Determining the Test Sources/Sinks for NoC TAMs

Alexandre Amory, Edson Moreno, Fernando Moraes
FACIN - PUCRS University
Porto Alegre, Brazil
Email: {alexandre.amory,edson.moreno,fernando.moraes}@pucrs.br

Marcelo S. Lubaszewski
CEITEC S.A.
Porto Alegre, Brazil
Email: marcelo.lubaszewski@ceitec.org.br

***Abstract*—Conventional approaches using the Network-on-Chip (NoC) as Test Access Mechanism (TAM), called NoC TAM, model the test sources/sinks and the routing algorithm as constraints to the test scheduling, reducing its efficiency. This paper is based on a new NoC TAM model where these constraints do not exist, potentially resulting in shorter tests. The contribution of this paper is to present the part of the test flow which determines the optimal number and location of the test sources and sinks in a NoC TAM without constraining the test scheduler. Searching the minimal number of sources/sinks can minimize the silicon area overhead since each NoC source/sink requires about 4300 gates for a NoC channel with 32-bit width.**

Keywords*—*VLSI test, SoC test, networks-on-chip.

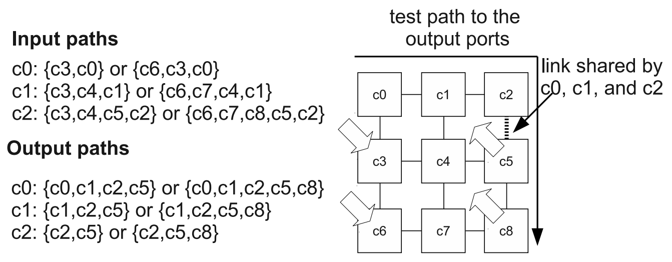

Input paths
c0: {c3,c0} or {c6,c3,c0}
c1: {c3,c4,c1} or {c6,c7,c4,c1}
c2: {c3,c4,c5,c2} or {c6,c7,c8,c5,c2}
Output paths
c0: {c0,c1,c2,c5} or {c0,c1,c2,c5,c8}
c1: {c1,c2,c5} or {c1,c2,c5,c8}
c2: {c2,c5} or {c2,c5,c8}
test path to the output ports
link shared by c0, c1, and c2
c0
c1
c2
c3
c4
c5
c6
c7
c8

Fig. 1. Example of a 3x3 NoC with mesh topology used to demonstrate how the I/O port and the routing algorithm can constrain the test scheduling. The incoming and outgoing arrows represent test sources and sinks, respectively. It also lists the set of possible input/output paths for each c1, c2, and c3.

I. Introduction

The SoCs were emerging back in 2000, enabling the integration of tens of cores into a single chip and requiring innovative SoC testing approaches. In this context, *Modular testing* has been proposed as a divide-and-conquer solution to test such complex SoCs [1]. Its conceptual model consists of test wrappers, test sources and sinks, and Test Access Mechanisms (TAMs). The most common practice for TAM design is to use *dedicated and global test buses* for test data transportation. Nowadays, the SoC integration level continues to increase towards hundreds of cores [2], [3] interconnected by Networks-on-Chip (NoCs), replacing buses which are not scalable.

On the other hand, the conventional test architecture still uses non-scalable buses as TAMs, which motivated the use of the existing NoC to transport test data [4], providing the scalability of NoCs to the test architecture. This approach is referred as *NoC TAM*. However, Yaun et al. [5] have formally demonstrated that the existing NoC TAM approaches can have longer test time compared to bus-based TAMs. This is caused the excessive number of constrains on the NoC TAM problem formulation, reducing its efficiency.

This paper, on the other hand, is built on top of a new NoC-based test method which is more efficient than the methods analyzed by [5]. This test method divides the NoC TAM definition problem in two major steps: (*i*) the test scheduler based on graph partitioning and (*ii*) determining test source/sink for the test partitions, which is the *main contribution of this paper*.

This paper is organized as follows. Section II presents previous NoC TAM approaches and shows how these approaches severely constrain the NoC TAM test scheduler. Section III summarizes the new NoC-based test optimization method where the algorithm presented in this paper is inserted. Sections IV and V present and evaluate, respectively, the proposed algorithm to determine the test source/sink. Section VI concludes the paper.

II. Prior Work

This section presents prior work about NoC TAMs test scheduling, their similarities and open issues. The problem formulation proposed by Cota et al. [6] is the core problem statement for most, if not all, papers related to test scheduling for NoC TAMs. This initial problem formulation has been extended several times, including new constraints and variables to the problem (power dissipation [7], thermal [8], among others), or assuming the existence of certain NoC functionality (variable clock rate [8], virtual channels [9], circuit switching, among others). Despite these extensions, the core problem statement is the same and can be stated as:

- *given* a mesh-based NoC with XY routing algorithm; a set of cores, their test information (internal scan chains, number of I/O terminals, and so on), and the core's location in the NoC; a set of I/O ports, the IO location in the NoC, and its bandwidth capacity; *determine* an assignment of cores to input/output pairs *such that* the total test time is minimized and the I/O capacity is not exceeded.

In summary, the NoC TAM test scheduling problem can be summarized as how to efficiently assign input/output pairs to cores, such that the overall test time is minimized. These NoC TAM approaches require I/O port information (their bandwidth and location) and the NoC routing algorithm as inputs of the test scheduling optimization problem. Then, the algorithm must assign one of these fixed I/O ports to test the Core Under Test (CUT) according to the routing algorithm.

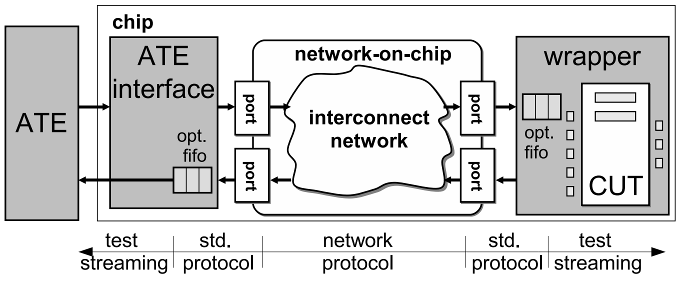


Fig. 2. Proposed conceptual test architecture, its DfT modules (ATE interfaces and test wrappers), and the protocol conversions. The gray blocks are related to test.

This approach constrains the test scheduler, increasing the SoC test time, as illustrated in Fig. 1. In this example, the test of cores c0, c1, and c2 cannot be executed in parallel due to the location of the output ports and the routing algorithm (XY, where the packet follows first the X axis and, if required, the Y axis). These three cores can only send their test stimuli to the output ports located in c5 or c8. This way, according to the XY routing algorithm, the three test paths share the link between core 2 and 5 to reach an output port. The paper [5] formally demonstrates that this specific constraint causes longer test time than the conventional tests based on bus-based TAMs, reducing the research interest in NoC TAMs.

Amory et al. [10] proposed a new NoC TAM test scheduling algorithm, summarized in Section III, which does not suffer from the issues mentioned above.

III. New NoC TAM Test Method

This section describes the background used by the proposed method: the conceptual NoC TAM test architecture (Section III-A) and the test scheduling (Section III-B).

A. The NoC TAM Test Architecture

Fig. 2 shows the conceptual *test architecture*, its DfT modules, and the protocol conversions along the test data flow. The edges of the test flow (i.e. the Automatic Test Equipment (ATE) and the CUT) require test streaming, the NoC interface requires a standardized protocol such as OCP (Open Core Protocol), and the NoC internals use some network protocol, such as handshake, which is transparent to the test. The DfT modules, called ATE interface and wrapper, do all required protocol and width conversions such that both the ATE and the CUT are not aware of the NoC.

An ATE interface [11] is the DfT module that connects the test pins to the NoC. It does bi-directional data transfer of test stimuli from test pins to the NoC (toward the CUT) and test reponses from the CUT (via NoC) to the test pins. NoCs typically use standardized on-chip protocols, like OCP, which is different from the test protocol used by ATEs. Thus, the main tasks of an ATE interface are to do width conversion between the number of test pins and the network data width, protocol conversion from the ATE to the NoC (and vice-versa), and traffic shaping for the test stream.

This test traffic shaping using NoCs is perhaps the most relevant challenges of NoC TAMs. The test data flow requires deterministic latency and no jitter such that the scan chains do not have to use, for instance, clock gating to stop shifting test data. However, NoCs use routers as shared resources. Shared resources lead to an arbitration logic which leads to variable latency and jitter, according to the traffic conditions on the NoC. One way to guarantee deterministic traffic in NoCs is to reserve the NoC resources using Quality-of-Service approaches like circuit switching. However, not all NoCs have this feature. The method used in this paper spatially distributes the test data flows on the NoC such that no flow uses the same router. This way, there are no shared resources, providing deterministic latency and no jitter, even though the NoC does not have circuit switching. Next section explains how multiple test flows are distributed in the NoC.

B. Problem Statement of the Test Scheduler

The paper [10] presents a test scheduling algorithm which is the basis for the method presented in this paper. Let an undirected graph $G = (V, E)$ represent the NoC topology, where V is the set of NoC nodes and E is the set of NoC links that determines the connectivity between the NoC nodes. A NoC node, or just node, consists of exactly one NoC router and a set of zero or more CUTs which are connected to this router. Let us also define the following symbols which are used along this paper:

- Slanted upper case symbols $\mathcal{P}$ are a **partition** of G;
- Upper case symbols P are a **part** of the partition $\mathcal{P}$;
- Lower case symbols n represent a single **node** of the part P;
- $t(P)$ is the **test length** (in clock cycles) of the part P;
- $b(P)$ is the **test bandwidth** (in Mbps) assigned to the part P;
- $w(P)$ is the **part width** (in number of test pins) assigned to the part P.

The *NoC TAM test scheduling problem* is stated as: *Given* the total number of test pins w_{max} and the maximum sustainable NoC channel bandwidth b_{noc}. *Given* a set of cores and the their test-related information (as defined in [10]). *Given* a graph $G = (V, E)$, representing the NoC topology. The goal of the test scheduling algorithm is to *determine* a partition $\mathcal{P}$ of G *such that*:

1) $(\sum_{i=0}^{|\mathcal{P}|} w(P_i)) \leq w_{max}$, i.e. the sum of all part widths $w(P_i)$, with $P_i \in \mathcal{P}$, do not exceed the number of test pins w_{max};
2) $\forall_{P_i \in \mathcal{P}} (b(P_i) \leq b_{noc})$, i.e. the test bandwidth of each part $b(P_i)$ do not exceed the maximal bandwidth of the NoC channels b_{noc};
3) The SoC test time T is minimized, where $T = max_{i=0}^{|\mathcal{P}|} t(P_i)$;
4) A valid partition has the following properties: all nodes of a part are connected, all nodes are assigned to exactly one part, and all parts are disjoint;

This new problem statement is significantly *more general* and *less constrained* than the problem statement presented in

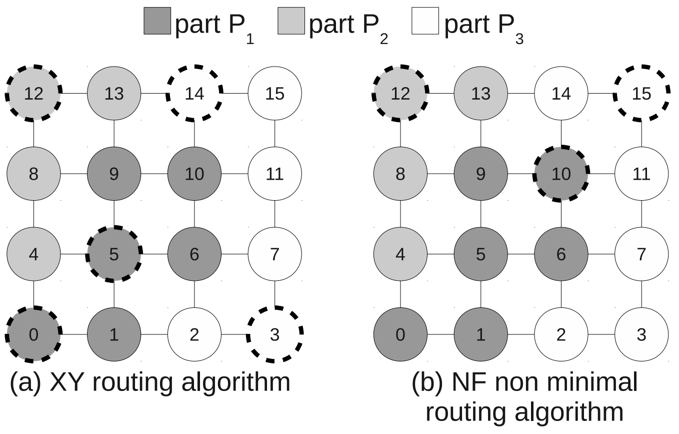


Fig. 3. Example of test partition of a 4x4 mesh and the resulting ATE interfaces location assuming XY (a) and NF non minimal (b) routing algorithms. Nodes with dashed line represent the location of an ATE interface.

Section II. For instance, it supports most NoC topologies, even the irregular ones, and the routing algorithm and test ports do not constrain the test scheduling algorithm. However, the information generated by this algorithm only optimizes the SoC test time and the CUT test wrappers. It does not define the ATE interfaces for the NoC TAMs. Next section presents the proposed method to optimize the ATE interfaces, completing the test flow.

IV. Determining the Location of ATE Interfaces

This section presents the contribution of this paper and how it complements the test method presented in Section III-B.

A. Example

Fig. 3 illustrates an example test partition with 3 parts (P_1, P_2, and P_3) of a 4x4 mesh NoC generated by the test scheduler presented in Section III-B. This example partition is used to explain the problem of determining the location of the test sources and sinks for NoC TAMs. Two classes of routing algorithms are assumed in this example: *deterministic routing algorithms*, e.g. XY, always use the same path for the same source-target nodes; *partially adaptive routing algorithms*, e.g. North First non minimal (NF), it might take different paths for the same source-target nodes, according to the NoC congestion status [12].

The *goal of the proposed method* is to determine where an ATE interface can be placed such that it can send test stimuli and receive test responses for all CUTs within its test partition without using nodes from other parts, which could cause network congestion as explained before. For instance, assuming XY and part P_3 (Fig. 3(a) in white color), P_3 requires two ATE interfaces to reach its nodes. One ATE interface is located at node 3, with access to the nodes $\{2, 3, 7, 11, 15\}$, and the other is located at node 14, with access to $\{14, 15\}$. It means that node 3 can send test stimuli and receive test responses from the nodes $\{2, 3, 7, 11, 15\}$ without using resources from other parts, creating a deterministic test data flow. Note that node 3 cannot test node 14 because it would use nodes 6 and 10, which do not belong to P_3. This way, another ATE interface is required for P_3. Similarly, the part P_1 also requires two ATE interfaces; one at node 0 and the other at node 5. The part P_2 requires a single ATE interface at node 12 because it has access to all nodes of this part.

Assuming the NF routing algorithm [1] (Fig. 3(b)), the ATE interfaces would be located on nodes 10, 12, and 15 for the parts P_1, P_2, and P_3, respectively. Thus, one ATE interface is required to access all nodes of each part.

These examples show that the definition of test sources/sinks for NoC TAMs depends on the *NoC routing algorithm* and the *geometry (shape) of the parts*. Moreover, each pair of test source/sink requires an ATE interface to perform the appropriate data flow adaptation, and this ATE Interface costs silicon area [11]. Thus, the proposed method also *minimizes the number of ATE interfaces*, respecting the constraint of routing algorithm and the test partition geometry. Finally, a part might need more than one ATE interface, like P_3 in Fig. 3(a). In this case, these ATE interfaces share the same test pins and they are activated sequentially, not causing network congestion within the part. In the example of P_3 in Fig. 3(a), it means that the ATE interface in node 3 is used as test source/sink for the nodes $\{2, 3, 7, 11\}$ and, after testing these nodes, the node 14 starts testing the nodes $\{14, 15\}$. Once the ATE interfaces are defined for the whole test partition, the NoC test data flow is completly defined. It means that it is possible to determine, for instance, which ATE interface is used for each CUT. For example, each CUT's test wrapper is configured in design time with its test sink address in the NoC. This way, when node 2 is in test mode, it knows its test stimuli must be sent to the node 3. The test stimuli of node 15 must be sent to the node 14, and so on.

B. Problem Statement

Let us define that for each part $P \in \mathcal{P}$, there is a non-empty subset of P, called A, representing the location of the ATE interfaces of part P. Let this relationship between P and A be expressed in terms of a function $A = \mathit{LocateATEIForPart}(P)$. For example, if we assume the part P_2 illustrated in Fig. 3(a), then $A_2 = \mathit{LocateATEIForPart}(P_2) = \{12\}$ since the node 12 has access to all nodes of the part P_2. Let also $\mathcal{A}$ be the set of A for the entire partition $\mathcal{P}$. For example, assuming the partition illustrated in Fig. 3(a), then $\mathcal{A} = \{A_1, A_2, A_3\} = \{\{0, 5\}, \{12\}, \{3, 14\}\}$ since these are the ATE interfaces for the three parts.

The problem is stated as: **given** the test partition $\mathcal{P}$ of the graph G, as presented in Section III-B, and the NoC routing algorithm, **determine** $\mathcal{A}$ for the partition $\mathcal{P}$ **such that**:

1) The total number of ATE interfaces ($\sum_{A \in \mathcal{A}} |A|$) is minimized;
2) Each part of a partition has at least one ATE interface, i.e. $\forall_{P \in \mathcal{P}} \mathit{LocateATEIForPart}(P) \neq \emptyset$;
3) Each node $n_1 \in P$ is accessible by at least one node $n_2 \in A$ assuming the given routing algorithm such that the path from n_1 to n_2 and from n_2 to n_1 must only consist of nodes in P.

[1]NF routing algorithm states that, if it needs to go to the north, then the north direction must be the first turn and the north direction cannot be taken again.

The *goal* of the proposed algorithm is to determine the minimal number of ATE interfaces and their location considering a given NoC routing algorithm and a given network partition. The selected ATE interface locations must be able to send test stimuli and receive test responses from all nodes of a part without using resources (nodes and links) from other parts, avoiding any type of disturbance on the test of other parts, creating test data flows with deterministic latency and no jitter.

C. The Algorithm

Let us define the function *getPath*(n_1, n_2, P) that implements the routing algorithm. It returns the set of nodes in the path between the nodes n_1 and n_2 (test stimuli path) and also from n_2 to n_1 (test responses path) such that all nodes in both paths belong to the part P. It returns an empty set if there is no such path between n_1 and n_2 using only the nodes in P.

The Algorithm 1 determines the ATE interface location for the entire partition. It concatenates the results from Algorithm 2 for each part of the partition into $\mathcal{A}$. Let us assume the example in Fig. 3(a). The first iteration generates the ATE interface location for part P_1, resulting in $\mathcal{A} = \{\{0, 5\}\}$. The second iteration includes the ATE interface location for part P_2, resulting in $\mathcal{A} = \{\{0, 5\}, \{12\}\}$. The last iteration generates the final result $\mathcal{A} = \{\{0, 5\}, \{12\}, \{3, 14\}\}$, including the ATE interface location for part P_3.

Algorithm 1 [LocateATEIForPartition($\mathcal{P}$)]

```
01. $\mathcal{A} = \emptyset$
02. for all $P \in \mathcal{P}$
03.   $\mathcal{A} = \mathcal{A} \cup \{$ LocateATEIForPart$(P)\}$
04. return $\mathcal{A}$
```

The Algorithm 2 determines the ATE interface location for a single part of a partition. It starts by building the reachability set ($\mathcal{P}_{conn}$, lines 3 to 9) for the part P. The reachability set can be explained by an example. Let us assume the part P_3 in Fig. 3(a). Node 2, for instance, can reach nodes $\{2, 3\}$, which means that node 2 can send test stimuli to and receive test responses from the nodes $\{2, 3\}$. The rest of the reachable nodes are presented below:

- *node 2*: $\{2, 3\}$;
- *node 3*: $\{2, 3, 7, 11, 15\}$;
- *nodes 7 and 11*: $\{3, 7, 11, 15\}$;
- *node 14*: $\{14, 15\}$;
- *node 15*: $\{3, 7, 11, 14, 15\}$.

Finally, the reachability set $\mathcal{P}_{conn}$ is formalized as a set of tuples (n, R) where the item n represents the source node and the item R is a set of all reachable nodes from the node n. For instance, according to the example above, the tuple for node 2 is $(2, \{2, 3\})$. The entire reachability set of the part P_3 (in Fig. 3(a)) is $\mathcal{P}_{conn} = \{(2, \{2, 3\}), (3, \{2, 3, 7, 11, 15\}), \cdots, (15, \{3, 7, 11, 14, 15\})\}$.

The remaining part of Algorithm 2 is an exhaustive search algorithm which finds the best solution, i.e. the minimal number of ATE interfaces for part P. First, it checks whether there is a single node that can access all other nodes of the same part (line 13). If this is the case (like in the part P_2 of Fig. 3(a)), then it returns a set with this single node (line 14). If there is no single-node solution, it tests for solutions with two nodes (lines 16 to 19), returning them if this is the case (like in the parts P_1 and P_3 of Fig. 3(a)). If it is not the case, then it tests for three-nodes solution (lines 21 to 25), and so on.

As an example, assuming the part P_3 in Fig. 3(a), Algorithm 2 first determines the $\mathcal{P}_{conn}$, as demonstrated before. Then, it checks whether there is a single node that has access to all other nodes of the part. Next, it checks for solutions with two nodes. The union of the nodes reachable by the nodes 3 and 14 is $\{2, 3, 7, 11, 15\} \cup \{14, 15\} = \{2, 3, 7, 11, 14, 15\} = P_3$, thus combining these two nodes (3 and 14) is sufficient to reach all nodes in the part P_3 and this is the minimal number of ATE interfaces for this part. The Algorithm 2 can also be described recursively. However, we believe that the presented description is more appropriate for the paper due to its simplicity and readability.

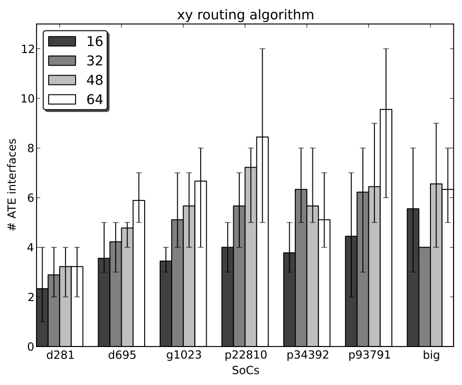
Algorithm 2 [LocateATEIForPart(P)]

```
01. $\mathcal{P}_{conn} = \emptyset$
02. //build the reachability set $\mathcal{P}_{conn}$ for part P
03. for all $n_1 \in P$
04.   $P_{aux}.n = n_1$
05.   $P_{aux}.R = \{n_1\}$
06.   for all $n_2 \in P$ such that $n_2 \neq n_1$
07.     if getPath$(n_1, n_2, P) \neq \emptyset$
08.       $P_{aux}.R = P_{aux}.R \cup \{n_2\}$
09.   $\mathcal{P}_{conn} = \mathcal{P}_{conn} \cup \{P_{aux}\}$
10. //test if there is a single node with access
11. //to all nodes in P
12. for all $P_{conn} \in \mathcal{P}_{conn}$
13.   if $P_{conn}.R = P$
14.     return $\{P_{conn}.n\}$
15. //test if there is a two-nodes solution
16. for all $P_{conn} \in \mathcal{P}_{conn}$
17.   for all $P_{conn2} \in \mathcal{P}_{conn}$
18.     if $(P_{conn}.R \cup P_{conn2}.R) = P$
19.       return $P_{conn}.n \cup P_{conn2}.n$
20. //test if there is a tree-nodes solution
21. for all $P_{conn} \in \mathcal{P}_{conn}$
22.   for all $P_{conn2} \in \mathcal{P}_{conn}$
23.     for all $P_{conn3} \in \mathcal{P}_{conn}$
24.       if $(P_{conn}.R \cup P_{conn2}.R \cup P_{conn3}.R) = P$
25.         return $P_{conn}.n \cup P_{conn2}.n \cup P_{conn3}.n$
26. //test if there is a four-nodes solution
27. ⋮
```

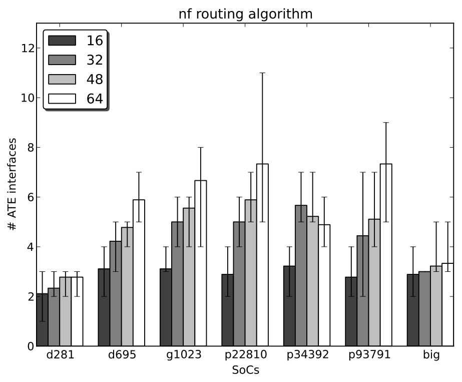
V. Experimental Results

A. Experimental Setup

This Section evaluates several SoC configurations where each SoC is evaluated with five different routing algorithms: XY, minimal Negative First (NF-min), minimal West First (WF-min), non minimal NF (NF), and non minimal WF (WF) routing algorithms [12].

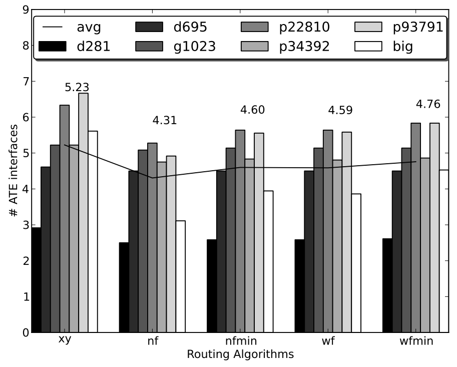


(a) XY

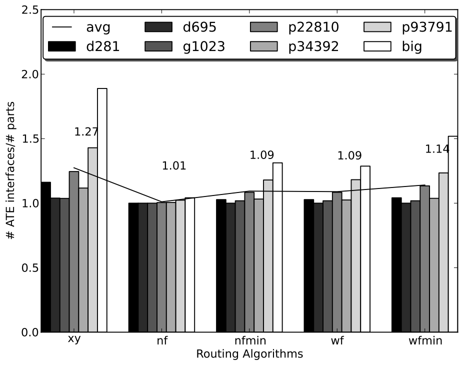


(b) NF

Fig. 4. Number of ATE interfaces per SoC with XY and NF routing algorithms. The columns represent the average number of ATE interfaces for ten SoC placements, considering 16, 32, 48, and 64 test pins. The error bars over each column represent the number of ATE interfaces for the best and worst placements.



(a) Total # of ATE interfaces



(b) Total # of ATE interfaces/# of parts

Fig. 5. Average number of ATE interfaces per routing algorithm.

Two categories of routing algorithms are used: deterministic and partially adaptive. The partially adaptive routing algorithms can be *minimal* or *non minimal* in terms of the path length. Minimal routing algorithms are those where a bounding box virtually exists between the source and destination, and it implies that only decreasing distances from source to destination are valid. Non minimal routing algorithms allow increasing the distance from source to destination.

The following systems from ITC'02 SoC Test Benchmarks [13] were modified to include a mesh-based NoC (the NoC dimensions, i.e. the number of routers for each system is in parentheses): d281 (3,3), d695 (3,3), g1023 (4,3), p22810 (5,5), p34392 (4,4), and p93791 (6,5). The number of cores of the system defines the NoC dimension. There is also the so called 'big(9,9)' SoC created to test the scalability of the proposed model. This SoC is placed in a 9×9 mesh with 117 cores of the five biggest ITC'02 SoC Test Benchmarks. Ten core placements were randomly generated for each SoC and each SoC is tested with 16, 32, 48, or 64 test pins. The position of cores in the NoC and the number of test pins generate different test partitions and each partition requires a different ATE interface placement. In total, 7 SoCs $\times$ 10 core placements $\times$ 4 different number of test wires result in 280 SoC configurations.

B. Results

Fig. 4 presents the average number of ATE interfaces of a SoC configuration with XY and NF routing algorithms because they are, respectively, the cases with the most and the least number of ATE interfaces. The columns of Fig. 4 represent the average number of ATE interfaces considering ten different SoC placements. The error bars represent the placement which resulted in the minimal and maximal number of ATE interfaces. This fluctuation in the number of ATE interfaces is related to the generated test partition for each combination of core placement and number of test pins.

Fig. 4 shows that *the number of ATE interfaces is not related to the size of the NoC*. For instance, both *d281* and *d695* have the same size (3x3) but the later has almost twice the number of ATE interfaces. The SoC *big* has 81 routers but has smaller number of ATE interfaces than *p22810* and *p93791* with 25 and 30 routers, respectively. The reason is related the the structure of the SoC. SoCs whose core test weight are evenly distributed tend to require partitions with more parts where each part requires a small number of test pins. *The number of ATE interfaces is related to the number of parts of the resulting test partition*.

Fig. 4 also shows the *relationship between the routing algorithm and the core placement*, which was randomly generated. It can be observed that the error bars for XY are bigger than the error bars for NF. This is related to the adaptiveness of

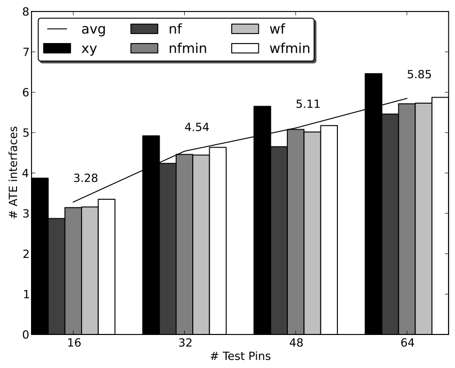


Fig. 6. Average number of ATE interfaces per number of test pins.

the routing algorithm. As a consequence, the core placement results in a greater variation on the number of ATE interfaces when deterministic routing algorithms are used.

Fig. 5(a) illustrates the average number of ATE interfaces per routing algorithm. Fig. 5(b) has an equivalent information, but normalized by the number of parts. For instance, if a SoC configuration has 5 ATE interfaces and 2 parts, then this chart shows $\frac{5}{2} = 2.5$ ATE interfaces per part. The absolute minimal number of ATE interfaces (one per part) could be achieved if we assumed, for instance, the source-based routing approach where any connected node can be accessed by annotating the required turns. Comparing this ideal circumstance with the evaluated routing algorithms, we can see in Fig. 5(b) that the XY routing algorithm requires 27% more ATE interfaces then the ideal case. On the other hand, NF routing algorithm requires only 1% more ATE interfaces. This difference among the algorithms is due to the adaptiveness of the algorithms. In this case, XY has no adaptiveness and NF, among the evaluated routing algorithm, is the one with most adaptiveness.

Fig. 6 illustrates the average number of ATE interfaces per number of test pins. This result shows that the *number of test pins has a relevant effect on the number of ATE interfaces than the routing algorithm*. For instance, increasing the number of test pins from 16 to 32 increases in 1.3 the number of ATE interfaces; from 16 to 64 adds 2.6 ATE interfaces to the system, almost duplicating the total number of ATE interfaces. The reason is related to the test scheduling algorithm which tends to create partitions with more parts when more test pins are available, increasing the test parallelism.

The paper [11], where the ATE interface design is presented, evaluates its silicon area. The results show that the area of an ATE interface varies from about 1000 equivalent gates to 8000, for NoC data width from 8-bits to 64-bits, respectively. An 32-bit ATE interface uses 4300 gates. Then, the silicon area overhead to implement all ATE interfaces can be estimated by the number of gates of a single ATE interface times the number of ATE interfaces ($\sum_{A \in \mathcal{A}} |A|$). For instance, the example in Fig. 1(a) with 5 ATE interfaces and a 32-bit NoC channel requires $5 \times 4300 = 21500$ gates. In terms of area overhead, the proposed approach might need more ATE interfaces then previous approaches. However, the efficiency of the previous approaches presented in Section II is totally dependent of the **given** ATE interfaces, affecting negatively the SoC test time. The new approach, on the other hand, defines the minimal ATE interfaces without sacrificing the SoC test time since ATE interfaces and SoC test time are optimized in different steps.

VI. Conclusions

This paper presented a new NoC TAM test method divided in two steps, where the second step is the focus of this paper. This paper presented an algorithm to determine the minimal number of ATE interfaces for a test partition, reducing the silicon area for the test circuitry. The results show that, on average, the routing algorithm, the number of test pins, and the core placement have a relevant influence on the number of ATE interfaces. The combination of the proposed 2-step test flow created a new NoC TAM test method that, compared to previous NoC TAM approaches, is pottentially more efficient because it eliminates unnecessary constraints from the test scheduler.

Acknowledgements

This work was developed while Alexandre was supported by postdoctoral scholarship from Capes/PNPD, grant number 02388/09-0. Fernando Moraes is supported by CNPq, FAPERGS, and CAPES, projects 302625/2012-7, 10/0814-9, 708/11, respectively.

References

[1] S. K. Goel and E. J. Marinissen, "SOC test architecture design for efficient utilization of test bandwidth," *ACM TODAES*, vol. 8, no. 4, pp. 399–429, Oct. 2003.

[2] S. O. E. Lindholm, J. Nickolls and J. Montrym, "NVIDIA tesla: A unified graphics and computing architecture," *Micro, IEEE*, vol. 28, no. 2, pp. 39–55, 2008.

[3] S. R. Vangal *et al.*, "An 80-tile sub-100-w teraFLOPS processor in 65-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, 2008.

[4] E. Cota and C. Liu, "Constraint-driven test scheduling for NoC-based system," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 25, no. 11, pp. 2465–2478, 2006.

[5] F. Yaun, L. Huang, and Q. Xu, "Re-examining the use of network-on-chip as test access mechanism," in *Proc. DATE*, 2008, pp. 808–811.

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

[7] C. Liu, V. Iyengar, J. Shi, and E. Cota, "Power-aware test scheduling in network-on-chip using variable-rate on-chip clocking," in *Proc. VTS*, 2005, pp. 349–354.

[8] C. Liu and V. Iyengar, "Test scheduling with thermal optimization for network-on-chip systems using variable-rate on-chip clocking," in *Proc. DATE*, 2006, pp. 652–657.

[9] J. M. Nolen and R. N. Mahapatra, "Time-division-multiplexed test delivery for NoC systems," *IEEE Design & Test of Computers*, vol. 25, no. 1, pp. 44–51, 2008.

[10] A. M. Amory, C. Lazzari, M. S. Lubaszewski, and F. G. Moraes, "A new test scheduling algorithm based on networks-on-chip as test access mechanisms," *Journal of Parallel and Distributed Computing*, vol. 71, no. 5, pp. 675 – 686, 2011.

[11] A. M. Amory, M. S. Lubaszewski, and F. G. Moraes, "DfT for the reuse of networks-on-chip as test access mechanism," in *Proc. VTS*, 2007, pp. 435–440.

[12] C. Glass and L. Ni, "The turn model for adaptive routing," *Journal of the Association for Computing Machinery*, vol. 41, no. 5, pp. 874–902, 1994.

[13] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, "A set of benchmarks for modular testing of SOCs," in *Proc. ITC*, 2002, pp. 519–528.