A new test scheduling algorithm based on networks-on-chip as test access mechanisms

Alexandre M. Amory, Cristiano Lazzari, Marcelo S. Lubaszewski, Fernando G. Moraes

PII: S0743-7315(10)00195-4
DOI: 10.1016/j.jpdc.2010.09.008
Reference: YJPDC 2805


Received date: 2 March 2010
Revised date: 14 July 2010
Accepted date: 14 September 2010


This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.
A New Test Scheduling Algorithm Based on Networks-on-Chip as Test Access Mechanisms

Alexandre M. Amory\textsuperscript{a,}\textsuperscript{*}, Cristiano Lazzari\textsuperscript{b}, Marcelo S. Lubaszewski\textsuperscript{c}, Fernando G. Moraes\textsuperscript{a}

\textsuperscript{a}PPGCC-Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil
\textsuperscript{b}INESC-ID, Lisbon, Portugal
\textsuperscript{c}PGMICRO-Departamento de Engenharia Elétrica, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil

Abstract

Networks-on-Chip (NoCs) can be used for test data transportation during manufacturing test. On one hand, NoC can avoid dedicated Test Access Mechanism (TAM), reducing long global wires, and potentially simplifying the layout. On the other hand, (a) it is not known how much wiring is saved by reusing NoCs as TAMs, (b) the impact of reuse-based approaches on test time is not clear, and (c) a computer aided test tool must be able to support different types of NoC designs. This paper presents a test environment where the designer can quickly evaluate wiring and test time for different test architectures. Moreover, this paper presents a new test scheduling algorithm for NoC TAMs which does not require any NoC timing detail and it can easily model NoCs of different topologies. The experimental results evaluates the proposed algorithm for NoC TAMs with an exiting algorithm for dedicated TAMs. The results demonstrate that, on average, 24\% (up to 58\%) of the total global wires can be eliminated if dedicated TAMs are not used. Considering the reduced amount of dedicated test resources with NoC TAM, the test time of NoC TAM is only, on average, 3.88\% longer compared to dedicated TAMs.

\textsuperscript{*}Corresponding Author

Email addresses: alexandre.amory@pucrs.br (Alexandre M. Amory), lazzari@inesc-id.pt (Cristiano Lazzari), luba@ece.ufrgs.br (Marcelo S. Lubaszewski), fernando.moraes@pucrs.br (Fernando G. Moraes)

Preprint submitted to Journal of Parallel and Distributed Computing October 5, 2010
Keywords: SoC test, test scheduling, networks-on-chip

1. Introduction

With the scaling of microchip technology, computation is becoming cheaper than communication. The main reason is that global wires do not scale as transistors and local wires [1] because they communicate across the chip. Global wires can be found in the chip-level communication infra-structures like buses and Networks-on-Chip (NoCs). NoCs [2] may replace global buses in near future due to scalability, parallel communication features, and short global wires.

Modular testing has been proposed as a solution to test such complex SoCs [3]. The conceptual model for modular testing consists of test wrappers (used to switch between functional and test modes), test sources and sinks (used, respectively, to generate test stimuli and to compare the actual test responses to the expected responses), and Test Access Mechanisms (TAMs) (used to transport test data from/to the test pins to/from the Core-Under-Test (CUT)). The most common practice for TAM design is to include dedicated and global test buses used only for test data transportation. Since these TAMs consist of long global wires, dedicated test buses are also subject to the same interconnect problems such as signal integrity, delay, and power dissipation. In an attempt to avoid the long global wires related to the TAMs, Cota et al. [4] proposed the use of the NoC to transport test data, avoiding long global wires both in functional and in test modes.

The main motivation to use a NoC as TAM is to avoid extra long global wires required to implement the dedicated TAMs. It has several potential benefits like reducing area (wiring area and buffers), layout congestion, and power dissipation over long wires with buffers. Additional motivation includes the large internal bandwidth of NoCs that can be used to optimize the test time and the support for globally asynchronous and locally synchronous systems. Testing systems with multiple clocks is noticeable a design challenge. The use of this type of NoC as TAM, which is an open research topic, could simplify the test data transportation for systems with multiples clocks or asynchronous systems.

The original contribution of this papers consists of an adaptable test scheduling algorithm for NoC TAM, a test environment used to quickly evaluate different test architectures, and a method to estimate the wire length.
required by dedicated TAMs. This paper proposes a new test scheduling algorithm for test architecture based on NoC TAM that requires only topology and channel bandwidth information of the NoC. This feature eases the modeling of other NoCs, for example, with different topologies, where most NoC reuse approaches existing so far are specific for mesh-based NoCs. The test environment has been built to ease the generation and analysis of results. It consists of a set of NoC-based SoCs used as benchmarks, scripts to automate the execution and analysis of hundreds of cases, the proposed test scheduling for NoC TAM, a conventional test scheduling approach used for comparison, and the proposed wire length estimation method. As far as we know, this environment is the first to support both NoC-based and dedicated TAMs test architectures and is the first to numerically evaluate the wire length of dedicated TAMs. This type of environment is important because, as demonstrated in the results, both test architectures have advantages and drawbacks, thus, the designer needs an environment like this to quickly evaluate and select the most appropriate solution.

This paper is organized as follows. Section 2 summarizes the basic concepts of SoC testing for dedicated TAMs and presents an example of test scheduling algorithm. The proposed test architecture for NoC TAM is introduced in Section 3. Section 4 presents previous papers about test scheduling for NoC TAM and compares them with the proposed approach. Section 5 presents the proposed wire length estimation method, motivating the use of the existing NoC as TAM. Section 6 presents the proposed test scheduling for NoC TAMs. Section 7 presents four sets of experiments: wire length, test time when NoC is faster than tester, test time when NoC is as fast as the tester, and test time for several NoC topologies. Section 8 presents the conclusion of the paper.

2. Background on SoC Testing

2.1. Test Architecture and Test Scheduling for SoCs

Given the previously presented components of a SoC test architecture (test wrapper, test source/sink, and TAM), the test scheduling and test architecture optimization can be defined as: given a set of modules and a given number of test pins, a test designer has to determine (1) the test architecture type, (2) the number of TAMs, (3) the widths of these TAMs, (4) the assignment of modules to TAMs, and (5) the wrapper design for each module,
Figure 1: Test architecture based on dedicated TAMs [3]. It presents the architectural view (left side) and test scheduling view (right side). The test scheduling has three TAMs of width 3, 4, and 2, respectively, executing test in parallel. Cores A and B are assigned to TAM 1; cores C, D and E are assigned to TAM 2; and core F is assigned to TAM 3. The height of the box (core) is the test bandwidth and the width is core test time. The black part at TAM 2 and 3 represents idle time.

such that (a) test costs (test length, silicon overhead, performance overhead) are minimized and (b) the number of test pins is not exceeded.

One of the major challenges for SoC testing is to design an efficient TAM to link the test sources and sinks to the CUT. The research on SoC testing has been focused on dedicated test bus access, as illustrated in Fig 1, due to its modularity and scalability. Some features of this test architecture are:

- Test application time depends on the test data bandwidth (number of test pins × transfer rate), and on the TAM distribution to cores;

- The SoC testing time can be minimized by distributing the width of each individual TAM proportionally to the amount of test data that needs to be transported to and from the cores connected to the TAM;

- The test data bandwidth might be distributed among several independent TAMs of different width;

- Multiple TAMs on a SoC operate independently and cores on a TAM are typically tested sequentially;

- Cores might be soft or hard cores. In soft cores the number and depth of scan chains can be decided during system integration, enabling balanced scan chains, improving the core test length. In hard cores the internal scan chains are fixed.
2.2. Example of Test Scheduling Algorithm for Dedicated TAMs

This section gives an example of test scheduling algorithm for dedicated TAM, called TR-Architect [3]. This example is detailed since the proposed test scheduling for NoC TAM is built on top of TR-Architect. This algorithm is extended due to its simplicity and feasibility. Other test optimization algorithms are surveyed by Xu and Nicolici [5].

TR-Architect executes the following procedures in sequence: CreateStartSolution, OptimizeBottomUp, OptimizeTopDown, and Reshuffle. Fig. 2 illustrates these procedures.

The procedure CreateStartSolution creates the initial test architecture such that it sorts the modules of the SoC in decreasing order of test data volume, then it assigns the ordered modules to one-bit TAMs. If there are more modules than test pins, then it assigns the remaining modules to the TAM with shortest test length. Otherwise, if there are more test pins than modules, it assigns test pins to the TAM with longest test length.

The procedure OptimizeBottomUp tries to merge the TAM with shortest test length to another TAM such that the freed test pins are allocated to the longest TAM to reduce the SoC test length. Fig. 2(a) shows that the shortest TAM (the one with cores A and D) is merged to another TAM and its $w_3$ test pins are assigned to the longest TAM, reducing the SoC test length.

The procedure OptimizeTopDown tries to merge the TAM with longest test length to another TAM such that the resulting TAM receives the test pins of both original TAMs. Fig. 2(b) shows that the shortest TAM is merged to the longest TAM that receives $w_3$ test pins, reducing the SoC test length.

The procedure Reshuffle tries to reduce the current SoC test length by moving one module of the TAM with longest test length to another TAM.
Fig. 2(c) shows that the module D, inside the longest TAM, is moved to another TAM, reducing the SoC test length.

3. Proposed Test Methodology

This section presents the proposed test approach for NoC TAMs.

3.1. Test Architecture and DfT Modules

Fig. 3 shows the proposed test architecture, its DfT modules, and the protocol conversions along the test data flow. The ends of the test (ATE and CUT) require test streaming, the NoC interface requires a standardized protocol such as OCP (Open Core Protocol, http://www.ocpip.org/), 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 [6, 7].

3.2. Introduction to the NoC Partition Method

We introduce the concept of NoC partition, or just partition, which is equivalent to the concept of partition of a set. A partition is a set of nonempty subsets such that every element is in exactly one of these subsets. The subset of a partition is denoted part.

Note that these definitions are analogous to the definitions of conventional test architecture for dedicated TAMs. The test architecture of a SoC can be seen as a set of TAMs of certain width. The test architecture would be a
partition while a TAM is a part of the partition. There are other similarities between the proposed and the conventional approaches such as:

1. The test pins are distributed among each individual part proportionally to the amount of test data that needs to be transported to/from the cores connected to the parts;
2. There might be multiple parts and they might have different test bandwidth. The sum of the bandwidth of each part do not exceed the total test bandwidth.
3. The underlying DfT (test wrappers and ATE interface) provides guaranteed test streaming at the ATE and at the CUT side;
4. Only one test is carried out at a time within a part;
5. Resources of a part (routers and links) are not shared with other parts;
6. A router belongs to one and only one part.

The features 1 to 4 also applies to test architecture based on dedicated TAMs. As a consequence of the features 4 to 6, there are no packet collisions during the test applications.

3.3. Required Constraints

The following constraints represent limitations imposed by the use of a NoC to transport test data. Such constraints are:

- **Channel bandwidth constraint:**
  It is the test bandwidth that can be assigned to a given part. If it is not respected, the NoC will not be able to deliver the test data correctly;

- **Clustering constraint:**
  This constraint implies that the cores connected to a router must all belong to the same part. It assures that they will be tested sequentially, avoiding packet collision. To implement this constraint the method must know the NoC topology;

- **Neighborhood constraint:**
  The algorithm does not allow disjoint parts, thus, a part $a$ is “connected” if any pair of nodes within part $a$ can be reached without crossing other parts. Otherwise, the part is “unconnected”. When a part is unconnected two different test streams might request the same network resource (a router or a data channel) during the test application, causing packet collision and delaying one of the test streams.
To implement this constraint the method must also know the network topology.

Note that the implementation of these constraints requires few details about the network: the first constraint requires the network bandwidth; the second and the third one requires the network topology, i.e. how the cores and routers are connected to each other, used to determine the neighborhood of the nodes.

3.4. Problem Statement

Let us first define some notations used here and also in Section 6. Let a 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 router and a set of zero or more cores which are connected to this router. Let lower case symbols ($n$) represent a single node, upper case symbols ($P$) represent a set of nodes (or a part of a partition), and slanted upper case symbols ($P$) represent a set of sets of nodes (or a partition of $G$). $t(P)$, $b(P)$, and $w(P)$ are, respectively, the test length (in clock cycles), the test bandwidth (in Mbps), and the part width (it represents the number of test pins assigned to the part and the number of scan chains that can be loaded in parallel in a test wrapper) assigned to part $P$.

Given the maximum sustainable NoC channel bandwidth $b_{noc}$, the maximal test bandwidth $b_{test}$, the test frequency $f_{test}$ (i.e. the frequency of the test logic), and the NoC frequency $f_{noc}$, the total number of test pins is $w_{max} = \left\lfloor \frac{b_{test}}{f_{test}} \right\rfloor$, and the physical channel width is $w_{noc} = \left\lfloor \frac{b_{noc}}{f_{noc}} \right\rfloor$. Given a set of cores and their test-related information (as defined in [7]). Given a graph $G = (V, E)$, representing the NoC topology. The goal of the test scheduling algorithm is to determine a partition $P$ of $G$ such that:

1. $(\sum_{i=0}^{P} w(P_i)) \leq w_{max}$, i.e. the sum of all part widths $w(P_i)$, with $P_i \in P$, do not exceed the number of test pins $w_{max}$;
2. $\forall P_i \in P, (b(P_i) \leq b_{noc})$, i.e. the bandwidth of each of part $b(P_i)$ do not exceed the NoC channel bandwidth $b_{noc}$, where $b(P_i) = w(P_i) \times f_{test}$;
3. For all parts $P \in P$: all nodes are connected, all nodes are assigned to exactly one part, and all parts are disjoint;
4. Total test time $T$ is minimized, where $T = \max_{i=0}^{P} t(P_i)$ and $t(P_i)$ is the test time of a part $P_i \in P$. 

8
The first and the last constraints are, respectively, the test bandwidth and the test time constraints, which are common to all test scheduling algorithms, even the ones for dedicated TAM. The second constraint is the channel bandwidth constraint which is common to all NoC based test schedulers. This constraint can also be represented in terms of maximal part width \(w_{dft}\), defined below. The neighborhood constraint, the third constraint, is embedded into the concept of partition of a set. Finally, the clustering constraint is embedded into the concept of NoC nodes, which represents the clustering of a router and zero or more cores into a single node.

Let us also define the maximal part width \(w_{dft}\), used in Section 7, which represents the maximal number of test pins that can be assigned to a part and the maximal number of parallel scan chains in a DfT module (ATE interface or test wrapper). The maximal part width is \(w_{dft} = w_{noc} \times \left\lfloor \frac{f_{noc}}{f_{test}} \right\rfloor\), i.e. the channel width multiplied by the NoC speedup \(^1\).

4. Prior Work

This section focuses on previous test scheduling algorithm for NoC TAMs, specially in the aspects related to adaptability and the underlying problem formulation.

4.1. Test Scheduling for NoC TAMs

There is an initial problem formulation for test scheduling of NoC TAMs, proposed by [4], which is, as far as we known, followed by all paper about test scheduling for NoC TAMs. This initial problem can be stated as given a mesh-based NoC with XY routing algorithm; a set of cores, its test information, and the location of these cores in the NoC; a set of input ports, its location in the NoC, and its capacity; a set of output ports, its location in the NoC, and its capacity; determine an assignment of cores to input/output pairs such that the total test time is minimized. The problem of NoC-based test scheduling can be summarized as how to efficiently assign input/output pairs to cores without resource conflicting, such that the overall test time is minimized.

Two main approaches have been derived from this initial problem formulation: the packet-based [4] and the dedicated routing path [8] approaches. 

\(^1\)\(w_{dft}\) can also be defined as \(w_{dft} = w_{noc} \times \left\lfloor \frac{f_{noc}}{f_{test}} \right\rfloor\) in order to simplify the hardware design of the bandwidth converters located at the DfT modules.
The packet-based approach schedules test packets and assigns them to pairs of I/O ports, locking the path between the I/O ports and the CUT while the packet is transmitted. Once the resources are freed (I/O ports and the path), a new packet can be scheduled. Packets of the same test set might be scheduled to different I/O ports, i.e. the test of a single core might be feed by different test ports, and a test application might be interrupted. The dedicated-path approach schedules a whole test set instead of every single packet, thus, the difference between both approaches is the granularity of the optimization. The former schedules each test packet independently while the latter schedules the entire core test application. Later Cota and Liu [9] proved that the packet-based scheduling can be reduced to dedicated-path scheduling if we assume that the test packets are sent continuously and the test data is sent/received from the same pair of I/O ports. Thus, the dedicated-path is a special case of the packet-based under restriction.

The initial problem formulation has been extended several times, including new constraints, variables to the problem, or assuming the existence of certain NoC functionality: Cota et al. [10] supports BISTed cores; Cota et al. [11] include power constraints to the test scheduling; Liu et al. [12] assume NoCs with on-chip clocking and Time-Division Multiplexing (TDM) [13] to improve the test data transfer and the channel usage; both [14] and [15] combine thermal constraints and on-chip clock to the scheduling; Liu [16] includes the support to hierarchical cores to the variable clock approach [15]; Dalmasso et al. [17] include the support of compressed data to the algorithm presented in [9]; Nolen and Mahapatra [18] and Hussin, Yoneda, and Fujisawa [19] also assume NoCs with TDM and NoCs with higher frequency than the tester. Finally, Liu, Link, and Pradhan [20] include the NoC test into the algorithm proposed in [8]. While some routers are tested, other routers are transporting test data in functional mode.

Yuan, Huang, and Xu [21] tackle the problem of defining general test approaches for NoC TAM. The authors argue that most existing NoC TAM approaches are based on a single NoC model (e.g. SoCIN [22]), and it makes difficult, even impossible, to port the solution to other NoCs. They present theoretical lower bounds for testing generic NoCs which assumes the initial problem formulation presented before. This lower bound is compared to a lower bound for dedicated TAMs. It is concluded that the NoC TAM problem, as it is usually formulated, implies in longer test time compared to conventional dedicated TAM.
4.2. Comparison of Test Architectures

Previous approaches require a major effort to adapt them to different networks because they internally implement a cycle accurate NoC model used to schedule the test paths or test packets. A large set of information (topological, algorithm, and timing details) is required to implement this NoC model. For instance, the NoC timing information is required to calculate when a certain resource (I/O pair or link) will be freed for the next test. One reason for the lack of adaptability is that they either deal with the NoC resource competition in the test scheduling, requiring cycle accurate NoC model, or they assume that there are some functional resource that deal with the NoC resource competition, like circuit switching or virtual channels. The first approach is very difficult to adapt to other NoCs while the second one is impossible to adapt if the functional resource is not available in the NoC. Even if a functional resource, like virtual channels, is available the test scheduling should model the functional and timing aspects of a specific virtual channel implementation.

This paper presents a test scheduling algorithm easier to adapt to other NoCs because it avoids at the test scheduler any situation that could lead to resource competition in the network, without relying on a specific functional resource of the NoC, NoC design (functional or timing related) details.

Previous approaches assume that I/O ports (their width and location) and routing algorithm are given and the test scheduling must assign one of these fixed resources to test the CUTs. Considering the example in Fig. 4, the test of c0, c1, and c2 cannot be concurrent due to the location of the I/O ports. Assuming an XY routing algorithm these three cores share part of path to the output ports, i.e. the link between core 2 and 5 is required by the three cores.

In this aspect the proposed approach is similar to the problem formulation presented in Section 2.1. Just the maximal test bandwidth (or number of test pins) is given. The location of test pins is considered less relevant than optimizing the SoC test time. As a consequence, the proposed approach does not have a constraint like illustrated in Fig. 4. The location of test pins is still required in our approach because it determines the location of the ATE interface, however, it is calculated after the test scheduling with no influence on test time.
5. Wire Length Estimation Model for Dedicated TAMs

The proposed model assumes that the SoC is represented by tiles \(^2\) which are evenly distributed in the entire SoC area such that the distance between any two neighbor tiles is the same. Each tile can have zero or more cores and exactly one network interface. The rest of the system (clock and reset tree, test wires, and NoC) are distributed among the tiles. This description is coherent to homogeneous NoC-based systems, where the tiles have similar logic, like in an homogeneous MPSoC system.

The proposed method counts the minimal number of hops required to reach all modules within a dedicated TAM. The wire length between cores within the same tile is supposed to be zero, while the wire length between cores in different tiles is equivalent to the number of hops between these two tiles. Fig. 5 represents a NoC-based SoC using three dedicated TAMs, and it is used to illustrate the proposed wire length estimation method. Each box represents a tile which consists of one router (identified by the number outside parentheses) connected to zero or more cores (identified by the number within parentheses).

The total number of wires to implement the set of TAMs of a SoC is defined as

\[
w_{SoC} = \sum_{i=1}^{n} (h_i + 1) \times w_i \times 2
\]

where \(n\) is the number of TAMs, \(h_i\) is the minimum number of hops to
implement TAM\textsubscript{i} plus one hop representing the wires from the I/O test pins to the closest core of the TAM. Finally, \(w_i\) is the width of the TAM\textsubscript{i}. Since there must be wires also for the test responses, then, the number of wires is multiplied by two.

Let us assume the following TAM assignment, illustrated in Fig. 5, created by a conventional test scheduling algorithm: \(\text{TAM1}=\{c1,c5,c6,c8,c9\}\), \(\text{TAM2}=\{c4,r01,r11,r12,r02\}\), \(\text{TAM3}=\{c0,c2,c3,c7,r00,r10,r20,r21,r22\}\) assuming 16 test pins to connect the chip to the ATE. The width of these TAMs are 7, 5, 4 wires, respectively.

For instance, TAM1 has five cores where two of them are located in the tile 01 and the remaining cores are located in tiles 11, 10, and 20, thus, the minimum distance between these four tiles is three hops (see the continuous fat line in Fig. 5). Since the width of TAM1 is seven test wires, then it results in \((3 + 1) \times 7 \times 2 = 56\) wires to implement the TAM1. The minimum number of hops for TAM2 and TAM3 are 4 and 5 hops, respectively. Finally, the total TAM wiring for this example is 154 wires \(((3 + 1) \times 7 \times 2 = 56\) for TAM1, \((4 + 1) \times 5 \times 2 = 50\) for TAM2, and \((5 + 1) \times 4 \times 2 = 48\) for TAM3).

5.1. Evaluation of the Model

Let us take the d695 SoC presented in Fig. 5 as an example to compare the actual and the estimated wiring for dedicated TAMs.

As calculated before, the estimated number of test wires required to implement dedicated TAMs is 154. In an \(i\)-by-\(j\) mesh, there are \(2 \times (i \times (j - 1) + j \times (i - 1))\) channels. For example, the system d695 is a 3-by-3 mesh,
Figure 6: Layout of the d695 SoC with dedicated TAMs. The squares are the HeMPS tiles while the horizontal and vertical lines are the Hermes NoC with dedicated TAMs.

thus, it has 24 channels of 32 bits or $24 \times 32 = 768$ wires\(^3\). Thus, according to the proposed model, close to $\frac{154}{768} = 20\%$ of the global wires of the chip are required to implement the dedicated TAMs.

Layout analysis is required to evaluate the actual wiring for dedicated TAMs. We implemented in VHDL the system presented in Fig. 5 using dedicated TAMs. The tiles are supposed to be hardcore while the NoC and the dedicated TAMs are softcore. Both the tile and the NoC are based on the HeMPS MPSoC [24] configured with buffers of size 16 and 32-bit channel width. The system has been synthesized to the library UMC 130nm.

The layout consists of two steps. The first step is to create the blackbox of the HeMPS tile with a 32-bit MIPS processor, network interface, DMA, and 16KB dual-port memory. It resulted in a box of $2560 \times 2550 \mu m$ of area. The second step connects the tile to the HeMPS NoC [24], to the blackbox. Finally, the dedicated TAMs, depicted in Fig. 5, are included into the SoC.

After the SoC setup, Cadence\textsuperscript{TM} tools were used for logic and physical synthesis. Fig. 6 illustrates the resulting layout of the SoC based on dedicated TAMs. The resulting wires are classified into four classes of wires: local, global, clock, and TAM wires. “Local wires” are required to implement the internal router logic. “Global wires” are used to connect the routers to each other and the router to the tile, excluding TAM and clock wires. “Clock wires” are used to implement clock and reset trees. Finally, “TAM wires” represent the dedicated TAMs. Table 1 shows the distribution of wire length among these types of wires.

\(^3\)This wire count does not consider control wires used to implement the protocol.
Table 1: Distribution of wires in a NoC-based SoC with dedicated TAMs.

<table>
<thead>
<tr>
<th></th>
<th>length (%)</th>
<th>length (%)</th>
<th>length (%)</th>
<th>length (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>67.32</td>
<td>19.42</td>
<td>8.11</td>
<td>5.15</td>
</tr>
</tbody>
</table>

The overhead of dedicated TAMs is small (5.15%) compared to the total wire length of the NoC. However, it consists of $\frac{5.15}{19.42} \approx 26.5\%$ of the global wires. Recall that the proposed model estimated that 20% of the global wires would be used to implement dedicated TAMs. The difference between the actual (26.5%) and estimated (20%) TAM wiring is due to routing congestion which is not captured in the proposed model. Section 7.3 presents average results for more SoCs.

This model must be faced as a lower bound of TAM wire length, thus, the model will always estimate shorter wires than actual layout. Figures not modeled by this method are kept to the minimal value. For instance, it does not model layout congestion (actual layout might need longer wire to avoid congested areas) and order of cores assigned to the TAM (for example, wire length between cores A,B,C might be longer than A,C,B, depending on the position of the cores). The model assumes that the layout congestion is zero and the order of the cores are such that the minimum wire length is required. On the other hand, this model is much easier and faster than full physical synthesis of large NoC-based SoCs, thus, it is ideal to explore several design alternatives.

6. Proposed Test Scheduling for NoC TAMs

Algorithm 1 presents the main procedures for the proposed test scheduling, which are detailed in the next sections.

Algorithm 1 [Main Test Scheduling Algorithm for NoC TAM]

01. ModifiedCreateStartSolution;
02. FixStartSolution;
03. ModifiedOptimizeBottomUp;
04. ModifiedOptimizeTopDown;
05. ModifiedReshuffle;
6.1. Auxiliary Procedures

Let us first introduce the sub-procedures \textit{unconnectedNodes}(P), \textit{neighborParts}(P), \textit{bridgeNodes}(P_1, P_2), and \textit{TestTime}(P, w) used in the proposed test scheduling algorithm presented in Section 6.2.

- The procedure \textit{unconnectedNodes}(P) returns the subset of nodes in P which does not have access because the part is segmented and unconnected. For instance, let us use the system in Fig. 7(a), then \textit{unconnectedNodes}(b) = \{20\} because the node 20 is not connected to the rest of the part.

- The procedure \textit{neighborParts}(P) returns the set of parts directly connected to the part P. For example, according to Fig. 7(b), \textit{neighborParts}(d) = \{a, b\} (note that node 22 is neighbor of the parts a and b, nodes 30 and 31 are neighbors of the part a only) and \textit{neighborParts}(c) = \{a, b\}. \textit{neighborParts}(n) returns the set of parts directly connected to the node n. For instance, assuming Fig. 7(b), \textit{neighborParts}(22) = \{a, b\}, \textit{neighborParts}(32) = \emptyset.

- The procedure \textit{bridgeNodes}(P_1, P_2) returns the set of nodes in part \(P_1\) in the border with the part \(P_2\). Taking Fig. 7(b) as an example, \textit{bridgeNodes}(b, a) = \{11\} and \textit{bridgeNodes}(a, b) = \{10, 21\}.

- The procedure \textit{TestTime}(P, w) calculates the test length of the part P assuming \(w\) test pins.

6.2. Main Procedures

The procedure \textit{ModifiedCreateStartSolution}, line 1 of Algorithm 1, has the same goal as the original one, but the channel bandwidth constraint is checked whenever test pins are added to the longest part. If the constraint is met, it means that the maximal number of test pins for this part (\(w_{dft}\)) has been reached and it is not possible to have further optimization. Then the procedure stops and the remaining test pins are not used.

The parts a, b, c, and d, Fig. 7(a), generated by first procedure might be unconnected, violating the neighborhood constraint. The procedure \textit{FixStartSolution}, line 2 of Algorithm 1, connects these parts minimizing the SoC.
test length. Fig. 7(b) illustrates the resulting graph without unconnected parts.

Algorithm 2 exhaustively moves an unconnected node from its current part to the neighbor part with the shortest test length. Thus, this algorithm ensures that all nodes are connected to a part and that there is no unconnected part. Variable $\mathcal{P}$ represents all sets of parts of the system. For all parts defined in ModifiedCreateStartSolution, the procedure unconnectedNodes returns the set of nodes in the part $P$ which are not connected, called $P_{unc}$. Next, for every unconnected node $r_{unc}$ in the set $P_{unc}$, Algorithm 2 finds the part $P_{min}$ among the set of neighbor parts of the unconnected node $r_{unc}$ with the minimal test length (line 4). Then, the unconnected node $r_{unc}$ is moved from the part $P$ to the $P_{min}$ (line 5) and the test lengths of $P$ and $P_{min}$ are updated (lines 6 and 7).

**Algorithm 2** [FixStartSolution]

```
01. for all $P \in \mathcal{P}$
02.   for all $P_{unc} \in \text{unconnectedNodes}(P)$
03.     for all $r_{unc} \in P_{unc}$
04.       find $P_{min}$ for which $t(P_{min}) = 
                    \min_{\text{neighborParts}(r_{unc})} t(J);
05.       $P_{min} := P_{min} \cup \{r_{unc}\}; P := P - \{r_{unc}\}$;
06.       $t(P_{min}) := \text{TestTime}(P_{min}, w(P_{min}));$
07.       $t(P) := \text{TestTime}(P, w(P))$
08.     }
```
For example, let us assume the system illustrated in Fig. 7(a). Let us say that \( P = b \), then, \( \text{unconnectedNodes}(b) = r_{unc} = \{20\} \), i.e. the node 20 is not connected to the rest of the part \( b \). Next, the procedure \( \text{neighborParts}(r_{unc}) = \{a, c\} \) checks the possible parts to move the unconnected nodes. In this case the node 20 is next to the parts \( a \) and \( c \). The chosen part is the one with the shortest test length. Let us assume that part \( a \) is shorter than \( c \), then \( P_{\text{min}} = \{a\} \). Finally, node 20 is moved from the part \( b \) to \( a \) since this part is the neighbor part with the shortest test length.

The procedures \( \text{ModifiedOptimizeBottomUp} \) and \( \text{ModifiedOptimizeTop-Down} \) (lines 3 and 4, respectively, of Algorithm 1) have similar modifications. They check the channel bandwidth constraint (\( w_{\text{dft}} \)) whenever test pins are added to a part, but they also check the neighborhood constraint by allowing only adjacent parts as merge candidates.

The procedure \( \text{ModifiedReshuffle} \), line 5 of Algorithm 1, tries to move one node in the part with the longest test length to another part to improve the SoC test length. Conversely to the original algorithm, the one presented in Algorithm 3 has to check the neighborhood constraint when moving nodes from one part to another one, i.e. parts cannot become unconnected.

The algorithm finds the part \( P_{\text{max}} \) with the longest test length (line 3). If \( P_{\text{max}} \) has only one node, then it is impossible to move one item (line 5) and the procedure is finished. Otherwise, for all parts \( P_{\text{cand}} \), neighbor of \( P_{\text{max}} \), it finds the set of nodes of \( P_{\text{max}} \) connected to the part \( P_{\text{cand}} \) (line 8). The set of these nodes in the border of \( P_{\text{max}} \) and \( P_{\text{cand}} \), called \( P_* \), are the ones that can be moved, however, some constraints must be checked first. The algorithm checks if moving a node \( r_* \in P_* \) does not increase the SoC test length (lines 10 to 14). If the move is accepted in terms of test length, then it checks if moving the node \( r_* \) will not unconnect the part \( P_{\text{max}} \) (line 15). If not, the move is accepted and the new SoC test length is updated (lines 16 to 19).

**Algorithm 3 [ModifiedReshuffle]**

```
01. improve := true;
02. while improve{

```
find $P_{\text{max}}$ for which $t(P_{\text{max}}) = \max_{J \in P} t(J)$;
PartFound := false;
if $|P_{\text{max}}| = 1$ { improve := false; }
else{
  for all $P_{\text{cand}} \in \text{neighborParts}(P_{\text{max}}) \land \neg \text{PartFound}$ {
    $P_* := \text{bridgeNodes}(P_{\text{max}}, P_{\text{cand}})$;
    for all $r_* \in P_* \land \neg \text{PartFound}$ {
      $P_{\text{max}2} := P_{\text{max}} - \{r_*\}; w(P_{\text{max}2}) := w(P_{\text{max}})$;
      $t(P_{\text{max}2}) := \text{TestTime}(P_{\text{max}2}, w(P_{\text{max}2}))$;
      $P_{\text{cand}2} := P_{\text{cand}} \cup \{r_*\}; w(P_{\text{cand}2}) := w(P_{\text{cand}})$;
      $t(P_{\text{cand}2}) := \text{TestTime}(P_{\text{cand}2}, w(P_{\text{cand}2}))$;
      if $t(P_{\text{cand}2}) < \text{TestTime}(P_{\text{max}})$ {
        if $\text{unconnectedNodes}(P_{\text{max}2}) = \emptyset$ {
          PartFound := true;
          $P_{\text{max}} := P_{\text{max}2}; P_{\text{cand}} := P_{\text{cand}2}$;
          $t(P_{\text{max}}) := \text{TestTime}(P_{\text{max}}, w(P_{\text{max}}))$;
          $t(P_{\text{cand}}) := \text{TestTime}(P_{\text{cand}}, w(P_{\text{cand}}))$;
        }
      }
    }
  }
  improve := PartFound;
}

For example, considering the system in Fig. 7(b) and assuming that part $d$ corresponds to $P_{\text{max}}$, i.e. part $d$ has the longest test length. The parts $a$ and $b$ are the candidate parts because they are next to $d$. So $P_{\text{cand}}$ assumes $a$ and $b$. When $P_{\text{cand}} = a$, $\text{bridgeNodes}(P_{\text{max}}, a) = \{22, 30, 31\}$ because these are the nodes of part $d$ that could be moved to part $a$ since they are next to part $a$. When $P_{\text{cand}} = b$, $\text{bridgeNodes}(P_{\text{max}}, b) = \{22\}$ because node 22 is next to the part $b$. Let us assume that when it moves the node 31 and 22 out of the part $d$ the SoC test length reduces. However, when the node 31 is moved the node 30 will be unconnected, which is not allowed. Finally, node 22 is moved from the part from $d$ to $b$ because it passes in all constraints and it reduces the SoC test length.
7. Experimental Results

Section 7.1 introduces the test environment used to generate the presented results. Section 7.2 presents the setup common to the four experiments presented in the following sections. The first experiment, presented in Section 7.3, evaluates the wire length required to implement dedicated TAMs, justifying the advantage of using NoC as TAM. Section 7.4 compares the test length generated by both test schedulers (dedicated and NoC). Section 7.5 evaluates the proposed test scheduling when the NoC bandwidth is lower than the test bandwidth. Section 7.6 demonstrates the proposed test scheduling with NoCs of different topology and the impact of these topologies on the test length.

7.1. Developed Test Environment

The developed test environment consists of two test flows: one based on dedicated TAMs and the other based on NoC TAM. The former consists of a test wrapper optimizer [25], a test scheduler [3] and the proposed wire length estimator described in Section 5. The later flow consist of test wrapper optimizer [7] and the proposed test scheduler presented in Section 6. All test related information of the SoCs presented in Section 3.4 as well as the results are described in XML format. There are several Python scripts that automate running both test flows and analyzing the results. The environment is very useful for test designers to evaluate and compare different test strategies quickly because it needs easy setup and has scripts for automated result analysis.

One of the most distinguishing features of the proposed test scheduling algorithm is its easy setup and adaptability for different NoCs. The reason for this is the small number of NoC related information required: only NoC channel bandwidth and NoC topology. The other reason is related to the implementation. The topology, for instance, is modeled as an abstract class that implements topology-dependent virtual methods, that can be extended, and topology-independent methods, that work for any topology. To support a new topology one just has to extend the abstract class, build a dynamic library, and describe the graph topology in a readable format (XML and 'dot' formats). The topology library is loaded at run time, thus, no recompilation is required to switch NoC topologies. Moreover, no modification in the main test scheduling is required.
7.2. Experimental Setup

The first step is to build the NoC-based SoCs for the evaluation. The following systems from ITC’02 SoC Test Benchmarks [26] were modified to include a NoC (the NoC size, i.e. the number of routers, for each system is in parentheses): d281 (3,3), d695 (3,3), g1023 (4,3), h953 (3,3), p22810 (5,5), p34392 (4,4), p93791 (6,5), u226 (3,3). The size of the NoC is selected based on the number of cores of the system. The SoCs f2126 and q12710 were excluded because they have only four cores and the SoC a586710 has only seven cores. It does not make sense to evaluate TAM wire length in such small SoCs. The t512505 SoC has been excluded because it has a bottleneck core, core 31, which requires about 88% of the entire SoC test data.

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 $9 \times 9$ mesh with 117 cores. These cores are the cores of the five biggest ITC’02 SoC Test Benchmarks, which are the SoCs p22810, p34392, p93791, t512505, and a586710.

The ITC’02 SoC Test Benchmarks were modified to include the NoC into the SoC. First, each core receives two OCP-like ports, as illustrated in Fig. 8, one port to receive and the other to send data from/to the NoC. Each port has 39 control terminals and 32 data terminals plus the original number of terminals of a core defined in the ITC’02 SoC Test Benchmark. For example, the original module 1 of d281 SoC has 60 inputs and 26 outputs, thus, the modified module has 60 + 26 + ($2 \times (39 + 32)$) terminals.

Second, the NoC is generated. For the sake of simplicity it is assumed that all routers in a system are identical, i.e. they have five bi-directional ports and the same number of test patterns. The number of terminals of a router is $5 \times (2 \times (2 + 32))$ (two control terminals and 32 data terminals multiplied by five bi-directional ports). Moreover, the router has 50 scanable flip-flops related to internal control logic. The test scheduler considers the routers as soft-cores.

Third, the cores of each SoC are placed on the NoC. Ten random placements have been generated for each SoC because the placement has an impact on the TAM wire length. The cores have been placed in the NoC randomly such that if the number of cores is greater than the number of routers, it makes sure that all routers have at least one core and no router receives more than two cores. It also makes sure that all placements are different from each other. At this point the SoCs have been generated.

Assuming that the NoC frequency is twice faster than the test frequency ($f_{noc} = f_{test} \times 2$, defined in Section 3.4, page 8) and the NoC channel width
has $w_{noc} = 32$ bits, then, the maximal part width is $w_{dft} = \left\lfloor 32 \times f_{\text{test}} \times 2 \right\rfloor = 64$ bits. In order to be conservative, we assume that the maximal part bandwidth is 75% of the maximal channel bandwidth of a NoC ($b(P) = 0.75 \times b_{noc}$). For instance, a circuit switching NoC could provide full bandwidth for testing while wormhole packet-switching NoC could not since bandwidth is lost with, for example, headers and trailers. Finally, the maximal part width ($w_{dft}$) is $0.75 \times 64 = 48$ bits.

7.3. Wire Length Required to Implement Dedicated TAMs

Given the previously presented experimental setup, the wire length estimation method is executed to evaluate the overhead of the dedicated TAMs. The detailed results for each of the 630 executions (9 SoCs $\times$ 10 placements $\times$ 7 number of test pins) have been omitted due to the lack of space, however, the average results for the average/best/worst placements are, respectively, 24.6%, 22%, and 27% (the placement of the cores in the chip has influence on the wire length because the cores might have different distances for each placement.). It means that, on average, about 24.6% of the SoC global wires are used to implement dedicated TAMs. For some systems, like d695 with 64 test pins, the TAM wiring can be about 58% of the SoC global wires. Note that the model is optimistic, i.e. it is a lower bound on the wiring overhead of dedicated TAMs. It means that the actual wiring for dedicated TAMs is larger.

7.4. Comparing the SoC Test Length of Both Approaches

Given the previously presented experimental setup, this section compares the test scheduling for dedicated TAM and the proposed method. Table 2 presents the results assuming different number of test pins ($w_{\text{max}}$). The columns conv, avg-reuse, best-reuse, worst-reuse, and diff show, respectively,
the SoC test length considering the dedicated TAM, the average/best/worst SoC test length for the proposed approach considering 10 different placements, and the difference between both test architectures. Positive percentage represents that the proposed architecture is slower than the dedicated one.

In most cases the proposed approach is slower, mainly because it uses the same underlying algorithm for both cases and it includes the constraints described in Section 3.3 to the proposed approach. However, in few cases the NoC TAM is faster because the resulting schedule is the same (or equivalent) of the conventional schedule and the number of test pins assigned to the parts is multiple of the channel width, which is 32 bits [7]. This situation causes the test wrappers to assign the test patterns to the CUT slightly faster due to the parallel-to-serial converters at the wrapper interface with the NoC [7]. In few situations the resulting test schedule for NoC TAM is indeed better than the equivalent one for conventional core. It happens because the constraints might force different merge candidates which might result in a shorter SoC test length. This case happens, for example, with the best placement of d281 SoC with 16 test pins. To check this statement the scheduling for NoC TAM has been assigned to the conventional scheduler and the SoC test length for conventional TAM has been reduced from 315117 to 310561 clock cycles. The rest of this section explains how the constraints act in the test scheduling for NoC TAMs for each case study.

The worst placement for p22810 and g1032 SoCs deviate negatively from the average test length because at least one of the 10 placements of these SoCs generated a node which has two large cores, which individually would not be considered bottlenecks. The clustering constraint causes this effect because it combines all cores which are connected to a router, creating a ‘super core’. The SoC placement presents a smaller impact on the other SoCs.

Finally, the average difference in test length for the average/best/worst placements are, respectively, 3.88%, 0.87%, and 9.81% (bottom of Table 2). It means that, on average, the NoC TAM approach results in tests 3.88% longer than dedicated TAMs.

The solutions to reduce the SoC test length for these situations mentioned before would be to: (i) increase the NoC channel bandwidth by increasing the NoC clock and/or increasing the physical channel width; (ii) increase the percentage of usage of the available NoC bandwidth to more than 75%;
Table 2: SoC test length for both test approaches.

<table>
<thead>
<tr>
<th>SoC</th>
<th>conv</th>
<th>avg-reuse diff(%)</th>
<th>best-reuse diff(%)</th>
<th>worst-reuse diff(%)</th>
<th>best-reuse diff(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>big</td>
<td>56</td>
<td>40</td>
<td>32</td>
<td>24</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>48</td>
<td>32</td>
<td>24</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>32</td>
<td>24</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>24</td>
<td>16</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2 continued...

<table>
<thead>
<tr>
<th>SoC</th>
<th>64</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
</tr>
</thead>
</table>

average 3.88 0.87 9.81
(iii) design the cores of the SoC such that their test data volume is evenly distributed among the cores, avoiding bottleneck cores; (iv) place the cores on the NoC such that the test data volume of a node is evenly distributed, reducing the impact of the clustering constraint.

7.5. Test Time Assuming a Slower NoC

This section compares a faster and a slower NoC to evaluate the impact of channel bandwidth constraint in low bandwidth NoCs.

Let us assume that the NoC frequency is equal to the test frequency \( f_{noc} = f_{test} \) and the NoC channel width has \( w_{noc} = 32 \) bits, then, the maximal part width is \( w_{dft} = \left\lfloor 32 \times \frac{f_{test}}{f_{noc}} \right\rfloor = 32 \) bits. Assuming that only 75% of the bandwidth is actually used to transport useful data, then, \( (w_{dft}) \) is \( 0.75 \times 32 = 24 \) bits. Up to 24 scan chains can be loaded with new bits every clock cycle.

Fig. 9(a) compares the test length of \( conv, reuse24 \) (representing the slower NoC), and \( reuse48 \) (representing the faster NoC, presented in Section 7.4) for the big SoC. Fig. 9(a), \( reuse24 \), shows the SoC test length is not decreased when more than 40 test pins are available. This effect is caused by the channel bandwidth constraint, because some part is requiring more than 24 test pins to reduce its test length. On the other hand, the results in Fig. 9(a) for \( reuse48 \) and \( conv \) seems to be almost the same.

Fig. 9(b) evaluates both \( reuse24 \) and \( reuse48 \) (representing average results for all 9 SoCs) in comparison with \( conv \). Note that, given enough NoC bandwidth, see \( reuse48 \), the difference between \( conv \) and \( reuse48 \) is almost constant, about 5%. The difference between \( reuse24 \) and \( reuse48 \) represents the impact of the channel bandwidth constraint. The impact is zero for up to 40 test pins, 4.38% for 48, 9.11% for 56, and 14.85% for 64 test pins.

As conclusion, although the channel bandwidth constraint can significantly impact the SoC test length, it happens only if the NoC does not provide enough bandwidth. Typically the NoC channel bandwidth is higher than the test bandwidth, thus, in this case the channel bandwidth constraint is not a relevant issue.

7.6. Evaluating NoCs with Different Topologies

This section presents test length results for NoCs with different topologies, demonstrating the adaptability of the test scheduling algorithm and the

---

\(^{4}\)The detailed results for \( reuse24 \) are not presented due to the lack of space.
impact of the neighborhood and clustering constraints on the test length.

The SoC d695 is used for this evaluation where the cores of this SoC are placed in 3x3 mesh, 3x3 torus, two irregular topologies (called irregular1 and irregular2 illustrated in Fig. 10(a,b)), and a fully connected topology (called fconn illustrated in Fig. 10(c)). Both irregular topologies have been randomly generated, however, irregular has three to four links per router while the irregular2 has two to three links per router. All topologies have nine routes. The NoC frequency is two times faster than the test frequency.

Figure 9: Comparing test length results with different NoC channel bandwidth. The error bars represent the average percentage of the worst and best placements.

Figure 10: Topologies evaluated: irregular1 (a), irregular2 (b), and fully connected (c).
Fig. 11(a) compares conv and fconn, excluding any impact of the neighborhood constraint (since fconn is an all-to-all topology) and channel bandwidth constraint (since no part required more than 48 test pins). The results show that the difference between conv and fconn is 1.4% which is attributed to the clustering constraint because it forces a router and its cores to be in the same part and to be tested sequentially. In other words, conv has more flexibility to place the scheduled routers and cores in any part, improving the test length. In addition, it can also be concluded that value of 1.4% can change if the test length of the routers change. For instance, the use of BIST for routers could shorten the test length the impact of clustering constraint.

![Figures](image-url)

(a) conv vs. fconn  
(b) conv vs. all topologies

Figure 11: Comparing the test length results from different NoC topologies.

Fig. 10(b) compares all five topologies against the conventional test scheduling. It can be seen that fconn is the fastest NoC TAM approach, which is expected since a given module can be tested with any other module of the SoC, as conv. The other four topologies reduce the merge candidates since the scheduling algorithm just allows adjacent nodes (i.e. nodes connected by a link) to be merged; affecting the test length.

When comparing fconn with all other topologies the mesh, torus, irreg1, and irreg2 are, respectively, 2.06%, 1.34%, 1.36%, and 1.93% slower than fconn. The result for torus is similar to irreg1 and mesh is similar to irreg2. The reason for this variation is the average number of ports of the routers (the port to the core is not counted). For instance, mesh has 4 routers with
2 ports, 4 routers with 3 ports and 1 router with 4 ports. Torus has 9 routers with 4 ports. *irreg1* has 4 routers with 3 ports and 5 routers with 4 ports (see Fig. 10(a)). *irreg2* has 3 routers with 2 ports and 6 routers with 3 ports (see Fig. 10(b)). It can be concluded that the average number of ports is directly related to the impact of *neighborhood constraint*.

8. Conclusion

This paper presented a *new test environment* used to evaluate both dedicated and NoC TAMs with several SoCs, including a large SoC with 117 cores. Metrics like SoC test length and TAM wire length were evaluated. To the best of our knowledge, this is the first paper to estimate the amount of wiring saved by using NoC as TAM. For instance, we concluded from the experimental results that the TAMs increase the total number of global wires of the chip in 25%; in some cases 58% is required.

The paper also presented a *new test scheduling for NoC TAMs* based on a *new problem formulation* which simplifies the test scheduling by requiring few details about the NoC. Results for five different NoC *topologies* have been presented, demonstrating the adaptability of the proposed test scheduling algorithm. As far as we know, this is the first test scheduler for NoC TAM to support other topologies rather than just mesh.

The proposed test scheduler for NoC TAMs relies on three constraints: channel bandwidth, clustering, and neighborhood constraints, which are individually evaluated in the results. For instance, it has been seen that: the channel bandwidth constraint is not relevant if the NoC bandwidth is bigger than the test bandwidth; the clustering constraint has an impact of 1.4% on test time; and the impact of neighborhood constraint is directly related to the average number of ports of the NoC routers.

Finally, the tools presented in this paper can help a designer to decide whether NoC TAM is the best test approach for a given design. Considering the reduced amount of dedicated test resources required by our approach, the SoC test length is only 3.88% longer than dedicated TAMs.

Acknowledgements

Part of this work was accomplished while Alexandre was holding a doctoral scholarship from the CNPq-PNM (grant number 141993/2002-2). Currently, Alexandre is supported by postdoctoral scholarships from Capes-
PNPD and FAPERGS-ARD, grants number 02388/09-0 and 10/0701-2, respectively. Cristiano is partially supported by FCT (INESC-ID multi-annual funding) through the PIDDAC Program funds. Marcelo is partially supported by CNPq (Projeto Universal), grant number 478200/2008-0. Fernando is partially supported by CNPq scholarship, grant number 301599/2009-2. The authors also acknowledge Luciano C. Ost, Edson I. Moreno, Ney V. Calazans, and Cesar A. M. Marcon for their help reviewing this paper.

References


Alexandre de Morais Amory is currently a Postdoctoral fellow at PUCRS University, Porto Alegre, Brazil. He received bachelor and master degrees in computer science from the PUCRS University, in 2001 and 2003, respectively. In 2007 he received the Ph.D. in computer science from UFRGS University, Porto Alegre, Brazil. His thesis received an honorable mention in the CAPES Thesis Award, in 2008. He did an internship at Philips Research Laboratories, The Netherlands, in 2005. He worked, from 2007 to 2009, at CEITEC as a lead ASIC verification engineer. He has two journal papers, 21 papers in international conferences such as ITC, ETS, VTS. His research interests include design, test, fault-tolerance, and verification of digital systems, especially MPSoCs and NoCs.

Cristiano Lazzari received the master degree in Computer Science from UFRGS University, Brazil, in 2003. He received the Ph.D. degree in Microelectronics from the UFRGS University and the Institut National Polytechnique de Grenoble (INPG), France, in 2007. Currently, Cristiano Lazzari is a researcher in the ALGOS group at INESC-ID, in Lisbon, Portugal. His research interests include developing techniques for design & test of NoCs, and developing algorithms for logic synthesis and technology mapping of multi-valued circuits.

Marcelo Lubaszewski received the Electrical Engineering and M.Sc. degrees from the Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1986 and 1990, respectively. In 1994, he received the Ph.D. degree from the Institut National Polytechnique de Grenoble (INPG), France. In 2001, he joined the Laboratoire d’Informatique, Robotique et Microélectronique de Montpellier in France as an Invited Researcher for 3 months and, in 2004, the Instituto de Microelectrónica de Sevilla (IMSE) in Spain for 1 year. He is currently with UFRGS, where he has been a Professor since 1990. His primary research interests include design and test of mixed-signal, micro-electro-mechanical, core-based and NoC-based systems.
self-checking and fault-tolerant architectures, and computer-aided testing. He has published over 200 papers in international journals and conferences on these topics. Dr. Lubaszewski has served as a Guest Editor of the Journal of Electronic Testing: Theory and Applications, of the Microelectronics Journal, as an Associate Editor of the Design and Test of Computers Magazine and as a lecturer for Latin America in the frame of the IEEE Computer Society Distinguished Visitors Program. He is presently a member of the Editorial Board of the VLSI Journal and the Editor-in-Chief of JICS, the Journal of Integrated Circuits and Systems of the Brazilian Computer and Microelectronics Societies. In the past, he has also served the IEEE CS Test Technology Technical Council as the Chair of the Latin-America Regional.

**Fernando Gehm Moraes** received the Electrical Engineering and M.Sc. degrees from the Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1987 and 1990, respectively. In 1994 he received the Ph.D. degree from the Laboratoire d’Informatique, Robotique et Microélectronique de Montpellier (LIRMM), France. He is currently at PU-CRS, where he has been an Associate Professor from 1996 to 2002, and Professor since 2002. From 1998 to 2000 he joined the LIRMM as an Invited Professor for 3 months each year. He has authored and co-authored 14 peer refereed journal articles in the field of VLSI design, comprising the development of networks on chip and telecommunication circuits. One of these articles, HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip, is cited by more than 200 other papers. He has also authored and co-authored more than 140 conference papers on these topics. His primary research interests include Microelectronics, FPGAs, reconfigurable architectures, NoCs (networks on chip) and SoC (system on chip design).