrapper. Two additional registers are required to store the address of the destination for test response and the delivery time of the response packets, respectively.

Control information, such as scan shift and response capture signals, is also delivered in packets, either in the test header (to be interpreted by the wrapper) or as specific bits in the payload (for direct connection to the target pins). In Fig. 3(b), the latter approach is assumed. In both cases, a test enable signal in the packet header indicates that a test configuration will take place in the wrapper. The original buffer structure of the router,

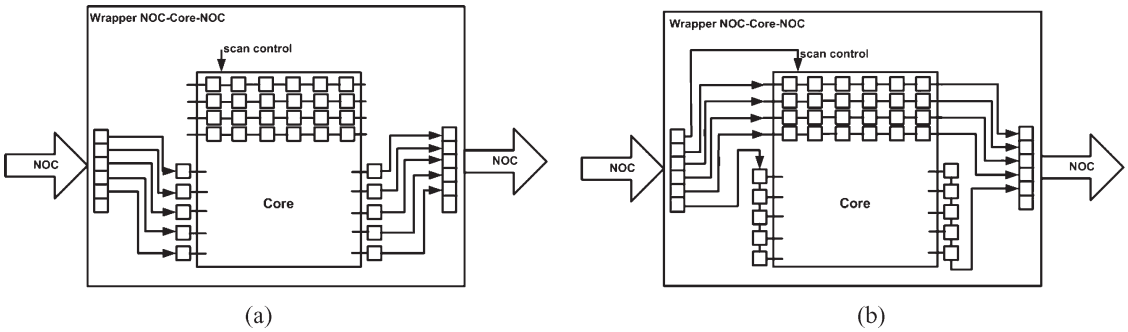


Fig. 3. Wrapper configurations: (a) normal mode and (b) test mode.

which is designed based on the mission-mode requirements, is reused as is to reduce the area overhead.

In this paper, we consider the scenario where an external tester is connected to the functional interface of the system. The input and the output ports of the network can be reused for the transmission of test packets (test vectors and responses) to/from all cores. Note that although testing the network components (routers and channels) concurrently with testing the embedded cores is possible, it is not addressed in this paper. We assume here the on-chip network has been tested *a priori* as fault-free by additional test process, hence, the network is in normal mode while cores are in test mode. The same protocol used for functional communication can be reused during test.

V. Test Scheduling Using Dedicated Routing Path

If a core allows preemptive testing, each test vector or the corresponding response for this core can be delivered as an individual packet using any available path. In this packet-based routing, test pipeline may be interrupted, as shown in [24] and [29]. However, if the core does not allow preemptive testing, e.g., BISTed cores, all the test vectors must be delivered as a consecutive series of packets without being interrupted. We discuss in this section the nonpreemptive test scheduling based on the use of dedicated routing path.

A. Dedicated Routing Path

As discussed earlier, the test-scheduling algorithms proposed in [24] and [29] assume that tests for the cores in NoC-based systems are preemptive, hence, provide a high level of flexibility to scheduling. However, preemptive testing is not always possible in practice, especially for BIST and sequential core test [19]. In addition, it is always desirable that the test pipeline of the core is not interrupted, i.e., the nth test vector will be shifted into the scan chains as the $(n-1)$th test response is shifted out. However, in case of preemption, the test pipeline has to be halted if either the test vector or test response cannot be scheduled due to the unavailability of test resources, i.e., channels and input/output ports. This may not only increase the complexity of wrapper control but also cause potential increase on test time.

The adverse effect on the test time can be explained by the following example. Let core 1 has higher priority over core 2 in the scheduler of [24], and test vector 1 of core 1 is first scheduled on an input port. We assume that the test vector 1 of core 2 cannot be scheduled because all inputs are now being used. We also assume a case that after vector 1 of core 1 is scheduled, vector 2 of core 1 cannot be scheduled on this input immediately because the test response of vector 1 cannot be scheduled on any output port due to resource conflict. At this point, test vector 1 of core 2 can be scheduled on this free input. However, it is possible that immediately after this point, the output is available for scheduling the test response of core 1, which is ready. However, the next vector for core 1 cannot be scheduled because the available input is being used by core 2. Hence, the pipeline of core 1 is halted. It is easy to see that in the worst case, the test of core 1 can always be interrupted by core 2, which increases the test time of core 1. If this occurs on several cores, the network parallelism may not overcome the individual test-time increase, and the total test time is not well optimized.

Another disadvantage of preemptive scheduling is on its complexity. The flexibility of handling every single packet leads to a vast solution space for test-scheduling algorithms. In an acceptable amount of simulation time, only a small portion of the solution space can be explored. Therefore, highly optimized solution may not be found, and test time may be compromised.

On the other hand, if the test pipeline is maintained, the preemption is not required. Moreover, the test wrapper can remain unchanged, and the test time can be potentially reduced. Therefore, a nonpreemptive scheduling for NoC-based system can be applied. In this approach, the scheduler will assign each core a routing path, including an input port, an output port, and the corresponding channels that transport test vectors from the input to the core, and the test response from the core to the output. Once the core is scheduled on this path, all resources (input, output, channels) on the path will be reserved for the test of this core until the entire test set is completed. Test vectors will be routed to the core, and test responses to the output in a pipelined fashion. Therefore, in this dedicated-path approach, the test of a core is identical to a normal test, and the flow control becomes similar to circuit switching. Fig. 4 shows the dedicated routing path of the system in Fig. 1 using XY routing for core 10 and core 8, respectively, using two input/output ports.

Note that compared to the preemptive scheduling, the dedicated-path approach handles a whole test set instead of every single packet, hence, creates a much smaller solution space for test-scheduling algorithms. As a result, a significantly larger portion of the solution space can be explored by the algorithm, and the result can be better optimized.

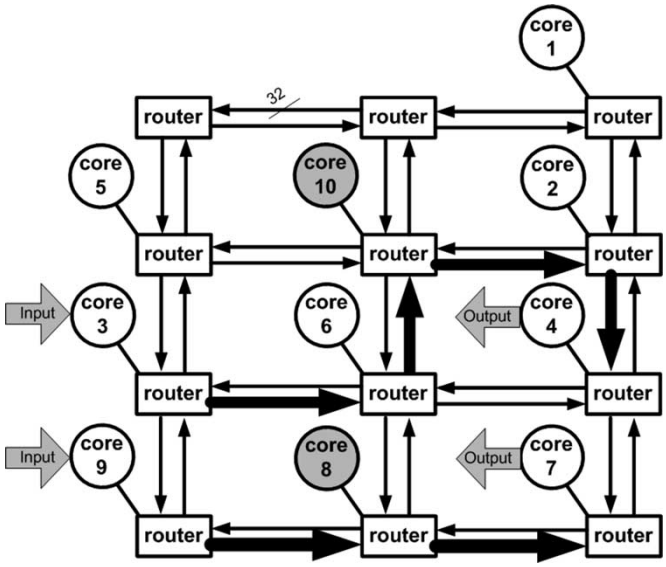


Fig. 4. NoC-based d695 with two routing paths scheduled.

B. Problem Formulation

It can be seen that in this dedicated-path approach, the problem of NoC-based scheduling is how to efficiently assign input/output pairs to cores without resource conflicting, such that the overall test time is minimized. It can be formally stated as follows.

> In an NoC-based system using dedicated routing path for testing, given $N_{\rm c}$ cores, $N_{\rm i}$ inputs, $N_{\rm o}$ outputs, routing algorithm, and the network topology, determine an assignment of cores to input/output pairs such that the total test time is minimized.

Note that this problem is equivalent to the well-known resource-constraint multiprocessor scheduling problem. The decision version of the resource-constraint multiprocessor scheduling problem can be stated as follows.

Instance: Set T of tasks, each having length $l(t)$, number $m \in Z^+$ of processors, number $r \in Z^+$ of resources, resource bounds $B_{\rm i}$, $1 \leq i \leq r$, resource requirement $R_{\rm i}(t)$, $0 \leq R_{\rm i}(t) \leq B_{\rm i}$, for each task t and resource i, and an overall deadline $D \in Z^+$.

Question: Is there an m-processor schedule σ for T that meets the overall deadline D and obeys the resource constraints, i.e., such that for all $u \geq 0$, if $S(u)$ is the set of all $t \in T$ for which $\sigma(t) \leq u < \sigma(t) + l(t)$, then for each resource i, the sum of $R_{\rm i}(t)$ over all $t \in S(u)$ is at most $B_{\rm i}$?

It can be easily proved using restriction that for $m \geq 2$, the above general resource-constraints problem is NP-complete [1], [2]. The equivalence between a decision version of NoC-based scheduling problem and the resource-constraint multiprocessor scheduling problem can be easily established by observing the correspondence between processors and input/output pairs, between tasks and test sets, and between resources and routing resources including channels, input, and output ports. The resource bounds in NoC-based scheduling are $B_{\rm i} = 1$, and the deadline D corresponds to the overall test application time.

It can be easily proved that the packet-based preemptive scheduling can be reduced to dedicated-path scheduling and, hence, is also NP-complete using the method of restriction [2]. We apply two restrictions on the packet-based preemptive scheduling as follows. 1) For each core, its test-vector packets must be sent continuously in order without being interrupted by packets of other cores. This also implies that the response packets will be sent in the same manner. This restriction corresponds to a set of precedence constraints applied to the packets of all cores. 2) For each core, all test-vector packets must use the same input port, and all response packets must use the same output port. This restriction also implies that the input and output routing paths are deterministic for each core, given an input/output pair. Under these strong restrictions, the packet-based preemptive scheduling is reduced to the dedicated-path scheduling and, hence, is NP-complete [2]. The latter is, in fact, a special case of the former under restriction.

We have developed an integer linear-programming model to solve the dedicated-path-scheduling problem exactly for NoC-based systems of small size. (We have omitted the detailed results in this paper.) However, the computation time of the integer linear programming (ILP) solution is prohibitively long due to the large number of constraints. We have observed in the experiments on a Sun Blade2000 workstation that even a fairly small instance of NoC-based system takes hours of CPU time to solve. Although the computation time can be significantly reduced by using enumeration [19], the ILP method can only be used for small NoC-based instances. We have observed that if the number of cores is larger than seven, the computation time becomes unacceptable. This is in part due to the numerous constraints representing the possible conflicting routing paths. Since an NoC-based system usually contains a large number of cores, efficient heuristic scheduling algorithms are necessary in practice instead of ILP.

C. Efficient Scheduling Heuristic

Here, we present an efficient heuristic scheduling algorithm based on the use of dedicated routing path. The algorithm is similar to the one proposed in [29] in that it also sorts the cores in decreasing order of test time. However, we do not try to allocate the shortest path to the largest core as in [29]. This is because in a dedicated-path approach, all tests are applied with full pipeline, and the distance from I/O ports to cores becomes less important. Instead, we assign larger cores to the first available I/O pair, hoping that by finishing the larger tests earlier, resources can be released, and more effectively utilized by smaller cores. The algorithm maintains a time tag on every resource (channels, input, and output ports) indicating its availability. Once a routing path for a core is determined and allocated, all related resources are reserved for the core, and the time tags are updated.

The pseudo-code of the algorithm is sketched in Fig. 5. The heuristic starts by creating an ordered core list as well as an I/O-pair list (Lines 1 and 2). The orders of I/O pairs in the list are permuted, and every permutation is attempted. Different permutations represent different priorities of the I/O pairs when more than one I/O pairs are free to be assigned to a core. For

```
Procedure NoC_schedule
1 Start with sorted cores in decreasing order of test time;
2 Permute all possible orders of I/O pairs;
3 For every permutation
4   While there are unscheduled cores
5     For each unscheduled core
6       Find a free I/O pair;
7       If no free I/O pair
8         Update current time, repeat from 4;
9       else
10        Check the corresponding routing path;
11        If path is blocked
12          If all cores have been attempted
13            Update current time, repeat from 4;
14          else
15            try next core in the list;
16        else
17          assign core to the path, update time labels;
18 Repeat from 2 for a user-defined number of cores permutations;
```

Fig. 5. Pseudo code of heuristic scheduling algorithm.

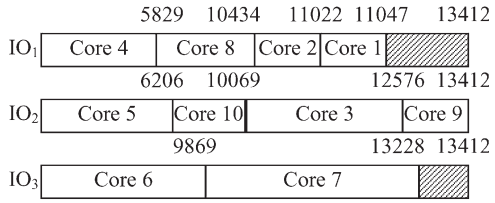


Fig. 6. Scheduling result of d695 using dedicated-path heuristic.

each permutation, we attempt to assign the core on top of the list to the first available I/O pair (Line 6). The availability of I/Os is examined by checking their time tags. If no I/O is available, the current time has to be updated by the next most recent time tag (Line 8), and the cores will be attempted once again; otherwise, a routing path is created, and the algorithm will check if it is blocked by other paths currently being used (Line 10), i.e., resources (channels, input, or output ports) conflict. If there exist one or more conflicts, either the next core is attempted (Line 15), or all unscheduled cores have to be attempted again (Line 13). Otherwise, the core is scheduled on the I/O, and the corresponding resources will be updated with new time tags (Line 17). The core is then removed from the list, and the next core is attempted for schedule. The whole procedure is repeated for different permutations of the core list, and this number is defined by the user.

The execution of this heuristic requires only a few milliseconds of CPU time even for the largest system. The complexity of the algorithm is $O(M!N_c)$, where N_c is the number of cores in the system, and M is the number of input/output ports. In order to explore a larger portion of the solution space, we can further apply a permutation on the order of cores on top of the above pseudocode. In this paper, we limit the number of such permutations to 200, and the execution takes only a few seconds.

In Fig. 6, we illustrate the result given by the above algorithm for benchmark d695 with three inputs and three outputs. Each test requires a set of channels and a pair of I/Os, which are denoted by IO_1, IO_2, and IO_3. We show the cores that are scheduled on the I/Os and the corresponding start times (not drawn to scale). The shaded areas represent idle time. It can be seen that the algorithm has successfully moved the idle time to the end of the test procedure on each I/O, by which the test time is optimized.

VI. Scheduling of Both Preemptive and Nonpreemptive Tests Under Constraints

Although the dedicated-path approach maintains test pipeline, it suffers from the lack of flexibility, i.e., the minimum manageable unit in test scheduling is the full test application time of a core. In practice, however, it is more feasible to assume that some cores require nonpreemptive scheduling for maintaining test pipelines, while others can be tested preemptively. This is important for a comprehensive test-scheduling algorithm. One of the examples is that during test application, excessive power dissipation and bad heat transfer can cause some cores to be significantly hotter than others, the so-called hot spots [32], [33]. Applying the entire test suite continuously can lead to dangerous temperature on these cores and cause damage. In this case, the test suite can be split into several test sessions (or even single test vector in the extreme case) that can be scheduled individually. We allow sufficient time between test sessions so that the core can be cooled down via heat conduction and convection.

It can be seen from the previous example that the nonpreemptive scheduling is not suitable for this case because it will occupy the routing path during the entire test application of the core, and extra time between test sessions will leave the path idle without being used by other cores. However, preemptive scheduling can easily handle this situation in nature [29].

Therefore, a more practical method should be able to handle both types of cores in the same schedule. In this section, we first discuss the inclusion of some practical factors such as multiple test sets and constraints. We also introduce the calculation of power dissipation when power constraint is considered. We then present a new scheduling method that can handle both preemptive and nonpreemptive tests simultaneously in the presence of multiple test sets and constraints. It can be easily proved using the method of restriction, the same as shown in Section V-B, that this combined scheduling is also NP-complete.

A. Multiple Test Sets and Constraints

The algorithm presented in Section V-C is focused on external tests. In practice, multiple test sets are often needed to test complex cores, e.g., cores are tested by both BIST and external test sessions. In addition, some precedence constraints may be required to impose a partial order among the tests

for various reasons [19]. For example, since BIST test can be applied at a much higher speed than external test, it is common to first apply BIST to target the random-detectable faults and then use external test to target the random-resistant faults. It may also be desirable to test the memory cores earlier because they can then be used to test logic cores. Moreover, since cores that occupy larger chip area are more likely to have defects caused by processing, it may be more desirable to test these large cores first [19]. Therefore, including BISTs and various precedence constraints can make the scheduling algorithm more practical and potentially increase the efficiency of the entire test procedure.

Another important consideration is power constraint. Scheduling algorithm has to guarantee that the power dissipation at any particular time is under a predefined limit of safety. We show the calculation of power dissipation in NoC-based systems in the next section.

B. Calculation of Power Consumption

In the proposed NoC architecture, we consider power consumption from four sources: cores, wrappers, routers, and communication channels.

Equations (1) and (2) give the consumption per cycle of a network router and a communication channel, respectively, for the transmission of a single packet. C_L, T, and σ represent the load capacitance (technology-dependent constant), clock period, and the switching factor, respectively. Variables $\mathrm{nb}_{\mathrm{ff}}$, nb_{gt}, σ_{ff}, and σ_{gt} represent the number of active components (either flip–flops or logic gates) and their corresponding switching factors in the router, respectively, when one packet is being routed. Note that for the flip flops, there is a constant switching factor caused by the clock in addition to the switching of the flip–flop itself

$$P_{\mathrm{router}} = C_L * Vdd^2 * \frac{1}{T} * [(\sigma_{\mathrm{ff}} + 1) * \mathrm{nb}_{\mathrm{ff}} + \sigma_{gt} * \mathrm{nb}_{gt}] \tag{1}$$

$$P_{\mathrm{channel}} = C_L * Vdd^2 * \frac{1}{T} * \sigma_{\mathrm{w}} * (\mathrm{ch}_{\mathrm{l}} * \mathrm{wire}_{\mathrm{w}} * \mathrm{ch}_{\mathrm{w}}). \tag{2}$$

In (2), the load capacitance of the channel is given by the product of the number of wires in the channel $(\mathrm{ch}_{\mathrm{w}})$, the length of the channel $(\mathrm{ch}_{\mathrm{l}})$, and the width of the wire $(\mathrm{wire}_{\mathrm{w}})$. Variable σ_{w} is the switching factor for the wire. In our approach, all channels are assumed to have the same length, although this may not be the case in real world. Since the power consumption is calculated per cycle, the size of the packet is less important.

The total power consumed by transmitting a packet is calculated according to the path established in the network for the packet. That is, for each router and each channel active in the path, their corresponding power consumptions are summed up, as shown in (3), where $\mathrm{nb}_{\mathrm{routers}}$ is the number of routers, and $\mathrm{nb}_{\mathrm{channels}}$ is the number of channels in the path.

$$P_{\mathrm{packet}} = \mathrm{nb}_{\mathrm{routers}} * P_{\mathrm{router}} + \mathrm{nb}_{\mathrm{channels}} * P_{\mathrm{channel}}. \tag{3}$$

The power consumption of a core during test depends on core design, test vectors, and the order of test vectors. In this paper, we do not present any power-consumption model for cores. Rather, we assume that the power consumption of each core during test is provided by the core designer. Moreover, since wrapper is usually developed for a specific core, we assume that the wrapper power consumption is known as well.

Also, in this paper, we consider the peak power consumption during test application for each core and its wrapper. In addition, the calculation of power can be performed on every cycle, so that it becomes independent of the test clock frequency. We note that if there is enough information for each core, a more accurate power profile can be used in the proposed method.

C. Improved Scheduling With Both Preemptive and Nonpreemptive Tests

We now present a scheduling algorithm that can handle both preemptive and nonpreemptive tests. The algorithm is based on the combination of packet-based and dedicated-path routing. For practical purpose, we also take into account multiple test sets, as well as precedence and power constraints, as discussed.

Let the input of the scheduling heuristic be the following.

1) A set $\mathbf{C} = \{i, 1 \leq i \leq N_{\mathrm{c}}\}$ of cores in an NoC-based system.
2) A set $\mathbf{T} = \{T_{j_{\mathrm{i}}}, 1 \leq j \leq N_{t_{\mathrm{i}}}, 1 \leq i \leq N_{\mathrm{c}}\}$ of test sessions defined for all cores in $\mathbf{C}$. Each core i may have a total of $N_{t_{\mathrm{i}}}$ test sessions. Each test session $T_{j_{\mathrm{i}}}$ can represent a BISTed, external, preemptive, or nonpreemptive test for core i.
3) A set of six tuples $\mathbf{I} = \{(\mathrm{wc}, \mathrm{cl}, p, \mathrm{pwr}, \mathrm{preemp}, \mathrm{payload})_k, 1 \leq k \leq |\mathbf{T}|\}$, representing the number of wrapper scan chains, maximum length of the scan chains, number of test patterns, power consumption during test, type of testing (preemptive or nonpreemptive), and payload size of the test packet, respectively, for each test session k in $\mathbf{T}$.
4) A set $\mathbf{Prec} = \{(p_1, \ldots, p_n)_k, 1 \leq n, k \leq |\mathbf{T}|\}$ of precedence constraints for each test session k. The precedence list indicates which test sessions must be finished before test session k is scheduled.
5) A graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with the set of vertices $\mathbf{V}$ representing the routers and the set of arcs $\mathbf{E}$ that connect two vertices representing the communication channels of the network. Each vertex has a list of associated cores, representing the cores connected to the router. Note that more than one core can be associated with a router, indicating either a mega core or a cluster of cores. In this case, each core can be tested independently, but all of them are accessible through the same router. On the other hand, a router can have no associated core when the network has more routers than cores.
6) A list of I/O ports corresponding to some vertices in graph $\mathbf{G}$, indicating that these cores can be used as I/O ports during testing.

As in the dedicated-path-based scheduling, each channel in the network is assigned a time tag, indicating the time when the channel is free for use. However, since preemptive scheduling is allowed here, the scheduler has to be able to handle each single packet and the corresponding latency. Equation (4) defines the

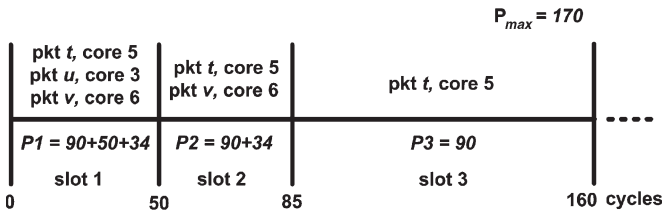


Fig. 7. Scheduling process considering power constraints [29].

time required to transmit a packet through a path. T_{router}, Nb_{routers}, T_{headers}, and payload represent the number of cycles spent by the packet header in each router to establish the path, the number of routers in the path, the clock cycles required to pack and unpack the header, and the number of flits carrying test data in the packet, respectively. In SoCIN network, we use $T_{\text{router}} = 3$ and $T_{\text{header}} = 1$, and a test header is assumed in addition to the packet header

$$T_{\text{packet}} = T_{\text{headers}} + T_{\text{router}} * Nb_{\text{routers}} + \text{payload}. \quad (4)$$

For each packet, two extra cycles are added to (4) during which the test vector can be applied to the core, and the wrapper is ready to pack and deliver the response packet. Packets belonging to the preemptive test sessions have a payload with the number of flits equal to the length of the longest wrapper scan chain of the core under test. Payload of packets of nonpreemptive tests, on the other hand, have the size corresponding to the sum of the flits of all test vectors defined in the test session.

The new test scheduling is an improved version of the list-scheduling algorithm [4]. The main idea is to process the available time instant in an increasing order (starting at zero) and schedule as many packets as possible at a certain instant before moving to the next. The ready list L_t contains the packets that can be scheduled at a given instant of time, and the algorithm will associate each packet with an available path for a certain amount of time. Initially, L_t contains the packets corresponding to the first test vector of each core, with the consideration of precedence constraints and different types of test sets allowed.

The schedule is defined as a set of time slots of different sizes, as shown in Fig. 7 [29]. Each time slot contains a set of packets being transmitted (each one associated to a core), and the size of a slot is measured by the number of clock cycles needed to apply the test vectors in the packets. Pi represents the power dissipation caused by the packets scheduled in the corresponding slot. The end of a slot indicates either the completion of a packet or the beginning of a new packet. The transmission of one packet may be distributed over several slots, and slots can be modified as the schedule proceeds (one slot can be broken up into two new slots, such that a new transmission can be scheduled to start or finish in the middle of the original slot). The schedule is initialized by a single time slot starting at cycle zero, with ending time being infinite. For example, the schedule depicted in Fig. 7 was generated as follows.

1) Packet t associated to core 5 is allocated in the single slot of the initial schedule. The duration of this packet, calculated by (4), is 160 clock cycles. Thus, the original single slot $[0, \ldots, \infty]$ is divided in two: Slot 1 is now $[0, \ldots, 160]$, and slot 2 is $[161, \ldots, \infty]$. Next, the power consumption of packet t is added to slot 1.
2) Then, packet u associated to core 3 is scheduled with its duration of 50 clock cycles. The total power consumption of packets t and u is within the specified power limit ($P_{\max}$ in Fig. 7), and packet u can be scheduled in the same slot of packet t. However, since the duration of packet u is smaller than the duration of slot 1, this slot is again divided in two, creating a new slot 2 from cycle 51 to cycle 160. Power consumption of packet u is added to the new slot 1.
3) Finally, packet ν of core 6 is selected with its duration of 85 cycles. Its power consumption is added to that of slots 1 and 2. Therefore, packet ν is scheduled to slots 1 and 2. The ending time of slot 2 is modified to cycle 85, and a third slot from cycle 86 to 160 is created.

In the proposed approach, we use t to represent one time slot rather than a single clock cycle. Thus, t corresponds to a range $[t_{\text{i}}, \ldots, t_j]$ of test cycles, and L_t contains all packets that can be delivered at time $t' \geq t_{\text{i}}$.

Fig. 8 depicts the proposed test-scheduling heuristic. The algorithm starts by setting the number of packets for each test session and creating an ordered list of test sessions (Lines 1 and 2). A list of I/O pairs is then defined (Line 3). The I/O pairs will be used to create the routing paths for the nonpreemptive tests. Similarly, for each core, a list of possible access paths is created (Line 5), sorted by the number of routers on each path. The list is used to find a path for packets in preemptive test sessions. Lines 6–8 explore the solutions space, which will be explained later.

The scheduling procedure starts at Line 9, the list L_t of packets that can be delivered is created (Line 10), and a packet p (associated to core i) in list L_t is selected (Line 11). A packet in a test session with precedence constraint can only be included in L_t if all precedent test sessions have been completed (Line 10). Since the duration of each packet in the NoC is deterministic, a flag can be set when the last packet of a test session is scheduled. In addition, we give BIST sessions higher priority over all other test sessions.

Note that the packet selected in Line 11 has a delivery time $t_{\text{d}} \in t$. There are three possibilities for this packet.

1) If this packet belongs to a nonpreemptive test, the first available I/O pair m that can be used by this packet is selected (Line 13). The I/O pair availability implies that all channels between the input and the core, and between the core and the output are free to be used. If there is no available I/O pair for packet p, a packet of another core in L_t is selected, and the delivery time t_{d} of packet p is set to the time when the first I/O pair in the list becomes available (Line 15). Otherwise, if I/O pairs and routing paths are available, the duration T_{packet} of packet p in this nonpreemptive session is determined based on (4) (Line 20). Notice that this "super" packet now includes all the flits of all test vectors in this test session. Next, power consumption of this packet is calculated using (3) (Line 21). If the addition of this packet in the current time slot does not exceed the system power limit, the

Procedure *NoC_schedule*

```
1 UBP = Create ordered list of unscheduled BIST test packets.
2 UEP = Create ordered list of unscheduled external test packets.
3 IOP = Create list of I/O pairs.
4 For each core i in C
5    Create ordered list of all possible access paths;
6 For N1 permutation of UBP list
7    For N2 permutations of UEP list
8       For every permutation of IOP list
9          While there are unscheduled packets in UBP⋃UEP
10            L_t = selected packets ready for schedule in UBP⋃UEP according to precedence constraints and
delivery times.
11            Select test packet p of core i in L_t.
12            If p is non-preemptive
13               Find a free I/O pair.
14               If no free I/O pair
15                  Update packet delivery time, repeat from 11.
16            else If p is preemptive or p is BIST
17               Find a free access path to/from core i.
18               If no free access path
19                  Update packet delivery time, repeat from 11.
20            Calculate duration of packet transmission.
21            Calculate power consumption for packet transmission.
22            If power constraint is met
23               Assign packet to the chosen path.
24               Update schedule and time tags.
25               If p is non-preemptive
26                  Assign response packet to chosen I/O pair.
27                  Update schedule and time tags.
28               else If p is preemptive or p is BIST
29                  Define delivery time of next packet of core i;
30               Update L_t;
31            else
32               Update packet delivery time, repeat from 11.
33            If all packets have been attempted
34               Update current time, repeat from 9;
```

Fig. 8. Pseudo code of heuristic scheduling algorithm with both preemptive and nonpreemptive tests.

packet is scheduled (Line 23). In this case, all network channels in the routing path determined by I/O pair m are set to be unavailable during the transmission time of this packet, i.e., between cycles $t_{\mathrm{d}} + T_{\mathrm{headers}} +$ payload and $t_{\mathrm{d}} + T_{\mathrm{packet}}$ (Line 24). The corresponding response packet is then automatically scheduled and removed from the list of unscheduled packets (Lines 26 and 27).

2) If packet p belongs to a preemptive test, the shortest available path k that can be used by this packet is selected (Line 17), and the transmission duration T_{packet} of this packet is determined according to (4) (Line 20). If there is no available path for packet p, a packet of another core in L_t is selected, and the delivery time t_{d} of packet p is set to be the time when the first path in the list of possible paths for the core becomes available (Line 19). Otherwise, if a path is available, the power consumption of this packet is calculated in Line 21. If the power limit is satisfied, the packet is scheduled (Line 23), and all network channels on path k are set to be unavailable during the transmission time of this packet, i.e., between cycles $t_{\mathrm{d}} + T_{\mathrm{headers}} +$ payload and $t_{\mathrm{d}} + T_{\mathrm{packet}}$ (Line 24). If packet p carries a test vector, the corresponding test response packet is set to be ready at time $t_{\mathrm{d}} + T_{\mathrm{packet}} - 1$ (Line 29), as the wrapper takes one cycle to unpack the header of the packet. If packet p carries a response vector, on the other hand, the latency l of the shortest path from input to the core is calculated as $T_{\mathrm{headers}} + T_{\mathrm{router}} * Nb_{\mathrm{routers}}$. The packet carrying the next test vector of core i is set to be ready at time $t_{\mathrm{d}} - l$ (Line 29), ensuring that the new vector will not arrive before the previous vector is processed. In both cases, the next packet of core i is inserted in L_t (Line 30).

3) Finally, if packet p refers to an autonomous BIST test session, a single flit containing the BIST enable signal and other required information for BIST [e.g., reconfiguration values for programmable linear feedback shift registers (LFSRs)] must be sent to the core. That is, we assume that the payload of this packet contains only one flit, and the test application time of each BIST session is known. The real number of flits may vary in practice. Under these assumptions, two cases are possible for BIST-engine utilization. First, if each BISTed core has its own BIST engine, the transmission of packet p is similar to that of a preemptive test. The first available input path is selected to deliver packet p (Line 17), and the chosen path is occupied only for the transmission of a single flit. If there is no available path for packet p, a packet of another core in L_t is selected, and the delivery t_{d} of packet p is set to be the time when the first path in the list of possible paths for core i becomes available (Line 19). Similar to the previous cases, packet p has a delivery time t_{d}. The corresponding response packet (compacted signature) is set to be ready at time $t_{\mathrm{d}} +$ test_time (Line 29), where test_time is the number of cycles required by the BIST session. Second, if several BIST engines are shared among all BISTed cores, the BIST sessions are scheduled as nonpreemptive tests with precedence constraints.

The power consumption per cycle is taken into account by assigning power information to each time slot. During

the scheduling of each slot, the total power consumption is calculated based on (5), where n is the number of packets being transmitted, and c is the number of cores being tested during this time slot

$$\text{Total}_{\text{power}} = \sum_{1 \le j \le n} P_{\text{packet}}(j) + \sum_{1 \le i \le c} P_{\text{core}}(i). \quad (5)$$

The system power limit must be evaluated at each time slot s, i.e., $\text{total}_{\text{power}}(s) \le P_{\max}$, where $P_{\max}$ is the maximum power consumption allowed for the system. Therefore, before the scheduling of any packet, the total power required for the transmission of this packet is calculated (Line 22 in Fig. 8). If the total power consumed in the slot does not exceed $P_{\max}$, the packet can be scheduled (Line 23). Otherwise, the packet is postponed to be scheduled later (Line 32).

Finally, when no packets could be scheduled in the current time slot, the current time is updated by the time slot corresponding to the shortest delivery time of all unscheduled packets (Line 34), and the process is repeated until all packets are scheduled.

D. Exploration of Solution Space

The initial order of the lists UBP and UEP of unscheduled packets (see Fig. 8) determines the priority of packets to be scheduled. Unscheduled packet on the top of the list is first attempted. As a result, different order of the packets leads to different usage of the network channels, hence, different test application times. The order of I/O pairs being attempted for nonpreemptive tests impacts the scheduling in the same manner.

Therefore, a better exploration of the solution space of test scheduling can be achieved by varying the orders of packets and I/O pairs in the lists. In the proposed algorithm, we perform a permutation on a small number of packets in Lines 6, 7, and all the I/O pairs in Line 8. This is because the number of packets are several orders larger than the number of I/O pairs. A complete permutation on all packets is apparently impractical. As shown in Section V-C, a complete permutation on I/O pairs is possible.

The complexity of the proposed heuristic is $O(M!N_{\text{p}})$, where N_{p} is the number of packets, and M is the number of I/O pairs. The permutation on packets is not counted because it is independent of the number of packets. Compared to the dedicated-path approach, the routing of individual packet has increased the complexity of the scheduling.

VII. Experimental Results

A. Experimental Setup

In this section, we present simulation results on some ITC'02 SoC Test Benchmarks [22] using the heuristic algorithms presented in Sections V and VI. The simulation is performed on a PC with a 1.8-GHz processor and 512M RAM. In [29], several network configurations (distinct associations between cores and routers, different NoC topologies, etc.) were considered. For the sake of comparison, we defined an arbitrary but single network configuration (network topology, core assignment to grid, inputs/outputs, etc.) for each benchmark to be used in this paper. Then, the original scheduling algorithm proposed in [29], as well as the algorithms proposed in this paper was performed over this single system configuration. For the heuristic algorithms, we perform the exhaustive I/O order permutations and a number of permutations, from 50 to 200, on the order of cores. For all the simulations, the algorithms can be concluded in less than 20 min. We note that the original scheduling does not support permutation of the cores, and allowing permutation on more cores in the improved heuristics can lead to better solutions.

The power consumption of the cores during test is not included in the ITC'02 benchmarks. Therefore, hypothetic data are used in simulation. The power profile of each core is defined as a function of the number of scan flip–flops, inputs, and outputs and is evaluated using (1), with nb_{ff} and nb_{gt} replaced by the estimated number of flip–flops and gates of the module, respectively. Moreover, the core and wrapper power consumption is considered as the peak value among all vectors. BIST and nonpreemptive test sessions imply higher power consumption of the core under test.

The power-consumption limit for the system is defined as a percentage of the total power consumption of all cores under test. For example, a power limit of 50% indicates that the power limit equals to half of the sum of the power consumption of all cores in test mode. Note that in practice, the power constraint should be set by the designer. Moreover, in this paper, we do not assume any specific type of test-set manipulation for reducing the power consumption, which can be easily performed by modifying the order of the test sessions and including additional precedence constraints. We also assume that the power consumption of the network routers and channels is one order of magnitude smaller than that of the cores.

B. Results for ITC'02 SoC Test Benchmarks

For each benchmark, we present results for different numbers of interfaces with the tester, different configurations of test sessions, with BIST and precedence rules, and for different power constraints. For the sake of succinctness, we refer to the packet-based preemptive test scheduling in [29] as the base-case algorithm. We refer to the dedicated-path routing for nonpreemptive test scheduling and the improved heuristic for both preemptive and nonpreemptive tests scheduling as Algorithms 1 and 2, respectively.

Tables I–III present the results for the original benchmarks, with and without power constraints, for different numbers of test interfaces (I/O ports). Column 1 gives the number of I/O ports. Columns 2 and 3 give the results of the base-case algorithm reported in [29], where only preemptive tests are allowed. In Column 2, no power limit is assumed for the test, whereas in Column 3, a power limit of 50% is considered. Columns 4 and 5 show the results of Algorithm 1 using the dedicated-path approach presented in Section V, where only nonpreemptive tests are allowed. Columns 6 and 7 show the test application time when both nonpreemptive and preemptive tests exist, using Algorithm 2, i.e., the improved heuristic proposed in Section VI-C. Preemptive and nonpreemptive test sessions

TABLE I
TEST-SCHEDULING RESULT FOR d695 WITH POWER CONSTRAINTS

	Base case		Algorithm 1		Algorithm 2	
# of I/Os	No power limit	50% Power limit	No power limit	50% Power limit	No power limit	50% Power limit
2/2	26012	27087	19909 (-23.5%)	19909 (-26.5%)	19128 (-26.5%)	19936 (-26.4%)
3/3	20753	20733	14373 (-30.7%)	16092 (-22.4%)	16286 (-21.5%)	16422 (-20.8%)
4/4	14785	17623	10472 (-29.2%)	11963 (-32.1%)	10582 (-28.4%)	12917 (-26.7%)

TABLE II
TEST-SCHEDULING RESULT FOR p22810 WITH POWER CONSTRAINTS

	Base case		Algorithm 1		Algorithm 2	
# of I/Os	No power limit	50% Power limit	No power limit	50% Power limit	No power limit	50% Power limit
2/2	315708	315708	285911 (-9.4%)	285911 (-9.4%)	277494 (-12.1%)	277494 (-12.1%)
3/3	222432	224411	199361 (-10.4%)	199361 (-11.2%)	207852 (-6.6%)	207852 (-6.6%)
4/4	170999	177330	140047 (-18.1%)	140047 (-21.0%)	148201 (-13.3%)	154322 (-13%)

TABLE III
TEST-SCHEDULING RESULT FOR p93791 WITH POWER CONSTRAINTS

	Base case		Algorithm 1		Algorithm 2	
# of I/Os	No power limit	50% Power limit	No power limit	50% Power limit	No power limit	50% Power limit
2/2	639443	639443	617115 (-3.5%)	617115 (-3.5%)	655123 (+2.4%)	655123 (+2.4%)
3/3	475311	470446	420459 (-11.5%)	420459 (-10.6%)	458397 (-3.6%)	458397 (-2.6%)
4/4	372615	366892	343312 (-7.8%)	343312 (-6.4%)	351102 (-5.7%)	343389 (-6.4%)

are chosen arbitrarily. In addition, Columns 4–7 present the corresponding test-time reduction in percentage compared to Columns 2 to 3, respectively.

It can be observed, from Columns 2 to 5, that Algorithm 1 is more efficient than the base-case algorithm using packet-based routing. A test-time reduction from 3% to 30% is achieved even under power constraints. This can be explained from the aspects of both resource conflict and solution space, as discussed in Section V-A. However, we note that if more computational power is available or more simulation time is allowed, the base case can search a larger portion of the solution space and may possibly yield shorter test time. This is because the base-case algorithm provides a higher level of flexibility on packet scheduling than Algorithm 1, which cannot handle individual packet. We also note that there is a permutation on I/O pairs in Algorithms 1 and 2, which is not available in the base case. This permutation also makes a significant contribution to the exploration of solutions space, since we have found in experiments that varying the assignments of packets to I/O pairs can cause dramatic change on the test time.

TABLE IV
TEST-SCHEDULING RESULT FOR d695 WITH VARIOUS CONSTRAINTS USING ALGORITHM 1

# of I/Os	Algorithm 1	With BIST	BIST and precedence	BIST, precedence, and 50% power limit
2/2	19909	20567	19571	19571
3/3	14373	15179	14589	14589
4/4	10410	10934	10257	10123

TABLE V
TEST-SCHEDULING RESULT FOR p22810 WITH VARIOUS CONSTRAINTS USING ALGORITHM 1

# of I/Os	Algorithm 1	With BIST	BIST and precedence	BIST, precedence, and 50% power limit
2/2	290080	2850106	1698115	1698115
3/3	199361	1542591	1595492	1595492
4/4	140047	1498765	1563397	1563397

TABLE VI
TEST-SCHEDULING RESULT FOR p93791 WITH VARIOUS CONSTRAINTS USING ALGORITHM 1

# of I/Os	Algorithm 1	With BIST	BIST and precedence	BIST, precedence, and 50% power limit
2/2	486156	7009102	5873046	5873046
3/3	486152	3937061	4070465	4070465
4/4	344343	3035193	3024979	3024979

For systems with both preemptive and nonpreemptive tests, we expect that Algorithm 2 using the proposed improved heuristic can lead to test application time between those given by the base case and Algorithm 1. This is intuitive because the improved heuristic is a combination of the packet-based routing and dedicated-path routing. The expectation is corroborated by the results shown in Columns 6 and 7. We observe that in all the cases, Algorithm 2 yields a shorter test application time than the base case. In most cases, the time is slightly longer than that given by Algorithm 1. However, in some cases, it yields even further reduction on test application time than Algorithm 1, indicating that a better solution could be potentially reached due to the higher flexibility offered by Algorithm 2.

It can also be observed that adding a power constraint can generally cause the increase on test application time. If the constraint is loose, however, test time may not be affected, as shown in some of the cases in the tables.

Tables IV–VI present the results when BISTed sessions, precedence rules, and power constraints are considered in

TABLE VII
TEST-SCHEDULING RESULT FOR d695 WITH VARIOUS CONSTRAINTS USING ALGORITHM 2

# of I/Os	Algorithm 2	With BIST	BIST and precedence	BIST, precedence, and 50% power limit
2/2	19128	20049	19928	20001
3/3	16286	15034	14889	14802
4/4	10582	10574	10621	11082

TABLE VIII
TEST-SCHEDULING RESULT FOR p22810 WITH VARIOUS CONSTRAINTS USING ALGORITHM 2

# of I/Os	Algorithm 2	With BIST	BIST and precedence	BIST, precedence, and 50% power limit
2/2	277494	1584729	1731596	1731596
3/3	207852	1492644	1604805	1604805
4/4	148201	1472733	1555656	1555656

TABLE IX
TEST-SCHEDULING RESULT FOR p93791 WITH VARIOUS CONSTRAINTS USING ALGORITHM 2

# of I/Os	Algorithm 2	With BIST	BIST and precedence	BIST, precedence, and 50% power limit
2/2	655123	5821302	5886746	5886746
3/3	458397	3973691	4103232	4103232
4/4	351102	3040505	3078347	3078347

Algorithm 1. Tables VII–IX present the results when BISTed sessions, precedence rules, and power constraints are considered in Algorithm 2. For d695, we assume that less than 1 k random test vectors are generated for BIST sessions. While for other larger benchmarks, we assume that each BIST session consists of 128 k random vectors. Column 1 gives the number of I/O ports. Column 2 gives the test application time using Algorithm 2 for both preemptive and nonpreemptive tests without any constraint, as listed in Column 6 of Tables I–III. Column 3 shows the results when some BISTed sessions with shared BIST engines are arbitrarily added to the test. The results in Column 4 show the impact of adding precedence rules among test sessions, and Column 5 presents the results when a 50% power limit is added on top of BIST sessions and precedence constraints.

When these constraints are considered, it can be seen that test application time is in most cases increased. For p22810 and p93791, the increase on test time is primarily caused by the long BIST sessions, which are dominant compared to the external test sessions. The inclusion of precedence and power constraints usually causes an increase on the test time. However, in some cases, they have indeed led to a test-time reduction. This is because imposing the precedence and power constraints can cause changes on schedule, e.g., different selection of cores, interfaces or test patterns, etc., which could potentially yield a better solution. Also, note that in many cases, the inclusion of precedence constraints has led to less power dissipation, and the power constraints are automatically met, hence, adding power constraints has not impact on the results.

Finally, we examine the performance of the two algorithms under a more accurate power model to further explore and explain the flexibility provided by Algorithm 2. Therefore, far in the experiments, we assume that a core consumes the same amount of power under either preemptive or nonpreemptive testing. However, a more realistic model should consider that the nonpreemptive testing actually consumes more power than the preemptive testing, since the latter spreads over a longer period of time than the former while the total energy of testing a core is fixed.

We take this into account in the experiments presented in Table X. As shown in the previous tables, the "base case" is still the preemptive algorithm where all cores in the system are preemptive, hence, we omit those results in this table. We also assume the same system power limit. Since Algorithm 1 uses nonpreemptive testing for all cores, the corresponding power consumption will increase compared to the base case. For Algorithm 2, only the cores with nonpreemptive tests have an increase in the power consumption. In Table X, we show the scheduling results with only power constraints (Columns 3, 5), and with power, BIST, and precedence constraints (Columns 4, 6) for the two algorithms, respectively.

For d695, we assume an increase of 20% on the power consumption of nonpreemptive cores. It can be observed that Algorithm 2 generates better results in two cases out of three, with only power constraint. This is because out of the ten cores in this system, five are using preemptive tests that yield less power consumption compared to the Algorithm 1 where all ten cores use nonpreemptive testing. When BIST and precedence constraints are added, however, Algorithm 2 is outperformed in two cases. This is due to the fact that BIST tests are nonpreemptive, which make the nonpreemptive tests dominant in the system. Thus, the advantage of less power consumption from preemptive cores using Algorithm 2 is less noticeable.

For systems p22810 and p93791, we assume an increase of 40% on the power consumption of the nonpreemptive cores over the preemptive cores, to account for the higher level of preemption of the larger cores, compared to d695. Similar results can be observed: Algorithm 2 leads to shorter test time when only power constraint is considered, but Algorithm 1 does better when BIST tests are added. Note that both systems contain "parent" cores, which consist of lower level "child" cores, and all these cores are assumed to be nonpreemptive. Therefore, in these two systems, the nonpreemptive cores dominate.

Note that the above results indicate that Algorithm 2 is more flexible than Algorithm 1. It allows the designer to determine the test of each core, either preemptive or nonpreemptive, based on its power consumption and other constraints, to yield a more optimized test. This flexibility can lead to a less test time in many cases, as shown in the experimental results.

VIII. CONCLUSION

This paper has proposed a comprehensive test-scheduling approach for embedded core-based systems in a NoC architecture.

TABLE X
RESULTS WITH CONSTRAINTS USING NEW POWER MODEL

		Algorithm 1		Algorithm 2	
		With power only	With BIST&precedence	With power only	With BIST&precedence
d695	2/2	20526	19571	19395	20050
	3/3	16071	15179	14273	14526
	4/4	13541	10754	13694	11346
p22810	2/2	280017	1698115	277494	1731596
	3/3	206975	1595492	198237	1604805
	4/4	165945	1564801	145794	1555656
p93791	2/2	626059	6130239	649327	6164302
	3/3	423344	4257413	440173	4304847
	4/4	345599	3024979	336125	3078347

A dedicated-path approach was first proposed to target the cores that do not allow preemptive testing. The problem of optimal test scheduling using the dedicated-path approach was proved as NP-complete, and a heuristic was presented. The scheduling approach was then improved by incorporating both preemptive and nonpreemptive tests to handle the real-world problems. Finally, BIST test session, precedence, and power constraints were included in scheduling to deliver a comprehensive and practical test solution for NoC-based system. Experimental results on ITC'02 benchmarks have shown the effectiveness of the proposed solution.

REFERENCES

[1] J. D. Ullman, "Complexity of sequencing problems," in *Computer and Job/shop Scheduling Theory*. New York: Wiley, 1976, pp. 139–164.

[2] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.

[3] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*. Los Alamitos, CA: IEEE Computer Soc., 1997.

[4] S. H. Gerez, *Algorithms for VLSI Design Automation*. Chichester, U.K.: Wiley, 1998.

[5] P. Harrod, "Testing reusable IP-A case study," in *Proc. Int. Test Conf.*, 1999, pp. 493–498.

[6] K. Chakrabarty, "Test scheduling for core-based systems using mixed-integer linear programming," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 10, pp. 1163–1174, Oct. 2000.

[7] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Proc. Design, Autom. Test Eur.*, 2000, pp. 250–256.

[8] W. J. Dally and B. Towles, "Route packets, not wires: On-Chip interconnection networks," in *Proc. Des. Autom. Conf.*, 2001, pp. 684–689.

[9] M. Nahvi and A. Ivanov, "A packet switching communication-based test access mechanism for system chips," in *Proc. IEEE Eur. Test Workshop*, 2001, pp. 81–86.

[10] C. Aktouf, "A complete strategy for testing an on-chip multiprocessor architecture," *IEEE Des. Test Comput.*, vol. 19, no. 1, pp. 18–28, Jan. 2002.

[11] S. Basu, I. Sengupta, D. R. Chowdhury, and S. Bhawmik, "An integrated approach to testing embedded cores and interconnects using test access mechanism (TAM) switch," *J. Electron. Testing: Theory Appl.*, vol. 18, no. 4/5, pp. 475–485, Aug.–Oct. 2002.

[12] L. Benini and G. D. Micheli, "Networks on chips: A new SoC paradigm," *IEEE Comput.*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[13] E. Cota, L. Carro, A. Orailoglu, and M. Lubaszewski, "Test planning and design space exploration in a core-based environment," in *Proc. Design, Autom. Test Eur.*, 2002, pp. 478–485.

[14] M. Benabdenbi, W. Maroufi, and M. Marzouki, "CAS-BUS: A test access mechanism and a toolbox environment for core-based system chip testing," *J. Electron. Testing: Theory Appl.*, vol. 18, no. 4/5, pp. 455–473, Aug. 2002.

[15] S. Goel and E. Marinissen, "Cluster-based test architecture design for system-on-chip," in *Proc. IEEE VLSI Test Symp.*, 2002, pp. 259–264.

[16] ——, "Effective and efficient test architecture design for SoCs," in *Proc. Int. Test Conf.*, 2002, pp. 529–538.

[17] Y. Huang *et al.*, "Optimal core wrapper width selection and SoC test scheduling based on 3-D bin packing algorithm," in *Proc. Int. Test Conf.*, 2002, pp. 74–82.

[18] V. Iyengar and K. Chakrabarty, "Test wrapper and test access mechanism co-optimization for system-on-chip," *J. Electron. Testing: Theory Appl.*, vol. 18, no. 2, pp. 213–230, Apr. 2002.

[19] ——, "System-on-a-chip test scheduling with precedence relationships, preemption, and power constraints," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 21, no. 9, pp. 1088–1094, Sep. 2002.

[20] S. Koranne, "Formulating SoC test scheduling as a network transportation problem," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 12, pp. 1517–1525, Dec. 2002.

[21] S. Koranne and V. Iyengar, "On the use of k-tuples for SoC test schedule representation," in *Proc. Int. Test Conf.*, 2002, pp. 539–548.

[22] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, "A set of benchmarks for modular testing of SoCs," in *Proc. Int. Test Conf.*, 2002, pp. 521–528.

[23] C. Zeferino, M. Kreutz, L. Carro, and A. Susin, "A study on communication issues for systems-on-chip," in *Proc. Symp. Integr. Circuits Syst. Des.*, 2002, pp. 121–126.

[24] E. Cota, C. Zeferino, M. Kreutz, L. Carro, M. Lubaszewski, and A. Susin, "The impact of NoC reuse on the testing of core-based systems," in *Proc. IEEE VLSI Test Symp.*, 2003, pp. 128–133.

[25] W. Zou, S. Reddy, I. Pomeranz, and Y. Huang, "SoC test scheduling using simulated annealing," in *Proc. IEEE VLSI Test Symp.*, 2003, pp. 325–330.

[26] B. Vermeulen, J. Dielissen, K. Goossens, and C. Ciordas, "Bringing communication networks on a chip: Test and verification implications," *IEEE Commun. Mag.*, vol. 41, no. 9, pp. 74–81, Sep. 2003.

[27] C. Zeferino and A. Susin, "SoCIN: A parametric and scalable network-on-chip," in *Proc. Symp. Integr. Circuits and Syst. Des.*, 2003, pp. 121–126.

[28] A. Bona, V. Zaccaria, and R. Zafalon, "System level power modeling and simulation of high-end industrial network-on-chip," in *Proc. Des., Autom. Test Eur.*, 2004, pp. 318–323.

[29] E. Cota, L. Carro, and M. Lubaszewski, "Reusing an on-chip network for the test of core-based systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 9, no. 4, pp. 471–499, Oct. 2004.

[30] M. Nahvi and A. Ivanov, "Indirect test architecture for SoC testing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 7, pp. 1128–1142, Jul. 2004.

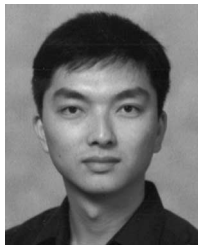
[31] C. Su and C. Wu, "A graph-based approach to power-constrained SoC test scheduling," *J. Electron. Testing: Theory Appl.*, vol. 20, no. 1, pp. 45–60, Feb. 2004.

[32] C. Liu, K. Veeraraghavan, and V. Iyengar, "Thermal-aware test scheduling and hot spot temperature minimization for core-based systems," in *Proc. Int. Symp. Defect Fault Tolerance VLSI Syst.*, 2005, pp. 552–560.

[33] E. Tafaj, P. Rosinger, and B. Al-Hashimi, "Improving thermal-safe test scheduling for core-based system-on-chip using shift frequency scaling," in *Proc. Int. Symp. Defect Fault Tolerance VLSI Syst.*, 2005, pp. 544–551.

Érika Cota (S'01–A'03–M'04) received the B.S. degree in computer science from the Universidade Federal de Minas Gerais, Belo Horizonte, Brazil, in 1994, and the M.S. and Ph.D. degrees in computer science from Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil, in 1997 and 2003, respectively.

She is currently an Adjunct Professor with the Instituto de Informatica of Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, where she participates in the Group of Test and Reliability of Integrated Systems and in the Embedded Systems Laboratory. She has been working in the test domain since 1994 and has published on the topics of analog and mixed-signal testing, BIST, system-on-chip testing planning, and network-on-chip (NoC)-based testing. Her research interests include the test and design for test of hardware and software systems, test of embedded systems, test planning, and fault tolerance of integrated systems.



Chunsheng Liu (S'00–M'03) received the B.S. and M.S. degrees in electronic engineering from Tsinghua University, Beijing, China, in 1997 and 2000, respectively, and the Ph.D. degree in electrical and computer engineering from Duke University, Durham, NC, in 2003.

He is currently an Assistant Professor of computer and electronics engineering with the University of Nebraska-Lincoln, Lincoln, NE. His research interests include very-large-scale-integration design, testing, and fault diagnosis. He is currently working in the areas of NoC design and testing.

Dr. Liu is a member of Association for Computing Machinery (ACM) and the ACM Special Interest Group on Design Automation.