EFFICIENT TECHNIQUES FOR TESTING NETWORKS ON CHIPS

by

Nancy Abou Eid

Submitted in partial fulfillment of the requirements for the degree of
Master of Computer Science

Lebanese American University
Department of Computer Science and Mathematics
Byblos, 1401 2010 Lebanon
May 2010

THESIS APPROVAL FORM

Student Name: Nancy Abou Eid

Thesis Title: EFFICIENT TECHNIQUES FOR TESTING NETWORKS ON CHIPS

Program: MS in Computer Science

Department: Computer Science and Mathematics

School: Arts and Sciences, Byblos

We the undersigned hereby certify that we have read and recommend to the School of Arts and Sciences, Byblos for acceptance a thesis entitled “EFFICIENT TECHNIQUES FOR TESTING NETWORKS ON CHIPS” by Nancy Abou Eid in partial fulfillment of the requirements for the degree of Master of Computer Science.

Supervisor:

Dr. Haidar Harmanani

Readers:

Dr. Danielle Azar

Dr. Chadi Nour

Dated: May 20, 2010

LEBANESE AMERICAN UNIVERSITY

DATE: May 20, 2010

AUTHOR: Nancy Abou Eid

TITLE: EFFICIENT TECHNIQUES FOR TESTING NETWORKS ON
CHIPS

DEPARTMENT: Computer Science and Mathematics

DEGREE: MS in Computer Science CONVOCATION: May 20 YEAR: 2010

I grant to the Lebanese American University the right to use this work, irrespective of any copyright, for the University's own purpose without cost to the University or its students and employees. I further agree that the University may reproduce and provide single copies of the work to the public for the cost of reproduction.

I attest that I have read and understood the Lebanese American University *Plagiarism Policy*. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.

Signature of Author

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

The author attests that permission has been obtained for the use of any copyrighted material appearing in the thesis (other than brief excerpts requiring only proper acknowledgement in scholarly writing) and that all such use is clearly acknowledged.

Table of Contents

List of Tables	vii
List of Figures	viii
Abstract	x
Acknowledgements	xi
Chapter 1 Introduction	1
1.1 Embedded Cores	1
1.2 Network-on-Chip Related Concepts	2
1.2.1 Topologies	3
1.2.2 Switching Techniques	7
1.2.3 Routing Strategies	12
1.3 Core Test Architecture Concepts	13
1.3.1 Test Pattern Source and Sink	14
1.3.2 Test Access Mechanism	14
1.3.3 Test Wrapper	16
1.4 SoC Testing	17
1.5 NoC Testing	18
Chapter 2 Related Work	22
2.1 NoC Architecture	22
2.2 NoC Wrapper Design	22
2.3 NoC Routing	23
2.4 NoC Power and Area Cost	24
2.5 SoC Testing and TAM Architecture	26
2.6 NoC Testing and TAM Architecture	28

Chapter 3	Problem Description and Thesis Outline	31
3.1	Problem Description and Thesis Outline	31
Chapter 4	NoC Test Scheduling using XY Routing and Simulated Annealing.	34
4.1	Solution Approach	34
4.2	Simulated Annealing Applied to Core Test Scheduling	35
4.2.1	The Initial Configuration	35
4.2.2	The neighborhood Solutions	35
4.2.3	The Cost Function	36
4.3	XY Routing Applied to Test Packets	37
4.3.1	Cores and Routers Map	37
4.3.2	Core Test Time	40
4.3.3	Flit XY Routing	46
4.4	Results	48
Chapter 5	NoC Test Scheduling using XY Routing, Simulated Annealing, and Core Partitioning based on Similar or Close Core Test Time	50
5.1	Problem Formulation	50
5.2	Solution Approach	50
5.3	Core Partitioning based on Close Test Time	50
5.4	Results	53
Chapter 6	NoC Test Scheduling using XY Routing, Simulated Annealing, and Core Partioning based on the Lower Bound	58
6.1	Problem Formulation	58
6.2	Solution Approach	58
6.3	The Lower Bound	58
6.4	Core Partitioning based on the Lower Bound	59
6.5	Results	62

Chapter 7	Conclusion	66
Bibliography		67

List of Tables

1.1	Topologies pros/cons	6
4.1	NoC XY Routing ITC'02 benchmark Results	49
5.1	NoC ITC'02 benchmark Partitions based on Similar or Close Core Test Time	53
5.2	ITC'02 benchmark Results for NoC Partitioning based on Sim- ilar or Close Core Test Time	56
6.1	NoC ITC'02 benchmark Partitions based on the Lower Bound .	62
6.2	ITC'02 benchmark Results for NoC Partitioning based on the Lower Bound	63
6.3	Our Results for the NoC ITC'02 benchmark	65
6.4	Our Results for the NoC ITC'02 benchmark	65

List of Figures

Figure 1.1	Network Topologies	5
Figure 1.2	Switching techniques: (a) Packet Switching; (b) Virtual Cut-Through; (c) Wormhole Routing	10
Figure 1.3	Network Latency for: (a) Packet Switching; (b) Circuit Switching ; (c) Virtual Cut-Through and Wormhole Routing	11
Figure 1.4	(a) Routing without Virtual Channels (b) Routing with Virtual Channels	11
Figure 1.5	Core test Architecture that applies test patterns generated by ATE (a) off-chip. The BIST configuration has source and sink (b) on-chip [1]	14
Figure 1.6	IEEE P1500 Core Test Wrapper [2]	17
Figure 1.7	Flit Structure	19
Figure 3.1	System d695 in NoC Architecture	32
Figure 4.1	2-D representation of a Core	34
Figure 4.2	Example of the first Neighborhood Solution	35
Figure 4.3	Example of the second Neighborhood Solution	36
Figure 4.4	Annealing Test Scheduling Pseudo Code	37
Figure 4.5	NoC Map	38
Figure 4.6	NoC Map snapshot of Core Testing	39
Figure 4.7	Core Map Configuration Pseudo Code	40
Figure 4.8	Core Test Time Stage 1: Core 0 Test Time = t_0 . The flit waiting buffer time is also added.	41
Figure 4.9	Core Test Time Stage 2: Core 0 Test Time = $t_0 + t_1$. The above two steps are repeated until the core to be tested is reached. Once the flits reach the Core to be tested, they are processed and the Core's test time is updated by adding the flit travel time to the actual Core testing time.	42

Figure 4.10	Core Test Time Stage 3: Core 0 Test Time = $t_0 + t_1 + t_2$. Now, if all Core's test patterns are processed and if the Core under test is the destination Core (output Core), then the Core's testing time is done. However, if the output Core is not reached yet, the path from the tested Core to the output core is determined and the flits' travel time is added to the Core's testing time.	43
Figure 4.11	Core Test Time Stage 4: Core 0 Test Time = $t_0 + t_1 + t_2 + t_3$. The flit waiting buffer time is also added.	44
Figure 4.12	Core Test Time Stage 5: Core 0 Test Time = $t_0 + t_1 + t_2 + t_3 + t_4$. Finally, once the routing path is determined, the time, the path, the routers and the bandwidth relative to the core to test are reserved from source to destination. The same procedure, above, is applied to all the cores to be tested using their test patterns.	45
Figure 4.13	Core Test Time Pseudo Code	46
Figure 4.14	XY Routing Example	47
Figure 4.15	XY Routing Path Pseudo Code	48
Figure 4.16	d695 XY Routing Schedule	49
Figure 5.1	Core Partitioning based on Close Test Time Stage 1	51
Figure 5.2	Core Partitioning based on Close Test Time Final Stage	52
Figure 5.3	Close Test Time Partition Pseudo Code	52
Figure 6.1	NoC Lower Bound Pseudo Code	59
Figure 6.2	Core Partitioning based on the Lower Bound Example	60
Figure 6.3	Lower Bound Partition Pseudo Code	61

Abstract

Network-on-Chip (NoC) is a new technology that embeds heterogeneous interconnected cores. NoC's design is based on a selected network topology, a switching technique and a routing strategy in order to allow on chip communication. Its advantages over System-on-Chip (SoC) are that NoC provides modularity, higher performance, better structure, and compatibility with core designs and reuse. Trades-off exist between them.

In this thesis, we tackle the NoC core testing time problem. We use a grid topology, a variable Test Access Mechanism (TAM), a computed optimal flit size based on the bandwidth and buffer size, the standard test wrapper, and XY routing strategy. Our goal is to test all cores, under the above constraints, while minimizing overall NoC test time. For this purpose, we have partitioned the cores into sets using two partitioning techniques. The first is based on grouping together all cores that have similar or close core testing time, and the second is partitioned based on the lower bound computed for each benchmark. In order to evaluate our work, we present experimental results that are compared to each other.

Acknowledgements

This research project would not have been possible without the support of many people. I wish to express my gratitude to my supervisor, Dr. Haidar M. Harmanani who was abundantly helpful and offered invaluable assistance, support and guidance in my work on this thesis and throughout my graduate studies. Deepest gratitude are also due to the members of the committee, Dr. Danielle Azar and Dr. Chadi Nour, for their encouragement. I would also like to convey thanks to the Lebanese American University for providing the financial means and laboratory facilities. And of course, I deeply appreciate my friends' and parents' support throughout my studies, graduate and undergraduate, as well as my entire life.

Chapter 1

Introduction

Nowadays, digital systems are becoming very complex due to the increase in the number of computational logic, and storage blocks that are being integrated in a single chip (or System-on-Chip (SoC)) [3]. Therefore, the custom design of components from scratch is not efficient, and the benefit of reusability is hence not taken advantage of. Methodologies such as SoCs have emerged.

1.1 Embedded Cores

Integrated circuits (ICs) are designed and manufactured in a way that the entire system can be placed on a single die. SoCs are large integrated circuits (ICs) that consist of large, predesigned and pre-verified reusable components, called embedded cores. SoCs have higher performance, lower power consumption, and smaller volume and weight than traditional multi-chips. Furthermore, embedded cores have a short time to market and allow external design expertise to be used. They are designed by two entities, core provider and core user, that could belong to the same company or to two different companies. Therefore, predesigned embedded cores may be produced by different entities at a different time than its use by an SoC [4].

Embedded cores can be soft, hard or firm. The soft cores are categorized as register-transfer level Hardware Description Language (HDL). They leave the synthesis and layout implementation to be done by the user [5]. Soft cores are flexible and process-independent [4]; thus, they can be used by different semiconductor processes [6]. In contrast, hard cores lack flexibility and they are process-dependent. They are designed by the core provider for predictable performance and/or power consumption [4]; they include technology-dependent layout timing information and hence, can directly be used in a system [6]. Because hard cores might have high performance and/or design complexity, the core user receives the hard cores as black boxes and

optimized layout. Microprocessors, phase locked loops and mixed signal blocks are some hard cores examples [5]. As for firm cores, they fall between soft and hard cores [4]; users can implement it in a predefined structured way. After logic synthesis and technology mapping, firm cores are provided as synthesized netlists that can be altered or simulated by the user as needed [5]. Particularly, firm cores are gate-level netlists HDL available for placement and routing; hence, they are more structured than soft cores. [6]. Typically, soft, hard and firm embedded cores present trade-offs and need to communicate with each other in a system.

Earlier SoCs were a bus-based communication methodology that addresses the following design problems: modularity and reuse, design productivity, global wire speed/power optimization, synchronization, and communication error recovery [7]. The shared bus is not sufficiently scalable and hence cannot handle the large amount of communication between the cores. Moreover, as the communication increases, the power increases [8]. Therefore, due to the increased number of embedded cores and system frequencies, the shared bus architecture causes problems related to signal and power integrity [3]. A new paradigm is introduced, Network-on-Chip (NoC), to solve the SoC's bus and congestion problems.

1.2 Network-on-Chip Related Concepts

NoC-based SoC is an on-chip communication methodology that provides modularity, higher performance, better structure, and compatibility with core designs and reuse [9]. NoC consists of interconnected heterogeneous devices that communicate by sending packets via a scalable network [10]. NoC requires switches, routers and communication protocols [11]. Routing strategy role is to find the communication path between non-neighboring source and destination nodes. On the other hand, switching technique is the process of transmitting messages from input buffers to output buffers. Finally, topology is represented by an undirected connected graph consisting of nodes representing the cores and edges representing the links between them [12]. An NoC can be based on the following network topologies: Linear array, ring, star, binary tree, fat tree, 2-D mesh, Torus, 3-D mesh (Cube), Hypercube and completely connected. Moreover, NoC presents trades-off between chosen topology, switching

technique and routing strategy.

1.2.1 Topologies

NoCs' network topology varies based on the systems' needs, module sizes and placements [7]. It affects network latency, throughput, area, fault-tolerance, power consumption, routing strategy design and core mapping to the network nodes [10]. In order to evaluate a chosen topology, it should be measured. Interconnection networks are measured by [13]:

- Network size (N): number of nodes in the network.
- Node degree (d): number of input and output links of a node.
- Network diameter (D): number of connecting arcs along the longest path S; where S is the set of the shortest path between all pairs of nodes in the network.
- Bisection (B): minimum number of links that need to be removed to break the network into two equal halves.
- Arc connectivity (A): minimum number of arcs that have to be removed from the network to cut it into two disconnected networks.
- Cost (C): number of communication links required by the network.

Typically, Interconnection topologies (Table 1.1) are classified based on their dimensions:

- One-dimensional topology is
 1. Linear array.
- Two-dimensional topologies are
 1. ring
 2. star
 3. binary tree

4. fat tree
5. 2-D mesh
6. Torus.

- Three-dimensional topologies are

1. 3-D mesh (Cube)
2. completely connected.

- Hypercubes topologies are

1. Hypercube
2. 2-D
3. 3-D
4. n-D.

A static view of the static interconnection topologies is illustrated in Figure 1.1.

Figure 1.1: Network Topologies

In addition, the advantages and disadvantages of each interconnection topology are shown in table 1.1.

Table 1.1: Topologies pros/cons

	Advantage(s)	Disadvantage(s)
Linear Array	Simplest Used in pipelined algorithms Low cost Low node degree	Worst diameter Very poor bisection width Very poor arc connectivity
Ring	Used at various levels of hierarchical topologies Improves diameter Improves bisection width Improves arc connectivity	
Star	Simple Low cost Improves diameter	Poor bisection width Poor arc connectivity
Binary Tree	Reduced network diameter without sacrificing the node degree Fits with parallel algorithms	Poor bisection width Poor arc connectivity
Fat Tree	Better bisection width Better arc connectivity	Variable node degree High cost
2-D Mesh	Shorter diameter Improves bisection width Improves arc connectivity	Higher node degree Higher cost
2-D Mesh Wraparound/ Torus	Improves diameter Improves bisection width Improves arc connectivity	Higher node degree
3-D Mesh (Cube)	Improves diameter Improves bisection width Improves arc connectivity Not causing too high node degree Not causing too high cost	Difficult realization
Hypercube	Smaller diameter Good bisection width	Higher cost Variable node degree

	Good arc connectivity	Difficult parallel fabrication
Completely connected	Ideal diameter	High cost High node degree

1.2.2 Switching Techniques

Switching is the process of transmitting a message from input to output buffer. It affects the message latency. Packet switching, circuit switching, virtual cut-through and wormhole routing are four different switching techniques.

Packet switching (store-and-forward)

Packet switching consists of dividing a message into packets and sending each packet independently through the network from source node to destination node. Each packet consists of a header and the data. The header contains the routing information that is used by the switching unit in order to forward the packet.

In packet switching, a received packet at an intermediate node is stored in a packet buffer and forwarded to the next neighboring node only if an empty buffer exists at that node. Figure 1.2 (a) depicts this technique.

Packet switching network latency (L) is:

$$L = (P/B) * D \quad (1.1)$$

Where, P is the packet length, B is the channel bandwidth and D (distance) is the number of nodes between source and destination nodes.

Packet switching network latency is illustrated in Figure 1.3 (a).

Packet switching presents two disadvantages. One disadvantage is that latency is proportional to the message path length and the other disadvantage is that a significant amount of memory space is consumed in order to buffer every incoming packet.

Circuit switching

Circuit switching first establishes a path between the source and destinations nodes by sending a signal called probe through the network. The path's channels are reserved exclusively for the message being sent through the path. Then, after the entire message is transmitted through, the path is terminated.

Circuit switching does not need packetizing, hence, no buffering. The message is entirely sent through the path. Therefore, memory space is reduced in comparison with packet switching technique. Moreover, the routing decision is handled by hardware. Therefore, switching time is reduced.

Circuit switching network latency is:

$$L = (P/B) * D + (M/B) \quad (1.2)$$

Where, P is the length of the probe and M is the message length.

Circuit switching network latency is illustrated in Figure 1.3 (b).

If $P \ll M$, then $[(P/B) * D]$ is negligible and hence, the latency becomes independent from the communication distance.

Virtual cut-through

Virtual cut-through divides the message into flits (flow control digits) and forwards these flits in a pipeline manner along available free channels. If a channel is busy, flits are buffered at the node. If the buffer size is large enough, the entire message fits into the node's buffer. Otherwise, the message is buffered at several nodes holding links between them. Figure 1.2 (b) depicts this technique.

Virtual cut-through network latency is:

$$L = (HF/B) * D + (M/B) \quad (1.3)$$

Where, HF is the length of the header flit.

Virtual cut-through network latency is illustrated in Figure 1.3 (c).

If $HF \ll M$, $[(HF/B) * D]$ is negligible and hence, the latency becomes independent from the communication distance.

Wormhole routing

Wormhole routing breaks the packets into flits and transports these flits in a pipelined manner [11]. Its mechanism is similar to that of virtual cut-through except that buffers at each node have the size of one flit. Figure 1.2 (c) depicts this technique.

The network latency is the same as in virtual cut-through and therefore; distance does not affect latency. Wormhole routing network latency is illustrated in Figure 1.3 (c). Moreover, its is useful for broadcast and multicast communication since it can replicate packets. It can send flit copies on many output channels.

Wormhole routing advantage is the introduction of virtual channels that allow the sending of multiple messages by sharing channels. One physical channel is allocated many buffers, each representing a virtual channel. Virtual channels help avoid deadlock and increase network throughput. Figure 1.4 shows the difference between routing with and without virtual channels. Message A is sent to destination P, then message B is sent to destination Q. In the first case (figure 1.4 (b)), message B can reach its destination Q even if message A is blocked. In the second case (figure 1.4 (a)), message B cannot reach its destination because message A is blocked.

Figure 1.2: Switching techniques: (a) Packet Switching; (b) Virtual Cut-Through; (c) Wormhole Routing

Figure 1.3: Network Latency for: (a) Packet Switching; (b) Circuit Switching ; (c) Virtual Cut-Through and Wormhole Routing

Figure 1.4: (a) Routing without Virtual Channels (b) Routing with Virtual Channels

1.2.3 Routing Strategies

Routing determines the path of a message between the source and destination nodes. It affects the network performance and power consumption. In addition, it also presents an area/performance trade-off in case of large designs [10]. Typically, we need to select an efficient and effective routing strategy in order to keep the NoC implementation free from deadlocks and livelocks [9]. There are two categories of routing algorithms: static routing and dynamic routing.

Static routing

In static routing, also called deterministic routing, the path of the message is completely determined by the source and destination nodes; intermediate nodes cannot alter the path.

Static routing can be classified as source routing, where the source node determines the path between the source and destination nodes, or distributed routing, where the intermediate node decides which node the message should be, next, sent to. However, static routing does not make full usage of the network, specially in case of heavy load [9].

There are three deterministic routing schemes:

1. Street-sign routing is classified under source routing. Therefore, the message header contains all the path information.
2. Dimension-ordered routing is classified under distributed routing. It is applied in n-dimensional meshes. XY routing is an example of a dimension-ordered routing applied in 2-dimensional meshes. It is a deadlock free and shortest path routing algorithm, where the packets first travel in the X direction, then in the Y direction to reach the destination node. [14]
3. Table-lookup routing is classified under distributed routing and can be used in any network topology. Each node has a routing table that determines the neighbor node the message should be sent to.

Dynamic routing

In dynamic routing, intermediate nodes can decide which neighboring node the message should be sent to, based on network conditions (as failures, bottlenecks, etc).

Dynamic routing can be classified as being either profitable, only channels that guarantee to get the message closer to destination are selected, or misrouting, where non-profitable channels are selected to lead the message to profitable channels in order to get closer to destination. Profitable channels result in a minimum-length path. They are free from livelock and they make it easier to prove deadlock freedom.

Dynamic routing can be progressive, where messages cannot go backwards on a certain followed path, or back-tracking, where messages can go backwards. In order to prevent same path search, history information is kept in the message header.

If there is no profitable channel at a node, progressive profitable routing waits for a free profitable channel; or a progressive misrouting protocol tries a non-profitable free channel; or backtracking routing steps backwards and starts again at a previous node.

The above protocols can be completely adaptive, where any channel can be chosen, or partially adaptive, when restrictions exist for deadlock freedom in the network.

1.3 Core Test Architecture Concepts

Embedded cores are predesigned, reusable modules integrated into system chips. These embedded cores need to be tested for defects. [4] presents their test architecture which consists of:

1. Test Pattern source and sink
2. Test Access Mechanism
3. Test Wrapper.

The core test architecture is depicted in Figure 1.5.

Figure 1.5: Core test Architecture that applies test patterns generated by ATE (a) off-chip. The BIST configuration has source and sink (b) on-chip [1]

1.3.1 Test Pattern Source and Sink

Test patterns have a source and a sink. In particular, the test pattern source's role is to generate test stimuli, while the test pattern sink's role is to compare resulting test responses to expected ones. There are three ways to implement test pattern sources and sinks. Either on-chip, by using built-in-self-test (BIST), or off-chip, by using Automatic Test Equipment (ATE), or combining the two. Source and sink of the same chip can be implemented differently. In case of on-chip implementation, BIST source and sink require less silicon area, hence shorter TAM, due to their close distance to cores under test. But BIST implementation occupies some area. In practice, only algorithmic patterns can be generated on-chip without excessive silicon area. In addition, advanced BIST implementation provides detailed diagnostic data but at the expense of more silicon area. On the other hand, external ATE systems have low frequency and timing accuracy to test SoCs and require large capital investment to be implemented. Moreover, ATEs require TAM from the IC pins to internal core terminals and vice versa. In particular, cores' tests have high data throughput which need large TAMs and hence, large silicon area is required. [4]

1.3.2 Test Access Mechanism

Test Access Mechanism (TAM) is the architecture that transports test stimuli from the test pattern source to the core under test and, test responses from the core under test to the test pattern sink. TAM is affected by its width and length. The TAM

width corresponds to the TAM transport capacity and the TAM length is the physical distance TAM has to traverse between source and core or core and sink. [4]

TAM can be implemented in different ways while respecting test time, area cost and use case. Having a set of cores and a number of test pins, the number of TAMs, the TAM widths, the core assignment to TAM and the wrapper design have to be defined in order to decide on which TAM design to implement. The number of TAMs and TAM widths are determined by the total number of cores in the circuit, the number of I/O terminals, the number and length of the internal scan chains, and the number of test patterns per core. [15]

TAM architectures can be implemented either by giving a partial TAM width to each core, or by granting the entire TAM width to all cores, or by combining both approaches. In the first case, independent testing is possible, while in the second it is not. Here are three main TAM architectures: Multiplexing, DaisyChain and Distribution. Daisychain and Multiplexing are full TAM access architectures. Multiplexing architecture allows only one wrapper access at a time, preventing parallel test access to wrappers and hence, testing becomes nearly impossible. On the other hand, DaisyChain architecture allows parallel access to wrappers; when wrapper test patterns are exhausted, the wrapper is switched to bypass mode. Finally, Distribution architecture also allows parallel wrapper access but grants partial TAM to each core based on the amount of test data transferred to and from the core. This architecture reduces SoC testing time. However, individual cores are assigned smaller TAMs as compared to the previous two architectures, hence, individual cores have larger testing time. [15] Another architecture, TestRail architecture, was proposed by [15]. It combines the Daisychain and the Distribution architectures. As in Distribution architecture, one or several independent TestRail(s) are placed in parallel on a single SOC. As in the Daisychain architecture, cores are connected to each TestRail entity and can be tested simultaneously or sequentially. Whenever cores run out of test patterns, the wrapper is switched to bypass mode. Finally, the Testrail architecture leads to an SoC test time equal to the maximum of its TestRails testing times.

1.3.3 Test Wrapper

In modular testing, a wrapper serves as the interface between a core and its SoC. It can operate in one of the following modes at a time:

1. Normal operation mode: the core is connected to its host core and the wrapper acts as if it was transparent.
2. Core-internal test mode: the TAM is connected to the core's inputs and outputs in order to send test stimuli and collect test responses.
3. Core-external test mode: the TAM is connected to the interconnect wiring and logic between the cores in such a way that the test stimuli are applied on the output of the core and the response is collected on the input of the next core.

Additionally, a core test wrapper can also operate in detach mode, where a core is separated from its SoC and TAM or bypass mode, in case of TAM exhaustion.

The core test wrapper role is to connect the core terminals to the TAM. It consists of a set of scan chains whose task is to serialize the input of the core. Core functionalities and application determine the number of core terminals, as for the TAM width, it is determined by the amount of silicon space the SoC manufacturer is considering to assign to it and to the bandwidth of the source and sink. The TAM width and the number of core terminals do not always match. Therefore, if the TAM width is smaller than the number of core terminals, the core test wrapper is altered in order to adapt to this mismatch, through serial-to-parallel conversion at the core inputs and parallel-to-serial conversion at the core outputs. [4]

Core test wrapper is standardized by the IEEE P1500 Standard for Embedded Core Test illustrated in Figure 1.6. It has a scalable TAM width and supports both TestBus and TestRail TAM architectures. Furthermore, the IEEE 1500 standard supports all the wrapper test modes discussed above, except the detach mode. [4]

Figure 1.6: IEEE P1500 Core Test Wrapper [2]

1.4 SoC Testing

Manufactured chips can be tested before being placed on a board, while cores can only be tested after being integrated in an SOC. In fact, internal testing consists of the internal Design For Testability (DFT) structure, that is scan chains or test points, and the test patterns associated with the DFT. [6]

As already mentioned, core provider and core user are two different entities. The core user can't design tests that are core specific since he is unaware of the core design produced by the core provider. Therefore, the core provider should supply the core user with the core's DFT structure and test patterns. However, the core provider might be unaware of the core's target application, the test method (BIST, scan, IDDQ, functional) that should be adopted, the type of faults (static, dynamic or parametric), and the desired level of fault coverage (low fault coverage leads to a bad chip and requires low testing time, while high fault coverage leads to a good chip but requires high power consumption, long test time, extra silicon area, and has low performance). Despite all of this, the core provider specifies the requirements of the core's internal test [6].

On the other hand, the core user performs test expansion by translating all the pre-computed tests, received from the core provider at the core terminals, into chip level tests, described at the IC pins [16]. The core user also creates a test access

path between IC pins and the core, with a proper bandwidth based on the core's requirements. Furthermore, the core user has to perform an entire system test by testing the cores as well as the User Defined Logic (UDL) and the interconnect wiring. The test design should be effective, efficient and doable on the used test equipment. Typically, test infrastructure design leads to trade offs between the ease of use, the test quality, the test time, and the silicon area. Another trade off exists, in test scheduling, and takes place among the ability to execute the test without conflicts and without increase in power dissipation, overall test time, and silicon area spent on DFT [4].

1.5 NoC Testing

Networks are preferred over buses because of their high bandwidth and their ability of supporting multiple concurrent communications [17]. Typically, NoC systems consist of cores, wrappers, switches, routers and communication channels, that affect the NoC testing time.

In normal operation of an NoC system, embedded cores, placed on a certain topology, communicate by sending and receiving request and response messages that are split into packets. On the other hand, in testing mode, SoC test data are transferred through a specific TAM. However, NoC test data are divided into test packets and communicate by reusing the available on-chip network as their TAM. The packet size depends on the routing strategy used, the buffer size, etc [18]. Each packet consists of a header that holds the information needed to create the path between the sender and the receiver, a payload that holds the actual data being transmitted, and a trailer. Packets are shorter than messages and are routed individually, hence occupy small number of channels which leads to efficient use of the network [19]. Moreover, packets are further divided into several flits (flow control unit). A flit consists of *Control* and *Data* bits, depicted in Figure 1.7. When used under test, each flit will have extra *Control* bits to identify the core under test and the test mode, while the *Data* bits will hold the test vectors of the core under test [20]. Flits are routed using the Wormhole routing switching technique, where each flit can be directly output after being read at a node. This technique reduces communication latency, which is the delay between message initiation and availability of the data at the destination node.

The communication latency has the following three elements [21]:

1. Start up latency: Time to prepare a packet that will be transmitted or received.
2. Network latency: Difference between time at which the message's head is sent from the source node and the time at which the message's tail is received by the destination node.
3. Blocking latency: Time delay due to network traffic and resources contention.

Figure 1.7: Flit Structure

Another issue concerning NoC-based system, during testing, is the high power consumption issue. Power is consumed by the core, the wrapper, the router and the communication channel. During test, the core provider specifies the core and wrapper power consumptions. In addition, in [12] it is mentioned that BISTed cores reduce testing time. However, the BISTed cores have high power consumption that prevents all cores to have a BIST testing approach and hence, power consumption holds back core testing time reduction.

On the other hand, for NoC core testing to be possible, it needs to perform protocol conversion and buffering, as well as balance the wrapper scan chains. [22].

When a requested output channel is busy, messages are stored into buffers. However, reducing the buffer size, increases the number of flits which leads to an increase in startup communication overheads. A trade-off exists between communication latency and overhead. [21] proposed an equation that computes the optimal buffer size (B) in bytes.

$$B = \sqrt{\frac{M \times \beta}{(D - 1) \times \alpha}} \quad (1.4)$$

Where, M is the message size (bytes), β is the fixed startup communication time (seconds), and α is the communication time proportional to the packet size (seconds/byte).

From the above equation, it can be noticed that the increase in optimum buffer size

is affected by the increase of the message size. In addition, the optimum buffer size decreases as the message path increases.

Moreover, the flit size is usually equal to the width of the NoC channel, but sometimes it happens that each flit's test data bits are larger than the core's wrapper [22]. Therefore, the NoC channels cannot be used entirely by the traditional wrapper design to transmit test data [20]. In fact, new wrapper designs, based on the IEEE std.1500, have been proposed in order to resolve the NoC channel width utilization efficiency problem [20, 23, 22]. [22] developed two heuristics that find the optimal wrapper design that minimizes test time and area overhead. These two heuristics are based on the maximum bandwidth or maximum test time.

Furthermore, reliable NoC core testing results depend on good routers. Hence, routers also need to be tested for effectiveness and efficiency. The same core testing technique can be used for router testing, such that the on-chip network is reused as TAM and all routers have their own wrapper, the IEEE std.1500, so they can be considered as regular circuits under test. A core and a router might either each have a separate wrapper, or a common wrapper that integrates both the router and its associated core. The generated router test response packets can be either routed to the output ports and analyzed, off-chip, by the ATE, or collected, for each router or group of routers, into an MISR (Multiple Input Shift Register) that is analyzed at the end of the testing process [18]. [18] developed a test scheduling algorithm that integrated both core and router testing. Their method showed improvements in test time reduction.

Finally, cores need to be scheduled for testing. Test scheduling is the process of arranging the cores testing order, given a certain TAM, such that the overall NoC testing time is minimized. Minimizing NoC testing time, is minimizing the time needed for all cores' response packets to be available at the system outputs [12]. In order to do so, a core's testing time needs to be measured. [24] presented an equation that computes a core's test time t (clock cycles), given by:

$$t = \{\{max(s_i, s_o) + 1\} \times p + min(s_i, s_o)\} \quad (1.5)$$

Where, s_i is the scan-in length, s_o is the scan-out length and p is the number of test patterns.

Chapter 2

Related Work

2.1 NoC Architecture

NoC adopts the packet-based architecture instead of the SoC's bus-based architecture. The following papers focused on the communication infrastructure in NoC.

Marculescu et al. [25] dealt with network design issues. They worked on having the right elements connected in the right communication pattern in order to achieve the right functionality. Network structure and network behavior, as well as the impact of the application have been discussed.

Ogras et al. [10] presented many NoCs design research problems. They focused on the architectural-level of the problem. The problems are divided into three categories: communication infrastructure synthesis, communication paradigm selection, and application mapping optimization. Each problem's motivation, formulation and open research issues have been discussed. Future work can cover other levels.

Yuan et al. [26] re-examine the cost of reusing the NoC as TAM to transport test data to/from embedded cores. They compare the testing time, area cost, test reliability and test control complexity resulting from using this method to the ones generated by using dedicated bus-based TAM. Based on the designers' test requirements, this analysis allows easier construction of test architectures for NoC-based systems.

2.2 NoC Wrapper Design

A wrapper consists of the interface between a core and the TAM. Designing a wrapper that efficiently connects TAM to core terminals will contribute to enhance core testing time. The following reviews present different wrapper designs in order to serve

on-chip testing.

Huang et al. [23] designed a new "jitter-aware" test wrapper and a new "jitter-transparent" ATE interface. Sometimes, electrical noise may happen and cause test data corruption during transmission; thus leading to test yield loss. However, even in this case, Huang's proposed solution provides reliable modular testing of SoC devices. They also prove that their technique is efficient by applying it to an industrial circuit and providing its results.

LI et al. [20] proposed a new wrapper design with interleaved test scheduling. This method minimizes test time while considering power constraints and without manipulating test frequencies. It also saves design effort for test engineers. Furthermore, in order to simulate test data transfer between cores under test in NoC-Based systems, they presented flit-level models for test flits and sections of NoC channel. In conclusion, their technique is justified by presenting results that show minimized testing time of NoC-based systems under power constraints.

Hussin et al. [22] developed two NoC wrapper designs (Type 1 and Type 2) that complement each other in order to optimize test schedule while minimizing overhead. Previous wrapper design did not take into consideration inefficient bandwidth utilization. Hussin et al. [14] have propose two heuristics to find the optimal wrapper design for a given maximum bandwidth or maximum test application time. The developed wrappers scale well for large circuits and provide small test time overhead for NoC reuse. Moreover, wrapper Type 2 efficiently uses NoC bandwidth without causing any overhead on the test application time.

2.3 NoC Routing

NoC routing is the process of transmitting messages from source node to destination node along a path. The following papers present different algorithm for routing.

Haramanani and Farah [9] presented a method for assigning tasks to nodes in a 2-D mesh, and used simulated annealing to determine the nodes positions on the

mesh. They developed a new efficient routing algorithm, using dynamic routing, that minimizes blocking while increasing bandwidth throughput.

Dally et al. [17] proposed the use of a general-purpose on-chip interconnection network. In their design, system modules communicate with each other by sending packets over the network instead of being connected by routing dedicated wires. The area overhead required to implement an on-chip network is about 6.6%. They introduced on-chip networks' concept, depicted a simple network, and discussed some architecture challenges and networks' design.

Majer et al. [14] presents a solution for the Dynamic Network on Chip (DyNoC) problem. DyNoC consists of a dynamic communication between modules placed dynamically on a reconfigurable device. They developed an adaptive Q-routing and S-XY-routing algorithms that support communication on DyNoC and compared their performance. Communication on the DyNoC is equivalent to routing on meshes with obstacles.

The results show that Q-routing performs well under varying network load while using only local information for its routing decisions. Q-routing drawback is that it has high area cost, it uses a table to store estimated delivery time.

Symons et al. [21] developed a new model of wormhole routing. Using this model, this paper shows how the optimal buffer size is derived and that the communication time using wormhole routing depends on the square root of the network diameter for medium to large messages. Message routing is necessary since a diameter of one cannot interconnect all processors.

2.4 NoC Power and Area Cost

The NoC area consists of the area occupied by the wires and the packet switch logic. As for the power, it is the amount of energy that the NoC consumes. A trade-off exists between area, power and performance constraints. The following reviews try to minimize NoC area and power costs.

Srinivasan et al. [27] developed new Mixed Integer Linear Programming (MILP) formulations for synthesis of custom NoC architectures. The optimization objective of the techniques is power consumption minimization taking into consideration performance constraints. The power consumption of NoC architecture is defined by the physical links (that depend on the length of the link which in turn, is dependent on the SoC layout) and routers.

They use a two-stage approach for solving the custom NoC synthesis problem:

1. Floorplanning that determines the core's and router's locations.
2. Interconnection network generation that generates topology of the NoC and the routes for the traffic traces using the same floorplan from stage 1.

The team runs both stages using optimal MILP formulation; it timed out for many benchmarks. They also run a clustering-based heuristic technique on the second stage to reduce run times of the MILP formulation; it generated better results in acceptable time. The results are analyzed by comparison with mesh-based NoC. As for deadlock, it can be removed by a post-processing step that introduces additional virtual channels at select routers.

Bolotin et al. [7] first presented the analysis of the NoC's generic cost in area and power and interconnect architectures: shared bus, segmented bus and point-to-point interconnect. Analytical expressions for area, power dissipation and operating frequency are derived as well as asymptotic limits of these functions. Advantages of NoC scalability are quantified by the analysis.

The authors, next, explore cost tradeoffs between the number of buffers and the link speed. Quality-of-Service NoC (QNoC) which is based on a grid of wormhole switches, shortest path routing and multiple quality of service (QoS) classes is used as reference architecture.

Total area cost = total area occupied by wire + area occupied by the packet switch logic (buffers, tables etc.).

Power cost = $\sum$ traffic that traverses each wire length and is received by input stages. Two traffic scenarios are considered: short packets, sensitive to queuing delays, and large block-transfers. The results show that network cost minimization is possible

while preserving QoS, by trading off buffers with links in the first scenario but not in the second.

Ivanov and De Micheli [11] presented five articles that focus on issues related to NoC. In the first article Pande et al. reviews problems and solutions related to NoC design, automatic synthesis, and post-manufacturing testing. Design trade-offs and performance optimizations are issues under debate. In the second article, Goossens et al. describe AEthereal NoC which aims at high performance multimedia embedded systems. The network concept is described by illustrating how it can meet quality-of-service specifications for real-time applications or high resource utilization objectives for best-effort service applications. AEthereal NoC can achieve balanced SoC solutions by the evaluation and trade off of programming models, performance, and cost. In the third article, S.-J. Lee et al. focus on the design and evaluation of real ICs. The results show that the star topology is the most effective when used as local network architecture. It is also useful as a basis for global network architecture when combined with low-power link schemes. Trade-offs exists between packet format, size, and corresponding protocols. In the fourth article, Murali et al. worked on error recovery methods in order to guarantee reliability with high-performance SoCs. They analyzed error recovery methods for NoCs. The comparison focused on power consumption, error detection capability, and impact on the network performance with overall objective of meeting reliability and performance specifications. In the last article, Bobda and Ahmadinia worked on DyNoC. They proposed circuit-routing solution based on reconfigurable multiple bus on Chip (RMBoC) and target devices with unlimited reconfiguration capabilities. The first approach might work only for today's reconfigurable FPGA devices and the second one is a more generic 2D dynamic model. More investigations are needed for clearing network regions.

2.5 SoC Testing and TAM Architecture

SoC-based system is a bus-based communication methodology that reuses core designs and core tests. During testing, test data is transported via a TAM. Reused core tests need to be appropriately scheduled in order to minimize testing time. The

following papers present different SoC testing design solutions and some of them propose new TAM architecture.

Marinissen et al. [28] presented in this paper the ITC'02 SOC Test Benchmarks in order to stimulate new SoC testing methods and tools and allow efficient and effective evaluation of the result. They also provided the benchmark format and naming, as well as a set of research problems for which the benchmark can serve as test cases.

Iyengar et al [29] proposed a new Test Access Mechanism (TAM) architecture and a new rectangle packing algorithm for test scheduling. Their algorithm allows test wrapper and TAM co-optimization. They have also developed new heuristics that minimize the idle time on TAM wires which leads to a fast and efficient algorithm for TAM width allocation and test scheduling. Moreover, the test schedule has power constraints and a group of tests are set as preemptable. In this way, power and hardware consumption conflicts are avoided and the schedule is more efficient. They have also identified an effective TAM width by trading off testing time with tester data volume. Experimental results of this test automation have been provided for four benchmark SoCs. However, their method does not take care of power dissipation, precedence, or interconnection test.

Haramanani and Farah [30] developed a new method based on simulated annealing to determine minimum SoC test schedules with wrapper design and TAM optimization. This method can perform SoC test scheduling with and without power and precedence constraints. They have tested their method using the ITC 2002 benchmarks and have found optimum test times in some cases.

Zorian et al. [4] explained the traditional and the core-based test development differences. They have overviewed industrial practices and performed academic research in the field of core design and core test reuse. They have also discussed VSIA and IEEE P1500 contributions and described new challenges.

Goel et al. [15] proposed in their paper a new TAM architecture that combines the

DaisyChain and Distribution architectures. It is called the TestRail architecture. This architecture efficiently tests cores and their interconnections. Two algorithms, applicable to cores having fixed-length scan chains as well as variable-length scan chains, are developed to reduce the overall SoC core test time. This new method is proven to reduce test time by comparing its results to previous ones using three SoC benchmarks.

Bergamaschi et al. [5] outline the following SoC related issues: market drivers and trends, technology and integration aspects, early architecture definition, methodology, hardware and software design and verification techniques.

Gupta et al. [6] describe the core design, the core test and the core use related issues. Since heterogeneous components are integrated into a single chip, while being rarely modified, core design challenges arise. For this purpose, tools and methodologies are developed in order to design, validate and test these systems.

Marinissen et al. [16] presents structured TAM where TESTSHELL wraps reusable IP modules. They used a standardized Test Control Mechanism (TCM) to control the TESTSHELL, and a standardized TESTRAIL to send test data from chip pins to TESTSHELL and vice versa.

2.6 NoC Testing and TAM Architecture

NoC-based systems reduce routing cost by reusing on-chip network as TAM to transmit test data to/from embedded cores. The latter need to be scheduled in a way that minimizes overall NoC testing time. The following reviews tackle NoC testing design while considering new TAM architecture in some cases.

Jin-Ho Ahn and Sungho Kang [3] proposed an Ant Colony Optimization (ACO)-based rectangle packing algorithm for test organization of Network-on-Chip (NoC)-based System-on-Chip (SoCs). The rectangle packing method combined with ACO improves the scheduling results by dynamically choosing the Test-Access-Mechanism

(TAM) widths for cores and changing the testing orders through the actions of an artificial ant, which effectively handles the minimal cost problem. It could improve test parallelism. The power dissipation and variable test clock mode are also considered. Experimental results using ITC'02 benchmark circuits show that the proposed algorithm can efficiently reduce overall test time. Moreover, the computation time of the algorithm is less than a few seconds in most cases.

Cota et al. [12] proposed the Built-In self Tested (BISTed) cores that contribute in test time and test cost reduction of NoC-based systems by reusing on-chip network as TAM. Using BISTed cores improves test parallelization but present higher power consumption during test. Therefore, a trade-off between system configuration and testing time exists.

The team presented a method to define which cores (not all cores) in an NOC-based system should have a BIST testing approach, so that the whole system test time is minimized.

Hosseiniabady et al. [8] developed a new algorithm that reuses on-chip networks for testing NoC switches. The proposed algorithm broadcasts test packets to switches through the on-chip networks and faults are detected by comparing output responses of switches with each other. The algorithm improves on bus-based NoC testing methods in that it does not need external comparison of the output response of the circuit-under-test with the response of a fault free circuit stored on a tester. It also doesn't need on-chip signature analysis, neither a dedicated test-bus to reach test vectors and collect their responses.

Liu et al. [18] developed a new method for test access and test scheduling in NoC-based system. This method consists of progressively reusing the network resources in order to transport test data to routers. Several implementation solutions are presented. The team also shows how application time testing can be reduced by concurrently scheduling router testing and core testing. Based on the ITC'02 SoC benchmarks experimental results, test application time using the proposed method is significantly reduced as compared to the use of serial boundary scan. The method can also help

reduce hardware overhead.

Cota et al. [19] reused the on-chip network as the TAM to the embedded cores in the systems. They developed an algorithm to minimize test time. Then, they evaluated the reuse strategy considering different positions of the cores in the network, power consumption constraints and number of interfaces with the tester.

Based on the ITC'02 SoC test Benchmarks experimental results, the system test time is reduced due to network parallelization, whereas area and pin overhead are strongly minimized.

Chapter 3

Problem Description and Thesis Outline

3.1 Problem Description and Thesis Outline

A Test Access Mechanism supports test patterns. In SoC-based systems, TAMs are used to transport test data between input/output pins of the SoC and the cores under test. While in NoC-based systems, on-chip network is reused as TAM to transport test data to/from embedded cores [26]. In the thesis, we will use NoC as a TAM along with the standard test wrapper as our first approach. We will allocate a TAM width to the NoC cores under test based on their Pareto-optimal points (first point beyond which adding TAM bits would keep testing time stable). From [29], we can see that the higher the Pareto-optimal point, the less core test time is needed. But a limited amount of TAM bits are available. First, we will start by allocating 32 bits. In order to reduce testing time, we will partition the cores into sets, and for each set, apply TAM bits through an input core chosen randomly from the Map generated by the Annealing Mapping algorithm in [9]. Hence, partitions will be tested in parallel. However, testing cores in parallel might not reduce a core's test time but can reduce the overall NoC test time. After the tests are performed, the results will be taken from each partition output core also randomly chosen from the Map. Moreover, we will use the Annealing test scheduling algorithm developed in [30] to choose which cores to test in parallel, in what order and which Pareto-optimal point to use for each core. Test scheduling is the process of arranging the cores testing order, given a certain TAM, such that the overall NoC testing time is minimized.

On the other hand, each test pattern will be transported by one flit in order not to change the router's configuration. A flit is the smallest unit that forms a packet. Wormhole routing is the switching technique that breaks the packets into flits and transports these flits in a pipelined manner. In [21], it is shown that as the buffer

size is reduced (latency is reduced), the number of flits is increased (overheads increased). Here, a latency/overhead trade off exists. Furthermore, [21] provides an equation (1.4), that we will use, to compute the optimal flit size based on the bandwidth and the buffer size. Based on the analysis in [21], it is shown that, using the wormhole routing and fewer communication links, the mesh topology is dominant for large message passing systems. Therefore, we may use either the 2D mesh or 2D mesh wraparound (Torus) topologies. We will start by applying our technique to a 3x4 2D mesh representing the NoC architecture of d695 benchmark depicted in Figure 3.1.

Figure 3.1: System d695 in NoC Architecture

Furthermore, the packets will reach the cores using XY routing strategy. Routing determines the path of a message between the source and destination nodes. XY routing is a dimension-ordered routing applied in 2-dimensional meshes. It is a deadlock free and shortest path routing algorithm, where the packets first travel in the X direction, then in the Y direction to reach the destination node.

In particular, we will partition the cores into sets using two partitioning techniques and apply all of the above strategies, explained furthermore in Chapter 4, to each

generated set from both partitions. The first partition, presented in Chapter 5, is based on grouping together all cores that have similar or close core testing time, while the second partition, presented in Chapter 6, is based on the lower bound computed for each benchmark. Finally, in order to evaluate our work, we present, in Chapter 7, experimental results that are compared to previous ones.

Chapter 4

NoC Test Scheduling using XY Routing and Simulated Annealing.

The problems addressed in this chapter are the NoC test packet routing problem along with core test scheduling. It is not a new problem and has been shown to be NP-hard.

4.1 Solution Approach

We represent cores using a 2-D layout as shown in Figure 4.1, where the X-axis represents the core test time, and the Y-axis represents the TAM bits width allocated to that core. In fact, the cores are placed in a 2D mesh and the test packets are routed based on the XY routing strategy. However, conflict of resources and packet collision can occur. The goal is to schedule these cores in a 2D architecture such that the test time along the X-axis is the smallest possible, while the predefined and fixed TAM bits width is not exceeded, and precedence and concurrency constraints are satisfied. Moreover, each core is assigned a start time and a finish time.

Figure 4.1: 2-D representation of a Core

4.2 Simulated Annealing Applied to Core Test Scheduling

A core is characterized by the following: a start time, a finish time, a TAM width that is allocated to it, its test packets source and destination core(s), and its individual test time. Input and Output cores are randomly chosen to respectively generate and collect each core's test packets.

4.2.1 The Initial Configuration

The initial configuration is achieved by scheduling the cores in the decreasing order of their test patterns' size, and associating each core with its test packets' source and destination core. In other words, the core with the highest test patterns is scheduled first, the core with the second highest test patterns is scheduled next, and so on.

4.2.2 The neighborhood Solutions

The first neighborhood solution consists of randomly choosing two cores and switching their positions in the cores' list. Hence, changing their execution order, while satisfying the precedence and concurrency constraints.

Figure 4.2: Example of the first Neighborhood Solution

Figure 4.2 illustrates the first neighborhood solution. Each vector represents the list of cores where each cell corresponds to one core along with its corresponding source and destination core. Cores 1 and 3 were randomly chosen, and their positions swapped.

After altering the core list, the second neighborhood solution consists of randomly choosing two cores and the smallest core coordinate between these two is placed right after the other randomly chosen core.

Figure 4.3: Example of the second Neighborhood Solution

Figure 4.3 illustrates the second neighborhood solution, where Cores 0 and 2 are randomly chosen and Core 0 coordinate, smaller than 2, is placed right after Core 2. Precedence constraints restrict cores from starting execution before the ones that should precede them. It should be noted that concurrency constraints prevent cores from having interleaving test times.

4.2.3 The Cost Function

The cost function is designed to minimize the overall NoC test time. The cost function is chosen to be the finish time of the latest scheduled core.

```

NoC_Testing
{
  Read benchmark
  Read Core Map
  Randomly choose 1 input cores
  Randomly choose 1 output cores
  Annealing_Test_Scheduling
}

Annealing_Test_Scheduling
{
   $\beta$  = iteration multiplier
   $S_0$  = initial solution
  maxTime  $\leftarrow$  Total allowed time for the annealing process
   $\alpha$   $\leftarrow$  Cooling rate
  M  $\leftarrow$  time until next parameter update
   $T_0$  = initial temperature
  currentS  $\leftarrow S_0$ 
  currentCost  $\leftarrow$  cost(currentS)
  bestCost  $\leftarrow \infty$ 
  T  $\leftarrow T_0$ 
  Time  $\leftarrow 0$ 

  While (time  $\leq$  maxTime)
  {
    While (M > 1)
    {
      Change  $\leftarrow$  false
      Done = 0
      While ((change = false) AND (done < 1000))
      {
        newS = neighborhood(currentS)
        Check for TAM, precedence and concurrency constraints
        If (no violation)
        {
          Draw the Map (cores and routers)
          newS = StartPath(newS)
          Change = true
          Done++;
        }
        newCost  $\leftarrow$  cost(newS)
        deltaCost  $\leftarrow$  newCost - currentCost
        if (delta < 0)
        {
          currentS  $\leftarrow$  newS
          currentCost  $\leftarrow$  newCost
          if (new_Cost < best_Cost)
          {
            bestS  $\leftarrow$  currentS
            bestCost  $\leftarrow$  newCost
          }
        }
        Else
        {
          If (random <  $e^{\text{delta\_cost}/T}$ )
          {
            currentS  $\leftarrow$  newS
            currentCost  $\leftarrow$  newCost
          }
        }
        M  $\leftarrow$  M-1
      }
      time  $\leftarrow$  time + M
      T  $\leftarrow \alpha * T$ 
      M  $\leftarrow \beta * M$ 
    }
    Draw Map (cores and routers)
    bestS = StartPath(bestS)
  }
}

```

Figure 4.4: Annealing Test Scheduling Pseudo Code

4.3 XY Routing Applied to Test Packets

In an NoC, cores are placed in a 2-D mesh and hence, test flits can take many different paths to reach the core under test. Each core is assigned a router and a buffer holding the test flits that reach their destination core and are waiting to be processed. The following details how the test flits are routed to their destination cores while taking collision and latency into consideration.

4.3.1 Cores and Routers Map

As illustrated in Figure 4.5, cores are placed in the 2-D mesh based on their order in the list generated by the two neighborhood solutions. Routers are then assigned to each core.

Figure 4.5: NoC Map

After allocating a TAM width to the set of cores, a time slot is assigned to each core. For every core to test, the Pareto optimal point, specifying the bandwidth, is calculated based on the available TAM, the core's test patterns are divided into packets, then into flits, the buffer size is computed, and the core's start time is determined. The finish time of the tested core is determined when the response flits reach the output core.

In Figure 4.6, the test wrappers of all the cores along the path are in their normal mode if the flits passing through are not to be processed at these cores. Once the flits reach the core to be tested, they either get processed right away, or in case the core is busy, the test flits are stored in the buffer assigned to that core. However, the buffer size could cause overhead if it is too large and not fully used. It could also cause communication latency if it is too small. Therefore, the buffer size is optimized and calculated based on the following equation.

$$B = \sqrt{\frac{M \times \beta}{(D - 1) \times \alpha}} \quad (4.1)$$

In particular, if the buffer of the core under test is full, then the flits will be stored in the buffer of the previous core in the path and so on. Finally, after processing the test flits, the response flits reach their destination core (output core) and the path is determined. In addition, the test time, the TAM, and the flits from source,

destination and core tested are verified. If no violation was found, then the time, the bandwidth and the flits are assigned to the tested core. Meanwhile, the remaining TAM bits, that exceeded the core's Pareto- ptimal point, are assigned to other cores that are executed in parallel and hence, contributing to the minimization of the overall NoC testing time.

Figure 4.6: NoC Map snapshot of Core Testing

```

StartPath
{
    Packets = 0
    Assign time slot to each core with a certain TAM
    For each coreToTest
    {
        Bandwidth = paretoOptimal point of coreToTest
        allTestPatterns = testPatterns * patternLength
        while (allTestPatterns > 0)
        {
            allTestpatterns = allTestpatterns - bandwidth
            packets = packets + 1
        }
        Compute BufferSize
        startTime
        Process_Path()
        Verify testTime, TAM and packets of source, destination and
        coreToTest
        If (no violation)
        {
            Allocate time, bandwidth, buffer and packets to coreToTest
        }
    }
}

```

Figure 4.7: Core Map Configuration Pseudo Code

4.3.2 Core Test Time

The core's test time is calculated from the time the core's test patterns start their route from the input core, going through the core to be tested, to the time they reach the output core. The following are four cores placed in a 2-D mesh, having as source core 1 and as destination core 3. We assume that the testing of Cores has already started and the state of the NoC cores is represented in Figure 4.8. As an example, core 0 test time is to be determined. The core's test time is first set to be 0, and the path to be taken by the flits from the source core (input core) to the core to be tested is determined. For each flit, the travel time to the next core is computed and added to the core test time.

Figure 4.8: Core Test Time Stage 1: Core 0 Test Time = t_0 . The flit waiting buffer time is also added.

Figure 4.9: Core Test Time Stage 2: Core 0 Test Time = $t_0 + t_1$. The above two steps are repeated until the core to be tested is reached. Once the flits reach the Core to be tested, they are processed and the Core's test time is updated by adding the flit travel time to the actual Core testing time.

Figure 4.10: Core Test Time Stage 3: Core 0 Test Time = $t_0 + t_1 + t_2$. Now, if all Core's test patterns are processed and if the Core under test is the destination Core (output Core), then the Core's testing time is done. However, if the output Core is not reached yet, the path from the tested Core to the output core is determined and the flits' travel time is added to the Core's testing time.

Figure 4.11: Core Test Time Stage 4: Core 0 Test Time = $t_0 + t_1 + t_2 + t_3$. The flit waiting buffer time is also added.

Figure 4.12: Core Test Time Stage 5: Core 0 Test Time = $t_0 + t_1 + t_2 + t_3 + t_4$. Finally, once the routing path is determined, the time, the path, the routers and the bandwidth relative to the core to test are reserved from source to destination. The same procedure, above, is applied to all the cores to be tested using their test patterns.


```

Process_Path
{
    time = 0
    stackF = PathXY (from source to coreToTest)
    stackS = null
    if (stackF is not null AND stackFSize > 0)
    {
        stackD = stackFSize - 1
        lastPosition = position (stackD)
        time = time of lastPosition + sleep + coreTestTime

        if (coreToTest is not the destination)
        {
            stackS = PathXY (from coreToTest to destination)
            if (stackS is not null AND stackSSize > 0)
            {
                stackD = stackSSize - 1
                lastPosition = position (stackD)
                time = time of lastPosition + sleep + packets
                Reserve time, path, routers and bandwidth for coreToTest from
                source to destination
            }
        }
    }
}

```

Figure 4.13: Core Test Time Pseudo Code

4.3.3 Flit XY Routing

This section deals with XY routing applied to a Core test flit along its path from Input Core to Output Core. In most cases the path is divided into two paths. The first path starting at the Input Core and ending at the Core to be tested, and the second path starting at the tested Core and ending at the output core. Both paths have source and destination Cores. For each path, the source and destination cores are located on the Map and routers are assigned to each core accordingly. Then, the path is determined based on the relative positions of the source and destination cores from each other. In fact, using XY routing, the flit is first routed in the X direction than in the Y direction. In other words, the X-axis is first explored. For example, if the x coordinate of the destination core is smaller than the X coordinate of the source core, the flit is routed to the core on its left, toward its destination. On the other hand, if the x coordinate of the destination core is greater than the X coordinate of the source core, the flit is routed to the core on its right, toward its destination. If the X coordinate of both source and destination Cores are equal, or if the X coordinate

is exhausted, then the Y-axis is explored. In this case, if the Y coordinate of the destination core is smaller than the Y coordinate of the source core, the flit is routed up to the next core, toward its destination. Otherwise, the flit is routed down to the next core. The procedure above is used to generate the paths from source to destination. The steps are repeated at every core until the path destination core is found and no core is visited twice. Figure 4.14 illustrates an example of the above XY Routing procedure on a 4x4 2D mesh.

Figure 4.14: XY Routing Example

Table 4.1: NoC XY Routing ITC'02 benchmark Results

NoC	Input Core	Output Core	Minimal Buffer Size	Test Time
d281	2	7	221	214517
d695	8	3	1	36707
f2126	3	1	128	335344
g1023	7	8	2	44783
p22810	28	16	1	420785
p34392	19	11	1	1355722
p93791	29	25	1	1188574
q12710	3	2	1553	2923365

NoC d695 from the benchmark above consists of ten cores which resulting schedule is shown in figure 4.16.

Figure 4.16: d695 XY Routing Schedule

Chapter 5

NoC Test Scheduling using XY Routing, Simulated Annealing, and Core Partitioning based on Similar or Close Core Test Time

5.1 Problem Formulation

Allocated a TAM width and the standard test wrapper, an NoC core's test patterns can originate from different cores and take different routes to reach their destination, possibly leading to conflict of resources and packet collision. Therefore, concurrency and precedence constraints need to be satisfied at all times. In other words, some cores should not be tested before other cores, and two test patterns belonging to the same core should not be scheduled concurrently. In this chapter, we will present a Core Test Scheduling technique that contributes in minimizing the overall NoC testing time while satisfying the above constraints.

5.2 Solution Approach

Due to NoC's architecture, packets can directly reach any core in the grid regardless of their source. We are using a 2D mesh topology. Therefore, any cores in the grid can now be partitioned into sets and scheduled in parallel to other cores.

5.3 Core Partitioning based on Close Test Time

Core Partitioning based on close test time consists of grouping the cores with similar or close testing time into the same partition.

First, we start by computing the test time of each core, tested individually.

Figure 5.1: Core Partitioning based on Close Test Time Stage 1

Second, we find the core with the maximum testing time and the core with the minimum testing time and we compute the mean. Then, the mean is divided by the number of cores and the resulting value is added to the maximum testing time defining the threshold for each group.

Based on the Example shown in Figure 5.1, Core 6 has the maximum test time 9869, and Core 1 has the minimum test time 12.

$$\text{mean} = \text{max} - \text{min} = 9869 - 12 = 9857$$

$$\text{variant} = \text{mean} / \text{number of cores} = 9857 / 10 = 985.7$$

$$\text{threshold} = \text{max} + \text{variant} = 9869 + 985.7 = 10854.7$$

Finally, we partition the cores such that each group of cores has a total test time less than or equal to the threshold value computed in the previous step. Each partition now holds cores of similar or considerably close testing time. However, cores belonging to different partitions have relatively larger testing time difference.

Figure 5.2 illustrates the core partitioning method explained above, using a 3x4 2D mesh. The core's with similar colors belong to the same partition.

Figure 5.2: Core Partitioning based on Close Test Time Final Stage

```

Partition_Close_TestTime
{
    Max = Find max CoreTestTime
    Min = Find Min CoretestTime
    threshold = (max - min) / coreListSize

    limit = max + threshold

    While (threshold <= limit)
    {
        Create newSet
        For every core in the List
        {
            If (CoreTestTime < threshold AND core != visited)
            {
                Add core to Set
                Core = visited
            }
        }
        Add Set to PartitionSets
        threshold = threshold + threshold
    }
    return PartitionSets
}

```

Figure 5.3: Close Test Time Partition Pseudo Code

For each partition, one input core and one output core are randomly chosen, a TAM Width, the standard test wrapper, a buffer and a router are assigned to every core. After this configuration, XY Routing and Annealing Test Scheduling are applied to every partition leading to parallel execution and reduction of overall NoC testing time.

5.4 Results

The NoC's of the ITC'02 benchmark suite are partitioned and tested using the above algorithm. The simulated annealing parameters used, were determined experimentally, and are the following: $\alpha = 0.99$, $\beta = 100$, $T_0 = 4000$, $M = 5$. Table 5.1 presents NoC's ITC'02 benchmark suite partitions along with their corresponding cores. In addition, Table 5.2 reports each partition's results, for each NoC, along with the input and output cores, the minimal buffer size, and the NoC test time of the best schedule for a maximum TAM width of 32 bits.

Table 5.1: NoC ITC'02 benchmark Partitions based on Similar or Close Core Test Time

NoC	Partition Number	Core(s)
d281	1	1,4,6,8
	2	2,3,5
	3	7
d695	1	1,2,9
	2	3
	3	7,10
	4	8
	5	4
	6	5
	7	6
f2126	1	3,4
	2	2

	3	1
g1023	1	10,11,12
	2	5,6,7,8,13
	3	3
	4	9,14
	5	1
	6	2
	7	4
p22810	1	3,4,6,8,13,15,16,17,19, 24,27,28
	2	10,22,23
	3	7,11,12,14,20
	4	9
	5	2,25
	6	5
	7	18
	8	26
	9	21
	10	1
p34392	1	3,4,6,7,8,9,11,12,13,14, 15,16,17
	2	5,19
	3	1
	4	10
	5	2
	6	18
p93791	1	2,3,4,7,8,9,15,16,18,21, 22,24,25,26,28,30,31
	2	10
	3	11
	4	5,32
	5	19

	6	17,29
	7	12
	8	13,14,23
	9	1
	10	27
	11	20
	12	6
<hr/>		
q12710	1	1
	2	3,4
	3	2
<hr/> <hr/>		

Table 5.2: ITC'02 benchmark Results for NoC Partitioning based on Similar or Close Core Test Time

NoC	Partition				Test Time
	Partition Number	Input Core	Output Core	Minimal Buffer Size	
d281	1	6	8	42	13981
	2	3	5	716	43563
	3	7	7	0	94253
d695	1	2	9	9	1648
	2	3	3	0	2510
	3	7	10	667	6533
	4	8	8	0	4608
	5	4	4	0	5832
	6	5	5	0	6209
	7	6	6	0	9872
f2126	1	4	3	1.0E9	71428
	2	2	2	0	135362
	3	1	1	0	335337
g1023	1	12	10	36	1134
	2	6	13	29	3304
	3	3	3	0	3134
	4	9	14	181	7895
	5	1	1	0	5942
	6	2	2	0	6377
	7	4	4	0	14797
p22810	1	24	19	1	13802
	2	23	22	38	7746
	3	12	7	50	20446
	4	9	9	0	21650
	5	2	25	915	56395
	6	5	5	0	43647

	7	18	18	0	44507
	8	26	26	0	72984
	9	21	21	0	87144
	10	1	1	0	102968
<hr/>					
p34392	1	16	11	4	27206
	2	19	5	1364	51060
	3	1	1	0	170279
	4	10	10	0	236602
	5	2	2	0	294067
	6	18	18	0	544582
<hr/>					
p93791	1	7	18	9	11552
	2	10	10	0	9322
	3	11	11	0	15606
	4	5	32	653	40408
	5	19	19	0	41569
	6	29	17	2166	100853
	7	12	12	0	73306
	8	13	14	1396	254646
	9	1	1	0	120132
	10	27	27	0	124714
	11	20	20	0	140531
	12	6	6	0	227981
<hr/>					
q12710	1	1	1	0	829118
	2	4	3	3122	2923362
	3	2	2	0	2222352
<hr/> <hr/>					

Chapter 6

NoC Test Scheduling using XY Routing, Simulated Annealing, and Core Partitioning based on the Lower Bound

6.1 Problem Formulation

This chapter deals with NoC test time optimization using XY routing, a 2D mesh topology, and TAM partitioning among all cores belonging to the same set while satisfying concurrency and precedence constraints. Being installed in a 2D mesh topology, NoC cores can be grouped into sets and the test patterns can reach any core from many different pathes. However, the problem is to minimize the overall NoC testing time.

6.2 Solution Approach

The solution to this problem, in this section, consists of partitioning the cores based on NoC's lower bound. In other words, the sum of the cores' test time belonging to the same set is less than or equal to the lower bound.

6.3 The Lower Bound

From [31], the Lower Bound LB is computed using Equation 6.1. The lower bound equation consists of finding the maximum value between two parameters. The first parameter holds the value of the Core's longer testing time, while the second parameter holds the value generated by adding all cores' testing time and dividing the result by the given NoC TAM Bandwidth. Since all cores' test times and TAM bandwidth are known, it is possible to compute the Lower Bound.

$$LB = \max\{\max_{\forall i}(t_i), \lceil \frac{\sum_{i=1}^n t_i}{N_{TAM}} \rceil\} \quad (6.1)$$

where, t_i is the test time calculated for every Core using Equation 1.5, and N_{TAM}

is the TAM bandwidth available to the NoC.

```

Lower_Bound
{
    cTime = 0
    For every core in the List
    {
        cTime = cTime + ti
        firstParameter = Find Max ti
    }
    secondParameter = ceil ( cTime / TAM)

    LB = max (firstParameter, secondParameter )
    return LB
}

```

Figure 6.1: NoC Lower Bound Pseudo Code

6.4 Core Partitioning based on the Lower Bound

Core Partitioning based on the lower bound consists of grouping into the same partition the Cores who's overall testing time is less than or equal to the lower bound. At first, we compute the lower bound of the NoC using the procedure explained in the previous section, we add a variation number to the lower bound and we create the first empty set. Second, we compute the test time of each core and, sequentially, add each core to the created set while always checking the overall set testing time. At the addition of a core, if the set's test time is found to be greater than the lower bound, then this set is closed, another empty set is created, and the core causing the inequality in the first set is added to the second one. This procedure is followed until all cores belong to a partition. Finally, each partition holds cores of overall testing time within the NoC lower bound. Figure 6.2 represents the Core partitions generated by the Lower Bound Partitioning. The same 3x4 2D mesh, used in the example of the previous Partitioning technique, is used here but with the Lower Bound of 9969.

Figure 6.2: Core Partitioning based on the Lower Bound Example

```

Lower_Bound_Partition
{
    LB = Lower_Bound ()
    LB = LB + delta

    While (coreList != empty)
    {
        Sum = 0
        Create new Set
        For every core in the List
        {
            sum = sum + CoreTestTime
            If (sum <= LB AND core != visited)
            {
                Add core to Set
                Core = visited
            }
            Else
            {
                sum = sum - CoreTestTime
            }
        }
        Add Set to PartitionSets
    }
    return PartitionSets
}

```

Figure 6.3: Lower Bound Partition Pseudo Code

As in the previous partitioning technique, for each partition, one input core and one output core are randomly chosen, a TAM Width, the standard test wrapper, a buffer, and a router are assigned to every core. After this configuration, XY Routing and Annealing Test Scheduling are applied to every partition leading to parallel execution and reduction of overall NoC testing Time.

6.5 Results

Applying the algorithm to the NoCs of the ITC'02 benchmark suite gave us results presented in table 6.1 and table 6.2. Table 6.1 presents NoC's ITC'02 benchmark suite partitions along with their corresponding lower bound and cores. As for Table 6.2, each partition's results, for each NoC, are reported along with the input and output cores, the minimal buffer size, and the NoC test time of the best schedule for a maximum TAM width of 32 bits. The simulated annealing parameters used are the following: $\alpha = 0.99$, $\beta = 100$, $T_0 = 4000$, $M = 5$, and were determined experimentally.

Table 6.1: NoC ITC'02 benchmark Partitions based on the Lower Bound

NoC	Lower Bound	Partition Number	Core(s)
d281	94250	1	1,2,3,4,5,6,8
		2	7
d695	9869	1	1,2,3,4,9
		2	5,7
		3	7,10
		4	6
		5	8,10
f2126	335334	1	1
		2	2,3,4
g1023	14794	1	1,2,5,10,11,12
		2	3,6,7,8,9,13
		3	4
		4	14
p22810	102965	1	1
		2	2,3,4,5,6,7,8,13,16,27
		3	9,10,11,12,14,15,17,18,19
		4	20,21,22
		5	23,24,25,28

		6	26
p34392	544579	1	1,2,3,4,5,6,7,8,9,11,12,13, 14,15,16,17
		2	10,19
		3	18
p93791	227978	1	1,2,3,4,5,7,8,9,10,11,15,16, 18,19,21,22,24,25,26,28,30,31
		2	6
		3	12,13,17
		4	14,20
		5	23,27
		6	29,32
q12710	2222349	1	1
		2	2
		3	3
		4	4

Table 6.2: ITC'02 benchmark Results for NoC Partitioning based on the Lower Bound

Partition					
NoC	Partition Number	Input Core	Output Core	Minimal Buffer Size	Test Time
d281	1	2	5	93	54240
	2	7	7	0	94253
d695	1	9	1	1	13782
	2	5	7	371	12213
	3	6	6	0	9872
	4	8	10	412	9105
f2126	1	1	1	0	335337
	2	3	4	1041	265479

g1023	1	5	10	6	14742
	2	6	13	23	7840
	3	4	4	0	14797
	4	14	14	0	5127
p22810	1	1	1	0	102968
	2	3	5	1	101076
	3	14	15	4	97460
	4	21	22	81	145549
	5	28	23	15	67620
	6	26	26	0	72984
p34392	1	8	6	1	776040
	2	10	19	1.0E9	440413
	3	18	18	0	544582
p93791	1	25	31	1	276988
	2	6	6	0	227981
	3	17	13	1788	259251
	4	20	14	1609	240749
	5	27	23	1671	231648
	6	32	29	1203	64370
q12710	1	1	1	0	829118
	2	2	2	0	2222352
	3	3	3	0	1588754
	4	4	4	0	1588754

The overall NoCs' Test Time generated from the above methods, for each NoC of the ITC02 benchmark suite, are reported in Table 6.3. The columns *No Partitions*, *Partition [1]* and *Partition [2]*, hold, respectively, the overall NoC Test Time results from chapters 4 and 5, and the above method.

Finally, it can be shown, from the table below, that, in most of the cases, *Partition [1]* outperformed the other two methods.

Table 6.3: Our Results for the NoC ITC'02 benchmark

NoC	No Partition	P1 Clusters	P1 Test Time	P2 Clusters	P2 Test Time
d281	214517	3	94253	2	94253
d695	36707	7	9872	4	13782
f2126	335344	3	335337	2	335337
g1023	44783	7	14797	4	14797
p22810	420785	10	102968	6	145549
p34392	1355722	6	544582	3	776040
p93791	1188574	12	254646	6	276988
q12710	2923365	3	2923362	4	2222352

The following are the results from Cota et.al [19], compared to our results.

Table 6.4: Our Results for the NoC ITC'02 benchmark

NoC	Nb. of CLusters	Test Time
d281	1268	9024
d695	1762	17334
f2126	1924	565261
g1023	4698	41764
p22810	50026	371814
p34392	132644	764570
p93791	46058	449422
q12710	9224	6230935

Chapter 7

Conclusion

This Thesis tackled the NoC core testing time problem using two partitioning methods. The first suggested solution was based on grouping together all cores that have similar or close core testing time and applying an XY routing strategy along with simulated annealing. The generated results were very promising. The second solution was based on partitioning the cores based on the lower bound of each NoC benchmark and applying an XY routing strategy along with simulated annealing. The results reported for this method show that core partitioning improves on the overall NoC Testing Time. In most cases, our results outperform Cota et al's [19] results. We also notice that, the higher the number of clusters, the better the test time. Future improvements could be considered. For instance, Tabu Search, Genetic Algorithm or a hybrid solution could be applied to the dynamic core test scheduling. This would lead to an improved core partitioning which in its turn would generate better NoC test time. Finally, adding thermal constraints to the NoC would improve test efficiency.

Bibliography

- [1] Y. Zorian, E. J. Marinissen, and S. Dey, "Testing Embedded-Core-Based System Chips," in *Proceedings of the 1998 IEEE International Test Conference*, p. 130, 1999.
- [2] A. sehgal, S. K. Goel, E. J. Marinissen, and K. Chakrabarty, "IEEE P1500-Compliant Test Wrapper Design for Hierarchical Cores," in *ITC International Test Conference*, pp. 1203–1212, 2004.
- [3] J.-H. Anh and S. Kang, "NoC-Based SoC Test Scheduling Using Ant Colony Optimization," *ETRI*, vol. 30, no. 1, pp. 129–140, 2008.
- [4] E. J. Marinissen and Y. Zorian, "Challenges in Testing Core-Based System ICs," *IEEE Communications Magazine*, vol. 37, no. 6, pp. 104–109, 1999.
- [5] R. A. Bergamaschi and J. Cohn, "The A to Z of SoCs," in *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design*, pp. 790–798, 2002.
- [6] R. K. Gupta and Y. Zorian, "Introducing Core-Based System Design," *IEEE Design and Test of Computers*, vol. 14, no. 4, pp. 15–25, 1997.
- [7] E. bolotin, I. Cidon, ran Ginosar, and A. Kolodny, "Cost Considerations in Network on Chip," *VLSI*, vol. 38, no. 1, pp. 19–42, 2004.
- [8] M. Hosseinabady, A. Banaiyan, M. N. Bojnordi, and Z. Navabi, "A Concurrent Testing Method for NoC Switches," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1171–1176, 2006.
- [9] H. M. Harmanani and R. Farah, "A Method for Efficient Mapping and Reliable Routing for NoC Architectures with Minimum Bandwidth and Area," in *Proceedings TAISA Conference*, pp. 29–32, 2008.
- [10] U. Ogras, J. Hu, and R. Marculescu, "Key Reserach Problems in NoC Design: A Holistic Perspective," in *Proceedings of the 3rd International Conference on Hardware/Software Codesign and System Synthesis*, pp. 69–74, IEEE/ACM/IFIP, 2005.
- [11] A. Ivanov and G. D. Micheli, "Guest Editors' Introduction: The Network-on-Chip Paradigm in Practice and Research," *IEEE Design & Test of Computers*, vol. 22, no. 5, pp. 399–403, 2005.
- [12] É. Cota, L. Carro, F. Wagner, and M. Lubaszewski, "BISTed Cores and Test Time Minimization in NOC-Based Systems," in *Proceedings International Test Conference*, pp. 612–621, 2003.

- [13] D. Sima, T. Fountain, and P. Kacsuk, *Advanced Computer Architectures: A Design Space Approach*. Addison-Wesley, 1997.
- [14] M. majer, C. Bobda, A. Ahmadinia, and J. Teich, "Packet Routing in Dynamically Changing Networks on Chip," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium Conference*, pp. 154b–154b, 2005.
- [15] S. K. Goel and E. J. Marinissen, "Cluster-Based Test Architecture Design for System-on-Chip," in *Proceedings of the 20th IEEE VLSI Test Symposium*, p. 259, 2002.
- [16] E. J. Marinissen, R. G. J. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, and C. Wouters, "A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores," in *Proceedings of the 1998 IEEE International Test Conference*, pp. 284–293, 1998.
- [17] W. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," in *Proceedings Design Automation Conference*, pp. 684–689, 2001.
- [18] C. Liu, Z. Link, and D. Pradhan, "Reuse-Based Test Access and Integrated Test Scheduling for Network-on-Chip," in *Proceedings of the conference on Design, Automation and Test in Europe*, p. 6, 2006.
- [19] É. Cota, L. Carro, and M. Lubaszewski, "Reusing an On-Chip Network for the Test of Core-Based Systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 9, no. 4, pp. 471–499, 2004.
- [20] J. LI, Q. XU, Y. HU, and X. LI, "Channel Width Utilization Improvement in Testing NoC-Based Systems for Test Time Reduction," in *DELTA*, pp. 26–31, 2008.
- [21] A. Symons and V. L. Narasimhan, "Calculating Optimal Flit Size and Upper Limit on the Performance of Wormhole Routing," in *Proceedings of the Conference on Algorithms and Architectures for Parallel Processing*, pp. 59–72, 1997.
- [22] F. A. Hussin, T. Yoneda, and H. Fujiwara, "Optimization of NoC Wrapper Design Under Bandwidth and Test Time Constraints," in *Proceedings of the 12th IEEE European Test Symposium*, pp. 35–42, 2007.
- [23] L. Huang, F. Yuan, and Q. Xu, "On Reliable Modular Testing with Vulnerable Test Access Mechanisms," in *Proceedings of the 45th annual conference on Design automation*, pp. 834–839, 2008.
- [24] S. K. Goel and E. J. Marinissen, "TAM Architectures and Their Implication on Test Application Time," in *In Proceedings of IEEE International Workshop on Test Embedded Core-Based Systems*, pp. 3.3–1–10, 2001.
- [25] R. Marculescu, J. Rabaey, and alberto Sangiovanni-Vincentelli in *Proceedings of the conference on Design, Automation and Test in Europe*, pp. 1–3, 2006.

- [26] F. Yuan, L. Huang, and Q. Xu, "Re-Examining the Use of Network-on-Chip as Test Access Mechanism," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 808–811, 2008.
- [27] K. Srinivasan, K. S. Chatha, and G. Konjevod, "Linear Programming-Based Techniques for Synthesis of Network-on-Chip Architectures," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 14, no. 4, pp. 407–420, 2006.
- [28] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, "A Set of Benchmarks for Modular Testing of SoCs," in *Proceedings of the International Test Conference*, pp. 519–528, 2002.
- [29] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "Test Access Mechanism Optimization, Test Scheduling, and Tester Data Volume Reduction for System-on-Chip," *IEEE Transactions on Computers*, vol. 52, no. 12, pp. 1619–1632, 2003.
- [30] H. M. Harmanani and R. Farah, "Integrating Wrapper Design, TAM Assignment, and Test Scheduling for SoC Test Optimization," in *Joint 6th International IEEE Northeast Workshop on Circuits and Systems and TAISA Conference. NEWCAS-TAISA*, 2008.
- [31] E. Larsson and H. Fujiwara, "System-on-Chip test scheduling with Reconfigurable Core Wrappers," *IEEE Transactions Very Large Scale Integration Systems*, vol. 14, no. 3, pp. 305–309, 2006.