Testing Network-on-Chip Communication Fabrics

Cristian Grecu, *Student Member, IEEE*, André Ivanov, *Fellow, IEEE*,
Resve Saleh, *Fellow, IEEE*, and Partha P. Pande, *Member, IEEE*

***Abstract*—Network-on-chip (NoC) communication fabrics will be increasingly used in many large multicore system-on-chip designs in the near future. A relevant challenge that arises from this trend is that the test costs associated with NoC infrastructures may account for a significant part of the total test budget. In this paper, we present a novel methodology for testing such NoC architectures. The proposed methodology offers a tradeoff between test time and on-chip self-test resources. The fault models used are specific to deep submicrometer technologies and account for crosstalk effects due to interwire coupling. The novelty of our approach lies in the progressive reuse of the NoC infrastructure to transport test data to the components under test in a recursive manner. It exploits the inherent parallelism of the data transport mechanism to reduce the test time and, implicitly, the test cost. We also describe a suitable test-scheduling approach. In this manner, the test methodology developed in this paper is able to reduce the test time significantly as compared to previously proposed solutions, offering speedup factors ranging from 2× to 34× for the NoCs considered for experimental evaluation.**

***Index Terms*—Interconnect infrastructure, multicast test, network-on-chip (NoC), test optimization, unicast test.**

I. Introduction

SYSTEM-ON-CHIP (SoC) design methodologies are currently undergoing revolutionary changes, driven by the emergence of SoC platforms supporting large sets of embedded processing cores. These platforms may contain a set of heterogeneous components with irregular block sizes and/or homogeneous components with regular block sizes. The resulting platforms are collectively referred to as multiprocessor SoC (MP-SoC) designs [1]. Such MP-SoCs implies the seamless integration of numerous intellectual-property (IP) blocks performing different functions and exchanging data through a dedicated on-chip communication infrastructure. A key requirement of these platforms, whether irregular or regular, is a structured interconnect architecture. The network-on-chip (NoC) architecture is a leading candidate for this purpose [2].

NoC architectures were proposed as a holistic solution for a set of challenges faced by designers of large multicore SoCs. In general, two types of NoC architectures have been identified, as shown in Fig. 1: (a) regular interconnection structures derived from parallel computing and (b) irregular custom-built NoC fabrics. The infrastructure for NoC includes switches, interswitch wires, interface blocks to connect to the cores, and a protocol for data transmission [3]. There is a significant number of active research efforts to bring NoCs into mainstream use [36].

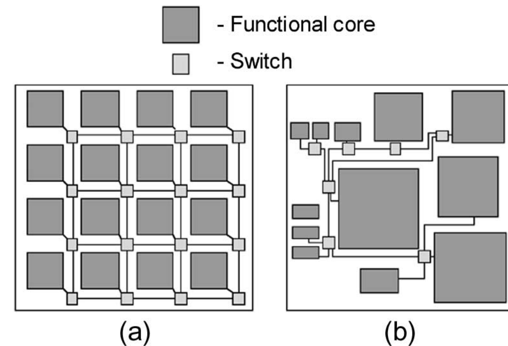


Fig. 1. (a) Regular NoC. (b) Irregular NoC.

Any new design methodology will only be widely adopted if it is complemented by efficient test mechanisms. In the case of NoC-based chips, two main aspects have to be addressed with respect to their test procedures: how to test the NoC communication fabric and how to test the functional cores (processing, memory, and other modules). Since the inception of SoC designs, the research community has targeted principally the testing of the IP cores, giving little emphasis to the testing of their communication infrastructures. The main concern for SoC test was the design of efficient test access mechanisms (TAMs) for delivering the test data to the individual cores under constraints such as test time, test power, and temperature. Among the different TAMs, TestRail [4] was one of the first to address core-based tests of SoCs. Recently, a number of different research groups suggested the reuse of the communication infrastructure as a TAM [5], [6], [35]. Vermeulen *et al.* [7] assumed the NoC fabric as fault-free and subsequently used it to transport test data to the functional blocks; however, for large systems, this assumption can be unrealistic, considering the complexity of the design and communication protocols. Nahvi and Ivanov [8] proposed a dedicated TAM based on an on-chip network, where network-oriented mechanisms were used to deliver test data to the functional cores of the SoC.

This paper complements previous approaches by developing the test strategy for the interconnect infrastructure itself. The test strategies of NoC-based interconnect infrastructures must address two problems [3]: 1) testing of the switch blocks and 2) testing of the interswitch wire segments. We test both switches and interswitch links and integrate the tests in a streamlined fashion. Two novel techniques characterize our solution. First, we reuse the already tested NoC components

Manuscript received November 13, 2006; revised March 25, 2007. This work was supported in part by the NSERC, by Micronet, by PMC-Sierra, by Gennum, and by CMC. This paper was recommended by Associate Editor A. Raghunathan.

C. Grecu, A. Ivanov, and R. Saleh are with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC V6T 1Z4, Canada (e-mail: grecuc@ece.ubc.ca).

P. P. Pande is with the School of Engineering and Computer Science, Washington State University, Pullman, WA 99164 USA.

Digital Object Identifier 10.1109/TCAD.2007.907263

0278-0070/$25.00 © 2007 IEEE

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:20:40 UTC from IEEE Xplore. Restrictions apply.

to transport the test data toward the components under test (CUTs) in a recursive manner. Second, we make use of the inherent parallelism of the NoC structures to propagate the test data simultaneously to multiple NoC elements under test. We provide test-scheduling algorithms that guarantee a minimal test time for arbitrary NoC topologies.

The remainder of this paper is organized as follows. We describe suitable fault models for interconnect and switches in Section II. Test-data organization is described in Section III, and their transportation mechanisms are elaborated in Section IV. A new scheme for test scheduling is proposed in Section V. Results are presented in Section VI. Section VII outlines the conclusions and directions for future work.

II. Fault Models for NoC Infrastructure Test

When developing a test methodology for NoC fabrics, we need to start from a set of models that can realistically represent the faults specific to the nature of NoC as a data-transport mechanism. As stated previously, a NoC infrastructure is built from two basic types of components: switches and interswitch links. For each type of component, we must construct test patterns that exercise its characteristic faults. Next, we describe the set of faults that were considered in this paper for the NoC switches and links.

A. Fault Models for NoC Interswitch Links

In [9], Cuviello *et al.* proposed a novel fault model for the global interconnects of DSM SoCs that accounts for crosstalk effects between a set of aggressor lines and a victim line. This fault model is referred to as maximum aggressor fault (MAF), and it occurs when the signal transition on a single interconnect line (called the victim line) is affected through crosstalk by transitions on all the other interconnect lines (called the aggressors) due to the presence of the crosstalk effect. In this model, all the aggressor lines switch in the same direction simultaneously.

The MAF model is an abstract representation of the set of all defects that can lead to one of the six crosstalk errors: rising/falling delay, positive/negative glitch, and rising/falling speedup. The possible errors corresponding to the MAF fault model are presented in Fig. 2 for a link consisting of three wires. The signals on lines Y_1 and Y_3 act as aggressors, while Y_2 is the victim line. The aggressors act collectively to produce a delay, glitch, or speedup on the victim.

This abstraction covers a wide range of defects including design errors, design-rule violations, process variations, and physical defects. For a link consisting of N wires, the MAF model assumes the worst-case situation with one victim line and $(N-1)$ aggressors. For links consisting of a large number of wires, considering all such variations is prohibitive from a test-coverage point of view [29].

The transitions needed to sensitize the MAF faults can be easily derived from Fig. 2 based on the waveform transitions indicated. For an interswitch link consisting of N wires, a total of $6N$ faults need to be tested, requiring $6N$ two-vector tests. These $6N$ MAF faults cover all the possible process variations and physical defects that can cause any crosstalk effect on any of the N interconnects. They also cover more traditional faults such as stuck-at, stuck-open, and bridging faults.

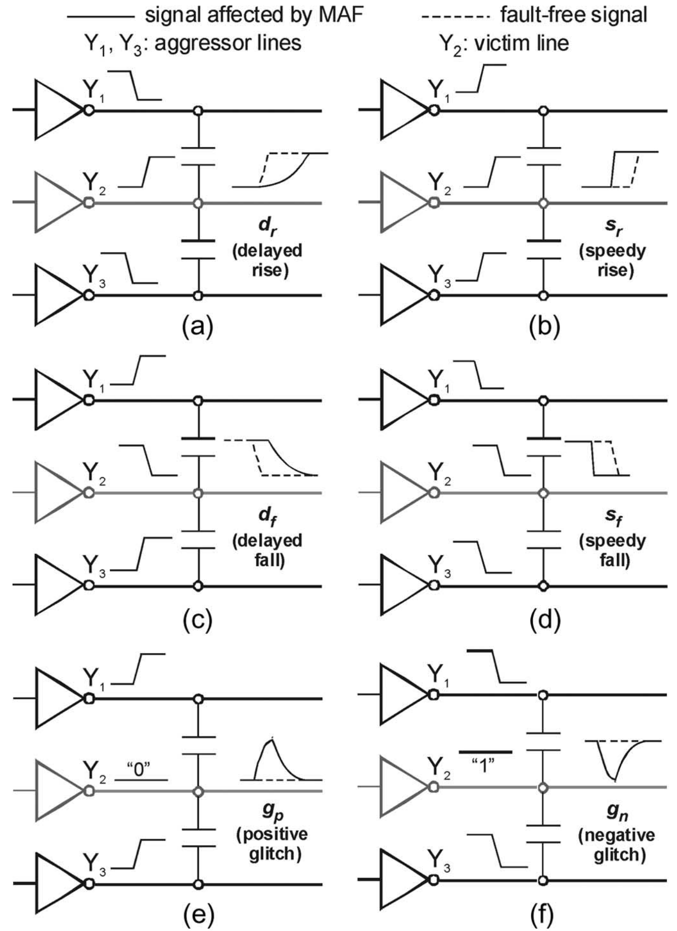


Fig. 2. MAF crosstalk errors (Y_2—victim wire; Y_1, Y_3—aggressor wires).

B. Fault Models for First-In–First-Out (FIFO) Buffers in NoC Switches

NoC switches generally consist of a combinational block in charge of functions such as arbitration, routing, error control, and FIFO memory blocks that serve as communication buffers [10], [18]. Fig. 3(a) shows the generic architecture of a NoC switch. As information arrives at each of the ports, it is stored in FIFO buffers and then routed to the target destination by the routing logic block (RLB).

The FIFO communication buffers for NoC fabrics can be implemented as register banks [11] or dedicated SRAM arrays [12]. In both cases, functional test is preferable due to its reduced time duration, good coverage, and simplicity.

The block diagram of a NoC FIFO is shown in Fig. 3(b). From a test point of view, the NoC-specific FIFOs fall under the category of restricted two-port memories. Due to the unidirectional nature of the NoC communication links, they have one write-only port and one read-only port and are referred to as (wo-ro)2P memories. Under these restrictions, the FIFO function can be divided in three ways: the memory-cells array, the addressing mechanism, and the FIFO-specific functionality [13].

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:20:40 UTC from IEEE Xplore. Restrictions apply.

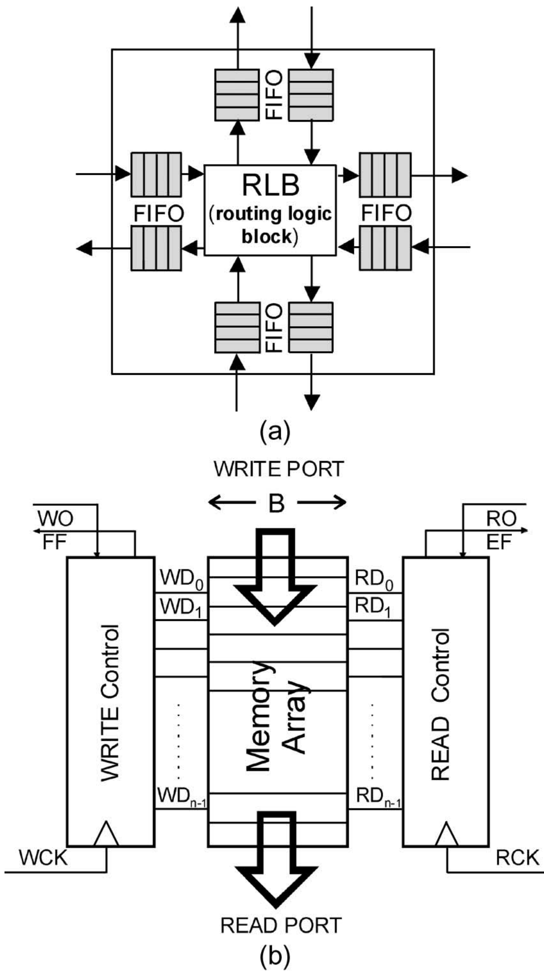


Fig. 3. (a) Four-port NoC switch—generic architecture. (b) Dual-port NoC FIFO.

Memory-array faults can be stuck-at, transition, data retention, or bridging faults [29]. Addressing faults on the RD/WD lines are also of importance as they may prevent cells from being read/written. In addition, functionality faults on the empty and full flags (EF and FF, respectively) are included in our fault models [13].

III. Test-Data Organization

A key feature that differentiates a NoC from other on-chip communication is the transmission of data in the form of packets [14]. In our approach, the raw test data, obtained based on the fault models and assumptions outlined in the previous section, are organized into test packets that are subsequently directed toward the different components of the NoC under test. Test data can be organized into packets by adding routing and control information to the test patterns generated for each NoC component. The routing information is similar to that of functional data packets and identifies the particular NoC component toward which the test packet is directed. The control information consists of fields that identify the packet type (e.g., test packet) and type of the NoC CUT (interswitch link, switch combinational block, FIFO). At the beginning and at the end of each test packet, dedicated fields signal the start and stop moments of the test sequence, respectively.

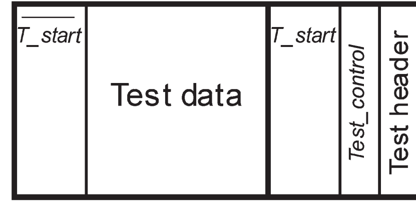


Fig. 4. Test-packet structure.

The test for the faults presented in Section II was developed based on [13]. The generic structure of a test packet is shown in Fig. 4. The test-header field contains routing information and packet-identification information, which denote the type of data being carried (test data). The second field (test control) contains the type of test data, i.e., interconnect, FIFO, or RLB test data. The test-data field is bordered by corresponding flags marking its boundaries.

A. Testing NoC Switches

For test purposes, scan-based testing is adopted. We generate test patterns for the RLB and FIFO blocks separately so that the test vectors can be optimized for each type of block. The test of the logic part of the switch (the RLB block) is performed while this part is isolated from the rest of the switch.

Assuming that the FIFOs are B bits wide and have n locations, the test uses B-bit patterns. As an example, consider the detection of a bridging fault, i.e., a short between the bitlines b_i and b_j $(i \neq j)$ that can eventually yield an AND or OR behavior. In order to detect such faults, four specific test patterns are used: $0101\ldots$, $1010\ldots$, $0000\ldots$, and $1111\ldots$, denoted by G_1, $\overline{G}_1$, G_2, and $\overline{G}_2$, respectively. Specifically, to test the dual-port coupling faults, the following sequence is used:

$$w\left\{\Uparrow_1^{n-1}(wr)\right\}r$$

for each of the four test patterns above. The first write operation sets the read/write pointers to FIFO cells zero and one, respectively; the next $(n-1)$ simultaneous read/write operations sensitize the coupling faults between adjacent cells, and the last read operation empties the FIFO and prepares it for the next test pattern.

B. Testing NoC Links

According to the MAF-fault model, each possible MAF on a victim line of a link requires a two-vector test sequence to be sensitized. The test sequence exhibits some useful properties which allow for a compact and efficient design of the MAF test packets.

1) For each test vector, the logic values on the aggressor lines are the opposite of that on the victim line.
2) After having applied the exhaustive set of test sequences for a particular victim line, the test sequence of the adjacent victim line can be obtained by shifting (rotating) the test data by exactly 1 b.

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:20:40 UTC from IEEE Xplore. Restrictions apply.

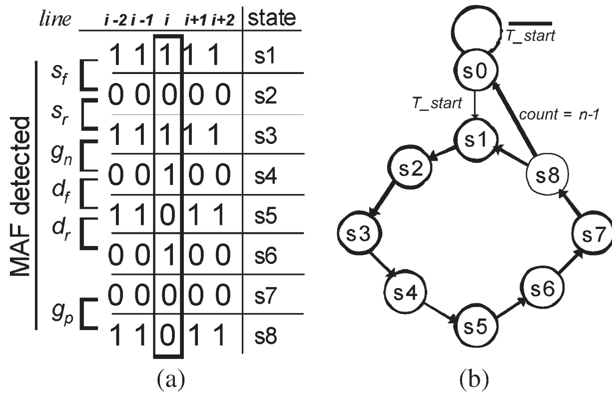


Fig. 5. (a) Test sequence for wire i. (b) Conceptual state machine for MAF patterns generation.

The transition from one test vector to another can be concatenated such that the number of test vectors needed to sensitize the MAF faults can be reduced from 12 vectors per wire to 8, as shown in Fig. 5.

The test-data packets are designed based on properties 1) and 2) above, by generating the logical values corresponding to the MAF tests in eight distinct states s1–s8. In an s1-to-s8 cycle, the state machine produces eight vectors. During each cycle, one line is tested and is assigned the victim logic values, while the rest of the lines get the aggressor values. The selection of the victim wire is achieved through the victim-line counter field that controls the test hardware such that, for the first eight test cycles, the first wire of the link is the victim. During the second set of eight test cycles, the second wire is the victim, and so on. After each eight-vector sequence, the test patterns shift by 1-b position, and an identical eight-vector sequence is applied with a new corresponding wire acting as the victim. This procedure repeats until all the lines of the link are completed.

According to the specific design style of the interswitch links, only a subset of the MAF crosstalk effects may be of interest. In these cases, the test packets that deliver the MAF tests will only possess the appropriate subset of patterns.

IV. Test-Data Transport

A system-wide test implementation has to satisfy the specific requirements of the NoC fabric and exploit its highly parallel and distributed nature for an efficient realization. In fact, it is advantageous to combine the testing of the NoC interswitch links with that of the other NoC components (i.e., the switch blocks) in order to reduce the total silicon-area overhead. The high degree of parallelism of typical NoCs allows simultaneous test of multiple components. However, special hardware may be required to implement parallel-testing features.

In this section, we present the NoC modes of operation and a minimal set of features that the NoC building blocks must possess for packet-based test-data transport. Each NoC switch is assigned a binary address so that the test packets can be directed to particular switches. In the case of direct-connected networks, this address is identical to the address of the IP core connected to the respective switch. In the case of indirect networks (such as butterfly-fat-tree (BFT) [24] and

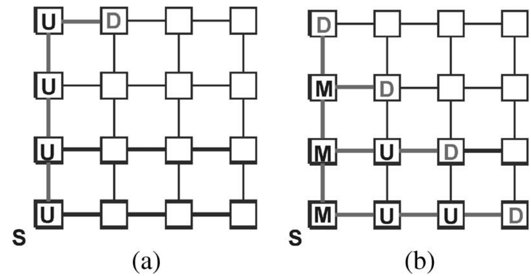


Fig. 6. (a) Unicast data transport in a NoC. (b) Multicast data transport in a NoC (S—source; D—destination; U—switches in unicast mode; M—switches in multicast mode).

other hierarchical architectures), not all switches are connected to IP cores, so switches must be assigned specific addresses in order to be targeted by their corresponding test packets. Considering the degree of concurrence of the packets being transported through the NoC, we can distinguish two cases.

Unicast mode: The packets have a single destination. This is the most common situation, and it is representative for the normal operation of an on-chip communication fabric, such as processor cores executing read/write operations from/into memory cores or microengines transferring data in a pipeline [15]. As shown in Fig. 6(a), packets arriving at a switch input port are decoded and directed to a unique output port, according to the routing information stored in the header of the packet (for simplicity, functional cores are not shown).

Multicast mode: The packets have multiple destinations. This mode is useful for the management and reconfiguration of functional cores of the NoC, when identical packets carrying setup and/or configuration information must be transported to the processing elements [16]. Packets with multicast routing information are decoded at the switch-input ports and then replicated identically at the switch outputs indicated by the multicast decoder. The multicast packets can reach their destinations in a more efficient and faster manner than in the case when repeated unicast is employed to send identical data to multiple destinations. Fig. 6(b) shows a multicast transport instance, where the data are injected at the switch source (S), replicated, and retransmitted by the intermediate switches in both multicast and unicast modes and received by multiple-destination switches (D). The multicast mode is particularly useful for test-data transport purposes, when identical blocks need to be tested as fast as possible.

One possible way to multicast is simply to unicast multiple times, but this implies a very high latency. The all-destination encoding is another simple scheme in which all destination addresses are carried by the header. This encoding scheme has two advantages. First, the same routing hardware used for unicast messages can be used for multidestination messages. Second, the message header can be processed on-the-fly as address flits arrive. The main problem with this scheme is that, as the number of switch blocks in the system increases, the header length increases accordingly and, thereby, results in significant overhead in terms of both hardware and time necessary for address decoding.

A form of header encoding that accomplishes multicast to arbitrary destination sets in a single communication phase

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:20:40 UTC from IEEE Xplore. Restrictions apply.

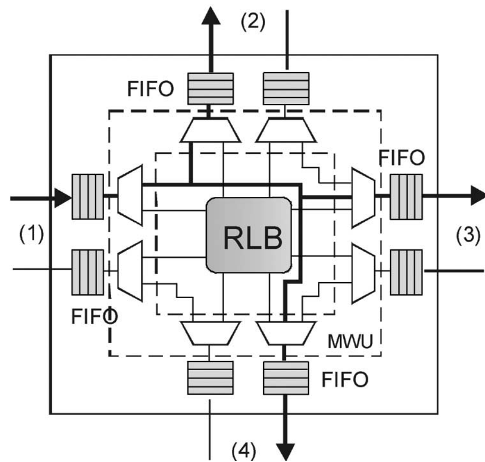


Fig. 7. Four-port NoC switch with MWU for test-data transport.

and also limits the size of the header is known as bit-string encoding [18]. The encoding consists of N bits, where N is the number of switch blocks, with a “1” bit in the ith position, indicating that switch i is a multicast destination. To decode a bit-string-encoded header, a switch must possess knowledge of the switches reachable through each of its output ports [18].

Several NoC platforms developed by research groups in industry and academia feature the multicast capability for functional operation [16], [17]. In these cases, no modification of the NoC switches hardware or addressing protocols is required to perform multicast-test-data transport.

If the NoC does not possess multicast capability, this can be implemented in a simplified version that only services the test packets and is transparent for the normal operation mode. As shown in Fig. 7, we modified the generic NoC switch structure presented in Fig. 3(a) by adding a multicast wrapper unit (MWU), whose functionality is explained as follows. It contains additional demultiplexers and multiplexers relative to the generic switch. The MWU monitors the type of incoming packets and recognizes the packets that carry test data. An additional field in the header of the test packets identifies that they are intended for multicast distribution.

For NoCs supporting multicast for functional data transport, the routing/arbitration logic block (RLB) is responsible for identifying the multicast packets, processing the multicast-control information, and directing them to the corresponding output ports of the switch [10]. The multicast routing blocks can be relatively complex and hardware-intensive.

In our design for multicast-test-data transport, the RLB of the switch is completely bypassed by the MWU and does not interfere with the multicast-test-data flow, as shown in Fig. 7. The hardware implementation of the MWU is greatly simplified by the fact that the test scheduling is done offline, i.e., the path and injection time of each test packet is computed prior to performing the test operation. Therefore, for each NoC switch, the subset of input and output ports that will be involved in multicast-test-data transport is known *a priori*, and we can restrict the implementation of this feature to these specific subsets. For instance, in the multicast step shown in Fig. 6(b), only three switches must possess the multicast feature. By exploring all the necessary multicast steps to reach all destinations, we can identify the switches and ports that are involved in the multicast transport and, subsequently, implement the MWU only for the required switches/ports.

The header of a multidestination message must carry the destination-node addresses [17]. To route a multidestination message, a switch must be equipped with a method for determining the output ports to which a multicast message must be simultaneously forwarded. The multidestination packet header encodes information that allows the switch to determine the output ports toward which the packet must be directed.

When designing multicast hardware and protocols with limited purpose, such as test-data transport, a set of simplifying assumptions can be made in order to reduce the complexity of the multicast mechanism. This set of assumptions can be summarized as follows.

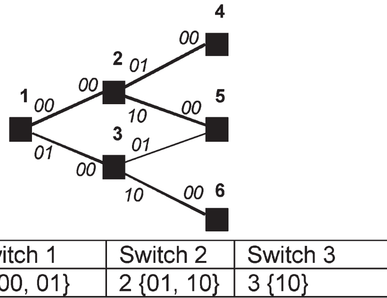
1) The test-data traffic is fully deterministic.
2) Test traffic is scheduled offline, prior to test application.
3) For each test packet, the multicast route can be determined exactly at all times (i.e., routing of test packets is static).
4) For each switch, the set of input/output ports involved in multicast-test-data transport is known and may be a subset of all input/output ports of the switch (i.e., for each switch, only a subset of I/O ports may be required to support multicast).

These assumptions help in reducing the hardware complexity of multicast mechanism by implementing the required hardware only for those switch ports that must support multicast. For instance, in the example of Fig. 7, if the multicast feature must be implemented exclusively from input port (1) to output ports (2), (3), and (4), then only one demultiplexer and three multiplexers are required. A detailed methodology for test scheduling is presented in Section V. The set of I/O ports of interest can be extracted accurately, knowing the final scheduling of the test-data packets, and then, those ports can be connected to the MWU block, as indicated in Figs. 6(b) and 7. Various options for implementing multicast in NoC switches were presented in [16]–[18]; therefore, we omit the details regarding physical implementation here. Instead, we describe how the multicast routes can be encoded in the header of the test packets and how the multicast route can be decoded at each switch by the MWU.

First, we assign binary addresses to each switch of the NoC. Then, for each switch, we assign an index to each of its ports, e.g., if a switch has four ports, they will be indexed (in binary representation) from 00 to 11. We then construct the multicast route as an enumeration of switch addresses, each of them followed by the corresponding set of output ports indexes. A simple example to illustrate how the multicast-test address is built is presented in Fig. 8.

Consequently, with the assumptions stated previously in this section, the MWU must simply decode the multicast routing data and place copies of the incoming packet at the output ports found in the port list of the current switch. Since the test data are fully deterministic and scheduled offline, the test

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:20:40 UTC from IEEE Xplore. Restrictions apply.



DA list	Switch 1	Switch 2	Switch 3
4, 5, 6	1 {00, 01}	2 {01, 10}	3 {10}

Fig. 8. Multicast route for test packets.

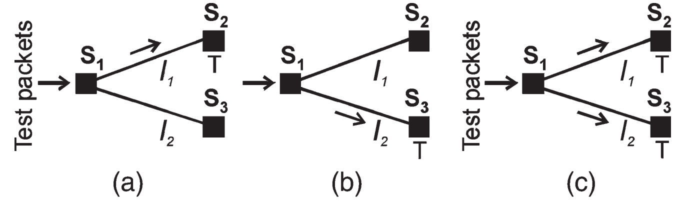


Fig. 9. (a) and (b) Unicast-test transport. (c) Multicast-test transport.

packets can be ordered such that the situation where two (or more) incoming packets compete for the same output port of a switch can be avoided. Therefore, no arbitration mechanism is required for multicast-test packets. In addition, by using this simple addressing mode, no routing tables or complex routing hardware is required.

The lack of input/output arbitration for the multicast-test data has a positive impact on the transport latency of the packets. Our multicast implementation has lower transport latency than the functional multicast, since the only task performed by the MWU block is routing. The direct benefit is a reduced test time as compared to the use of fully functional multicast, proportional to the number of processes that are eliminated. The advantages of using this simplified multicasting scheme are reduced complexity, lower silicon area required by MWU, and shorter transport latency for the test-data packets.

V. Test Scheduling

Our approach to NoC infrastructure test does not use a dedicated TAM to transport test-data to NoC CUTs. Instead, we propagate the test data toward the components in a recursive wavelike manner, via the NoC components already tested. This method eliminates entirely the need for a dedicated TAM and saves the corresponding resources. Another distinct advantage of our method over the dedicated TAM is that the test data can be delivered at a rate independent of the size of the NoC under test. The classic shared-bus TAMs are not able to deliver test data at a speed independent of the size of the SoC under test. This is due to the large intrinsic capacitive load of the TAM combined with the load of multiple cores serviced by the TAM [19].

An important preprocessing task that determines the total test time is referred to as test scheduling. Many of the previous research efforts have been devoted to reducing the test time of large SoCs designs by increasing test concurrence using advanced test architectures and test-scheduling algorithms. In the case of NoC-based MP-SoCs, the data-communication infrastructure contributes to the total test time of the chip, and this contribution must also be minimized. The test-scheduling problem can be formulated as optimizing the spatial and temporal distribution of test data such that a set of constraints is satisfied. In this paper, the specific constraint is the test time required to perform postmanufacturing test of the NoC infrastructure. At this point, we assume that the test data are already available and organized in test packets, as outlined in Sections III and IV. We also assume that, when required, the fault-free part of the NoC can transport the test data to the NoC CUTs using unicast or multicast, as detailed in Section V.

With these considerations, we can identify two different components of the test time for each NoC component. The first one is directly related to the amount of time required to deliver the test patterns to the NoC element that is targeted, and we call it transport time. The second component of the test time represents the amount of time that is actually needed to apply the test patterns to the targeted element and perform the actual testing procedure. We call this latter component test time per element, where element refers to a link segment, a FIFO buffer, or a routing/arbitration block.

A. Test-Time Cost—Problem Formulation

In order to search for an optimal scheduling, we must first use the two components of the test time to determine a suitable cost function for the complete testing process. We then compute the test cost for each possible switch that can be used as a source for test-packet injection. After sequencing through all the switches and evaluating their costs, we choose the one with the lowest cost as the source.

We start by introducing a simple example that illustrates how the test time is computed in the two test-transport modes, unicast and multicast, respectively. Consider the example in Fig. 9, where switch S_1 and links l_1 and l_2 are already tested and fault-free and switches S_2 and S_3 are the next switches to be tested. When test data are transmitted in the unicast mode, one and only one NoC element goes into the test mode at a time, at any given time, as shown in Fig. 9(a) and (b).

Then, for each switch, the test time equals the sum of the transport latency and the effective test time of the switch. The latter term accounts for testing the FIFO buffers and RLB in the switches. Therefore, the total test time $T^u_{2,3}$ for testing both switches S_2 and S_3 is

$$T^u_{2,3} = 2(T_{l,L} + T_{l,S}) + 2T_{t,S}$$

where $T_{l,L}$ is the latency of the interswitch link (usually one clock cycle if the link is reasonably short—see [19] for more details), $T_{l,S}$ is the switch latency (the number of cycles required for a flit to traverse a NoC switch from input to output), and $T_{t,S}$ is the time required to perform the actual testing of the switch (i.e., $T_{t,S} = T_{\text{FIFO}} + T_{\text{RLB}}$).

Following the same reasoning for the multicast transport case in Fig. 9(c), the total test $T^m_{2,3}$ time for testing switches S_2 and

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:20:40 UTC from IEEE Xplore. Restrictions apply.

S_3 can be written as

$$T^m_{2,3} = (T_{l,L} + T_{l,S}) + T_{t,S}.$$

From this simple example, we can infer that there are two mechanisms that can be employed for reducing the test time: reducing the transport time of test data and reducing the effective test time of NoC components. The transport time of test patterns can be reduced in two ways:

1) by delivering the test patterns on the shortest path from the test source to the element under test;
2) by transporting multiple test patterns on nonoverlapping paths to their respective destinations.

The test-application time per NoC element is governed by the considerations described in Sections III and IV. Therefore, in order to reduce it, we would need to reevaluate the fault models or the overall test strategy (i.e., to generate test data locally for each element, with the respective incurred overhead [33]). Within the assumptions in this paper (all test data are generated offline and transported to the element under test), the only feasible way to reduce the effective test time per element is to overlap the test of more NoC components. The direct effect is the corresponding reduction of the overall test time. This can be ultimately accomplished by employing the multicast transport and applying test data simultaneously to more components.

B. *Test-Transport-Time Minimization*

With the goal of minimizing the time required to deliver test patterns to the elements under test, we formulate the problem using a graph representation of the NoC infrastructure. We then find, for each NoC component, the shortest path from an arbitrary node on the NoC graph, traversing only the previously tested fault-free components. The total transport time equals the sum of all transport latencies for the set of shortest paths corresponding to the chosen node; consequently, since these paths are minimal, the total test time corresponding to the respective node is also minimal. By repeating the procedure similarly for all possible nodes in the network and choosing the solution that delivers the shortest test time, we are guaranteed to obtain the minimum test-transport time.

The graph representation of the NoC infrastructure used to find the minimum test-transport latency is obtained by representing each NoC element as a directed graph $G = (S, L)$, where each vertex $s_i \in S$ is a NoC switch and each edge $l_i \in L$ is an interswitch link. Each switch is tagged with a numerical pair $(T_{l,s}, T_{t,s})$ corresponding to switch latency and switch test time. Each link is similarly labeled with a pair $(T_{l,L}, T_{t,L})$ corresponding to link latency and link test time, respectively. For each edge and vertex, we define a symbolic toggle t, which can take two values: $\mathbf{N}$ and $\mathbf{T}$. When $t = \mathbf{N}$, the cost (weight) associated with the edge/vertex is the latency term, which corresponds to the normal operation. When $t = \mathbf{T}$, the cost (weight) associated with the edge/vertex is the test time (of the link or switch) and corresponds to the test operation.

We use a modified version of Dijkstra's shortest path algorithm [20] for graph traversal in order to find a test scheduling with minimum test cost. Dijkstra's algorithm solves the single-source shortest path problem for a directed graph with nonnegative-edge weights. It is known for its efficient and simple implementation.

Initially, the t toggle is equal to $\mathbf{T}$ for all edges/vertices. We start by choosing a node and traversing the NoC graph using Dijkstra's algorithm. Every time an element is encountered whose t toggle is $\mathbf{T}$, the test cost function is updated with the corresponding term, and t is switched to $\mathbf{N}$. When an element whose toggle is $\mathbf{N}$ is encountered, the test function is updated with the corresponding term (the element's current weight) and t remains unchanged. There are slight differences in our approach as compared to the classic algorithm: The vertices possess their own weights; the weights of the vertices and edges change dynamically during graph traversal. In addition, after a directed edge is traversed, the edge in the opposite direction is traversed as soon as possible. An additional constraint that we place on the modified algorithm is that all toggles t must be switched to $\mathbf{N}$ by the end of the algorithm if no fault is detected. This ensures that no edges remain that have not been traversed (i.e., no interswitch links remain untested). However, these differences are minor and only affect the way in which the test cost function is calculated.

The test cost function is computed differently, depending on whether unicast or multicast is employed. In the following, we present the algorithms used to determine a minimum-cost test scheduling for these two test-delivery methods.

Unicast-Test Scheduling

Problem Formulation: Given the graph $G(S, L)$ and the pairs $(T_{l,s}, T_{t,s})$ and $(T_{l,L}, T_{t,L})$ and assuming that only one vertex/edge whose toggle t equals $\mathbf{T}$ can be visited at a time, determine a graph-traversal sequence that covers all vertices and edges and has a minimum associated test cost function $F_{TC,u}$.

The fact that only one toggle can be switched at a time accounts for the unicast transport mechanism, when the NoC components are tested sequentially. The unicast test cost function $F_{TC,u}$ is defined recursively as

$$\begin{aligned} F_{TC,u}&(\text{current}) \\ &= F_{TC,u}(\text{previous}) \\ &\quad + \sum_{\text{all elements on path}} (T_{l,L}, T_{l,S}) \\ &\quad + \begin{cases} T_{t,L}, & \text{if current element is a link} \\ T_{t,S}, & \text{if current element is a switch} \end{cases} \end{aligned} \qquad (1)$$

where current and previous refer to the NoC elements (switch or link) under test. We can observe that, in each step of the sequence, we add the latency terms corresponding to the path along which the test data are transported and the terms corresponding to the effective test procedure per element. Hence, the unicast-based test procedure can be minimized by minimizing the latency component of the test cost function. We use this observation to realize a minimum-cost test scheduling by ensuring

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:20:40 UTC from IEEE Xplore. Restrictions apply.

that each element of the graph $G(S,L)$ is visited on the shortest path from the test source.

Unicast scheduling algorithm:
for each $s \in S$
—initialization—
Uni_min (G, S, L, w, t)
  for each vertex s in G
    v_toggle(s) := T;*—initialize all switches as untested*
    v_weight(s) := $T_{t,S}$;
    d[s] := ∞;
    previous(s) := undefined;
  for each edge between vertices u, v in G
    e_toggle(u, v) := T;*—initialize all links as untested*
    e_weight(u,v) := $T_{t,L}$;
    Q := S **union** L;
    R := empty set;
    $F_{TC,u}$:= 0;
—graph traversal on all shortest paths from switch s*—*
  while Q is not an empty set*—graph traversal*
    u := Extract Min{S};*—pick switch with min.* $T_{t,S}$
    R := R **union** {u};
    for each edge (u,v) outgoing from u
      if $\{\mathrm{d[u]}+\mathrm{e_weight(u,v)}<\mathrm{d[v]}\,\mathrm{and}\,\mathrm{v_toggle(v)}=\mathrm{T}\}$
        d[v] := d[u] + weight(u, v);
        update $F_{TC,u}$;
        v_toggle(v) := N;*—set unicast mode—*
        e_toggle(u, v) := N;
        v_weight(u) := $T_{l,S}$;*—change weights—*
        e_weight(u, v) := $T_{l,L}$;
        previous[v] := u;
return $F_{TC,u}$;
choose $\{s_{\min}, F_{TC,u,\min}\}$
end.

$\mathrm{d[u]}$ denotes the distance from the current test source to switch u under test when the NoC graph $G(S,L)$ is traversed using the modified Dijkstra algorithm, and it represents the test injection time corresponding to switch u. At the end of the algorithm, the switch $s_{\min}$ that yielded a test scheduling with a minimum cost $F_{TC,u,\min}$ is selected as the final test source.

The test cost function for unicast transport $F_{TC,u}$ is updated according to (1). This algorithm returns the optimum test source switch in the NoC and a test scheduling that is minimized with respect to test time, since all test paths that are returned have the shortest length.

Table I shows the test scheduling obtained using the unicast-based test-time-minimization algorithm, applied to the four-switch network in Fig. 10, when switch S_1 is selected as the initial test source. As outlined in Table I, the scheduling algorithm returns the test time and the path for each test packet.

At the end of the algorithm, when all possible test sources were exhaustively exercised, the optimum solution would be the one that selects S_2 (or S_3) as test sources. Inter-switch links are indicated in full, as pairs of unidirectional connections.

TABLE I
UNICAST-TEST-DATA SCHEDULING FOR THE FOUR-SWITCH NETWORK IN FIG. 10, INDICATING TEST COST AND TEST PATHS (TEST SOURCE = S_1)

Element under test	Test packets path	Unicast Test Cost $F_{TC,u}$
S_1	-	$T_{t,S}$
l_1	S_1	$T_{t,S}+T_{1,S}+T_{t,L}$
l_2	S_1	$T_{t,S}+2T_{1,S}+2T_{t,L}$
S_2	$S_1 \to l_1$	$2T_{t,S}+3T_{1,S}+2T_{t,L}+T_{1,L}$
l_1’	$S_1 \to l_1 \to S_2$	$2T_{t,S}+5T_{1,S}+3T_{t,L}+2T_{1,L}$
S_3	$S_1 \to l_2$	$3T_{t,S}+6T_{1,S}+3T_{t,L}+3T_{1,L}$
l_2’	$S_1 \to l_2 \to S_3$	$3T_{t,S}+8T_{1,S}+4T_{t,L}+4T_{1,L}$
l_5	$S_1 \to l_2 \to S_3$	$3T_{t,S}+10T_{1,S}+5T_{t,L}+5T_{1,L}$
l_5’	$S_1 \to l_1 \to S_2$	$3T_{t,S}+12T_{1,S}+6T_{t,L}+6T_{1,L}$
l_3	$S_1 \to l_1 \to S_2$	$3T_{t,S}+14T_{1,S}+7T_{t,L}+7T_{1,L}$
S_4	$S_1 \to l_1 \to S_2 \to l_3$	$4T_{t,S}+16T_{1,S}+6T_{t,L}+9T_{1,L}$
l_3’	$S_1 \to l_1 \to S_2 \to l_3 \to S_4$	$4T_{t,S}+19T_{1,S}+8T_{t,L}+11T_{1,L}$
l_4	$S_1 \to l_2 \to S_3$	$4T_{t,S}+21T_{1,S}+9T_{t,L}+12T_{1,L}$
l_4’	$S_1 \to l_2 \to S_3 \to l_4 \to S_4$	$4T_{t,S}+24T_{1,S}+10T_{t,L}+14T_{1,L}$

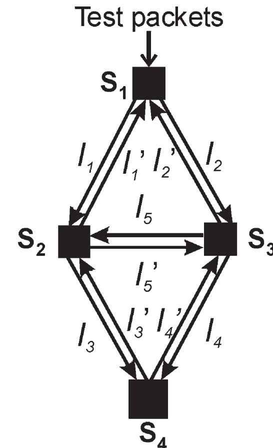


Fig. 10. Four-switch network with unidirectional links.

Multicast-Test Scheduling

Problem formulation: Given the graph $G(S,L)$, the pairs $(T_{l,s}, T_{t,s})$ and $(T_{l,L}, T_{t,L})$, and assuming that all vertices/edges whose toggle t equals $\mathbf{T}$ and are adjacent to edges/vertices whose toggle equals $\mathbf{N}$ can be visited at a time, determine a graph-traversal sequence that covers all vertices and edges and has a minimum associated test cost function F_{TC}.

The fact that more toggles can be switched at a time accounts for the multicast-transport mechanism, when the NoC components are tested concurrently.

The multicast-test cost function $F_{TC,m}$ is defined recursively as (2), shown at the bottom of the next page, where current and previous refer to the value of the multicast-test cost function in the corresponding multicast-test step. In each multicast step, the test cost function is updated, adding the latency of the longest multicast paths and the greatest test time required by the elements under test in the current test step.

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:20:40 UTC from IEEE Xplore. Restrictions apply.

Hence, the multicast-based test procedure can be minimized by minimizing both the latency component of the test cost function and the total test by transporting test data to more components concurrently. We use this observation to realize a minimum-cost test scheduling by ensuring that each element of the graph $G(S, L)$ is visited on the shortest path from the test source, and all elements whose toggle equals T and are adjacent to elements whose toggle equals N are visited at the same time.

The pseudocode of the algorithm that realizes the multicast-test scheduling is presented as follows.

Multicast scheduling algorithm:
for each s $\in S$
—initialization—
  Multi_min (G, S, L, w, t)
    for each vertex s in G
      v_toggle(s) := T;*—initialize all switches as untested*
      v_weight(s) := $T_{t,S}$;
      d[s] := ∞;
      previous(s) := undefined;
    for each edge between vertices u, v in G
      e_toggle(l) := T;*—initialize all links as untested*
      e_weight(l) := $T_{t,L}$;
    Q := S **union** L;
    R := empty set;
    $F_{TC,m}$:= 0;
—graph traversal on all shortest paths from switch s—
    while Q is not an empty set*—graph traversal*
      u := Extract Min{S};*—pick switch with min. $T_{t,S}$*
      R := R **union** {u};
      for all edges (u,v) outgoing from u
        for all nodes v
          if {d[u] + weight(u, v) < d[v] and toggle(v) = T}
            d[v] := d[u] + weight(u, v);
            v_toggle(v) := N;*—set multicast mode—*
            e_toggle(u, v) := N;
            v_weight(u) := $T_{l,S}$;*—change weights—*
            e_weight(u, v) := $T_{l,L}$;
            previous[v] := u;
      update $F_{TC,m}$;
  return $F_{TC,m}$;
  choose $\{s_{\min}, F_{TC,u,\min}\}$
  end.

The test cost function is updated according to (2), once per each multicast step. Since this happens concurrently for more nodes (edges), the total value of $F_{TC,m}$ will be lower than that of the unicast-test cost function $F_{TC,u}$ for identical networks.

TABLE II
MULTICAST-TEST-DATA SCHEDULING FOR THE FOUR-SWITCH NETWORK IN FIG. 10, INDICATING TEST COST AND TEST PATHS (TEST SOURCE = S_1)

Elements under test	Test packets path	Multicast Test Cost $F_{TC,m}$
S_1		$T_{t,S}$
l_1, l_2	S_1	$T_{t,S}+T_{1,S}+T_{t,L}$
S_2, S_3	$S_1 \rightarrow \{l_1, l2\}$	$2T_{t,S}+2T_{1,S}+T_{t,L}+T_{1,L}$
l_1', l_2', l_3, l_4, l_5, l_5'	$S_1 \rightarrow \{l_1, l_2\} \rightarrow \{S_2, S_3\}$	$2T_{t,S}+4T_{1,S}+2T_{t,L}+2T_{1,L}$
S_4	$S_1 \rightarrow l_1 \rightarrow S_2 \rightarrow l_3$	$3T_{t,S}+6T_{1,S}+2T_{t,L}+4T_{1,L}$

Table II shows the test scheduling obtained by using the multicast-based algorithm, when switch S_1 is selected as a test source. Once the algorithm terminates and, thereby, all possible test sources are considered, in this instance, the optimum test scheduling would select S_2 (or S_3) as the test source yielding the optimal solution.

Both test-scheduling algorithms are direct mappings of Dijkstra's shortest path algorithm for the NoC graph, the difference being in the way the cost functions are calculated and updated. Therefore, the complexity of the algorithms can be estimated as $O(e \log \nu)$, where e is the number of directed edges and ν is the number of vertices in the NoC graph [20].

VI. EXPERIMENTAL RESULTS AND ANALYSIS

In order to evaluate the efficiency of our testing methods in a realistic manner, we first need to present some implementation details that made possible the application of methods and algorithms described in the previous sections.

A. Test-Output Evaluation

The methods described in this paper are primarily targeted for postmanufacturing testing, where the objective is to deliver a set of test patterns in the shortest possible time and the final outcome is the fail/pass decision.

In classical core-based testing, test data are injected from a test source, transported and applied to the core under test, and, then, the test output is extracted and transported to the test sink for comparison with the expected response [22]. In this paper, we adopted a more effective solution first proposed in [32], where the expected data are sent together with the input

$$F_{TC,m}(\text{current}) = F_{TC,m}(\text{previous}) + \underset{\text{all adjacent elements}}{\text{Max}} \left(\sum_{\text{all elements on path}} (T_{l,L}, T_{l,S}) \right) + \underset{\text{all adjacent elements}}{\text{Max}} \left(\begin{cases} T_{t,L}, & \text{if current element is a link} \\ T_{t,S}, & \text{if current element is a switch} \end{cases} \right) \quad (2)$$

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:20:40 UTC from IEEE Xplore. Restrictions apply.

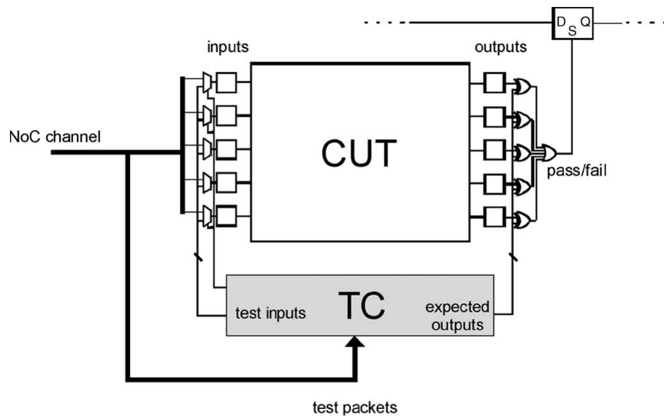


Fig. 11. Test-packet processing and output comparison.

test data, and the comparison is performed locally at each CUT. A clear advantage of this approach is that, in this case, since there is no need to extract the test output and to transport it to a test sink, the total test time on the NoC infrastructure can be significantly reduced. Moreover, the test protocol is also simplified, since this approach eliminates the need for a flow control of test output data (in terms of routing and addressing). The tradeoff is a small increase in hardware overhead due to additional control and comparison circuitry and increased size of the test packets (which now contain the expected output of each test vector, interleaved with test input data).

As shown in Fig. 11, the test packets are processed by test-controller (TC) blocks that direct their content toward the inputs/outputs of the CUT and perform the synchronization of test output and expected data. These data are compared individually for each output pin, and, in case of a mismatch, the component is marked as faulty by raising the pass/fail flag. The value of this flag is subsequently stored in a pass–fail flip-flop, which is a part of a shift register that connects pass–fail flops of all switches. The content of this register is serially dumped off-chip at the end of the test procedure.

In our implementation, the TC block is shared by a switch and its adjacent links in order to reduce the area overhead of the scheme.

B. Test Modes for NoC Components

The components of the NoC fabrics are quite heterogeneous with respect to their test requirements, and their test modes differ significantly. For each type of component (FIFO buffers, routing/arbitration blocks, interswitch links), we provide test modes for executing the test procedure in a manner suitable for the nature of the CUT.

The test data can be injected from the external automatic test equipment (ATE) by multiplexing the functional I/Os so that, in test mode, test packets are directed toward the first switch under test determined according to the algorithm in Section V. Subsequently, the test path is constructed progressively by adding the individual NoC components, after their test procedure is completed successfully. In other words, chip I/Os are multiplexed to a NoC channel, through which test data are injected into the NoC to access the flip-flops within the RLB and FIFO.

As stated in Section III, we adopt scan insertion as the design for test (DfT) strategy for the routing/arbitration blocks. The number of scan chains is chosen to be equal to the interswitch-link width.

Since we perform functional test in the case of the FIFO blocks, they do not require special control signals or additional hardware to manage the test-input data. In this case, the TC block only separates the input data from the expected output data and performs the output comparison for the FIFO test. A similar situation arises for the interswitch links, except that the link test packets do not contain any expected output data. Instead, since the expected output is identical to the MAF test inputs, the TC block duplicates the latter and thus creates the expected output, which is then used to perform the test-output comparison.

C. Test Scheduling Complexity and Results

We evaluated the proposed scheduling algorithms with respect to their run-times (time required to run the program that implements the algorithms on a computer) and final test cost (the value of the test cost function associated with the optimal scheduling solution returned by each algorithm). For estimation purposes, we considered two different types of NoC topologies: the mesh and BFT [10]. For each of these types, we built NoCs of three different sizes that we consider representative of the level of integration at which the NoC paradigm becomes a viable solution: 16-IP (“small NoC”), 64-IP (“medium” NoC), and 256-IP (“large” NoC). Correspondingly, the number of switches that service each NoC instance differs for the two types of topologies, due to the fact that they are direct (mesh) and indirect (BFT) networks; their respective number of switches is given in Table III (more details on the particular characteristics of these topologies can be found in [10], [24], and [25]). The switches were designed using commercial-grade digital-design tools by Synopsys and synthesized in a 90-nm standard-cell technology from ST Microelectronics. The test patterns for the routing/arbitration blocks were obtained running Synopsys’ TetraMAX ATPG tool [26] and arranged in test packets by in-house written scripts. The FIFOs were also designed using standard cells; however, they were isolated from the logic blocks for test-generation purposes, and their test data were obtained and organized in test packets according to Sections III and IV. The size of the FIFOs was set to four flits per virtual channel, with four virtual channels per port and symmetrical input–output buffering (for more design details, please refer to [10] and [27]).

For all NoC instances, the bit width of the interswitch links was set to 32. All interswitch links consist of a pair of unidirectional interconnections. The MAF test packets were generated according to the MAF model presented in Section II-A. The code implementing the two scheduling algorithms was run on a Linux PC with a 2-GHz X86-family processor and 1 GB of DRAM.

Table III shows the amount of time required to obtain an optimal scheduling running the two algorithms presented in Section VI.

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:20:40 UTC from IEEE Xplore. Restrictions apply.

TABLE III
TEST-SCHEDULING RUN-TIMES

NoC type & size		Scheduling - unicast run-time [s]	Scheduling – multicast run-time [s]
Mesh	4 x 4	4	3
	8 x 8	12	9
	16 x 16	33	27
BFT	6	2	2
	28	7	5
	120	15	11

TABLE IV
GATE COUNT AND COMPARISON FOR THE PROPOSED TEST MECHANISMS (PER SWITCH)

NoC type & size		Unicast [gates]	Test-only multicast [gates]	Functional multicast [gates]	%diff
Mesh	4 x 4	524	825	1025	19.5
	8 x 8	548	792	1025	22.7
	16 x 16	576	721	1025	29.6
BFT	6	693	816	1210	32.5
	28	718	771	1210	36.3
	120	736	722	1210	40.3

This does not include the time required to run the ATPG tool to generate the test patterns for the logic blocks, since this has to be done for any test-transport implementation. Clearly, the test-scheduling methods do not require significant run-times, even for relatively large NoCs (256 nodes mesh, 120-switch BFT).

An important quality metric of our test methodology is the silicon-area overhead of the unicast- and multicast-test-data transport mechanisms. There are two major components that contribute to the overhead of the schemes. The first is the TC unit, in charge of test-packet processing, test-input-data injection to CUT inputs, and test-output comparison. The second is the MWU, which implements the single-source multiple-destination test-data-transport feature. We compare the gate-count of the test-only multicast solution with the gate-count of the functional multicast implementation. Table IV shows the gate-count for the unicast, test-only multicast, and functional multicast implementations. The gate-count for the unicast case corresponds to the TC blocks, which are identical for a given topology. The TC blocks differ for different topologies, since they have to handle a different number of components (FIFOs, routing blocks, interswitch links) adjacent to a switch. The gate-count reported for the test-only multicast is the sum of TC gate-count and the MWU gate-count, averaged for all the switches in the respective NoC instance. The averaging is needed because the test-only multicast feature is implemented selectively only for those ports per switch which are involved in multicast data transport. The list of ports that need to be connected to the MWU for each switch was obtained by examining the optimal test scheduling in Section VI-B. The last column in Table IV, labeled %diff, shows the percent difference between the gate-count of the functional multicast implementation and the test-only one.

The absolute area required to implement the test mechanisms we proposed is quite acceptable, particularly when compared to the typical gate-count of a NoC switch (around 30 000 gates, as reported in [10], [30], and [31]).

Moreover, when multicast data transport is adopted, by implementing this feature selectively for only the switch ports on multicast paths, significant area reduction can be obtained (up to 40% when compared to the functional multicast realization for the NoC instances in this paper).

The MWU hardware introduces a certain amount of delay in the data path of the packets, equivalent to the delay of a demultiplexer/multiplexer pair, as shown in Fig. 7. This delay adds to the delay of the routing block RLB. Through gate-level design and analysis, we found the equivalent delay of the MWU to be equal to 3 fan-out of four (FO4) delay units. The typical delay of the RLB block is around 6 FO4 units [19]. Therefore, the total delay of the RLB and MWU blocks is around 9 FO4 units, which fits well within the limit of 15 FO4 units according to International Technology Roadmap for Semiconductors projected trends for high-performance multiprocessors.

The most important quality metric for the test methods developed in this paper is the corresponding test time required to perform both the unicast- and multicast-based schemes. We compare our method with two previously proposed test methods for NoC routers. None of these prior methods addresses the test of interconnects. We include them in our comparison since they are the most relevant NoC test methods currently available. In the following, we provide a brief overview of the two NOC test methods used for comparison.

The work in [21] proposed a test methodology for NoC routers based on a combination of flat-core–full-scan and hierarchical-core–full-scan methods. For test purposes, the NoC is considered as a flat core and wrapped with an IEEE 1500 test wrapper for test-data access to internal scan chains of individual NoC routers. Test data are delivered in parallel to all identical scan chains of the routers, and the test responses are compared internally. The number of comparators used depends on the number of scan chains: A comparator block is required for each scan chain in the routers. Ideally, all routers are tested in parallel, and a single comparator is needed. For NoCs of larger size where practical fan-out limitations cannot be ignored, the number of scan chains per router (and, correspondingly the number of comparator blocks) must be increased.

In [35], the authors use progressive transport of test data to the routers under test, whereas the NoC channels are assumed fault-free. Test data are organized in packets and injected from ATE using input ports, routed through several channels and routers, then captured by the test wrapper of the router under test. This wrapper is similar to the IEEE 1500 test wrapper. The main concern for test-packet scheduling is to avoid using untested routers in their paths. Moreover, only unicast data transport is considered, with no particular focus on minimizing

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:20:40 UTC from IEEE Xplore. Restrictions apply.

TABLE V
TEST-TIME RESULTS AND COMPARISON

NoC type & size		Test method and test time [cycles]				Relative test time improvement		
		Test time [21]*	Test time [35]*	Unicast	Multicast	Test time improvement [21]/ multicast	Test time improvement [35]/ multicast	Test time improvement unicast/ multicast
Mesh	4 x 4	9,451	18,840	19,958	4,603	2X	4X	4.3X
	8 x 8	40,309	79,921	85,122	7,559	5.3X	10.5X	11.2X
	16 x 16	105,775	209,720	223,368	15,223	7X	13.7	14.6X
BFT	6	7,226	15,352	16,107	3,258	2.2X	4.7X	4.9X
	28	27,690	58,830	61,724	7,036	4X	8.3X	8.7X
	120	125,576	266,896	280,073	8,107	15.5X	33X	34.5X

** Test time corresponding to interconnect test is not included (NoC links are assumed fault-free in [21] and [35]).*

the test-transport time. Test response is processed locally on-chip through the use of either comparator blocks or multiple-input shift registers. In [35], test-time results are presented for both routers-only test and integrated-routers/cores test. For the purpose of this comparison, we only considered the case of routers-only test.

Depending on the availability of test resources (I/O pins) and other considerations, designers may choose to include dedicated TAM that can be used in combination with NoC-based test transport. In this paper, we considered that the NoC reuse is the only mechanism used to transport test data.

The results showing the test time required by the unicast, multicast, and the previous test methods presented in [21] and [35] are summarized in Table V.

Before discussing the specifics of the experimental results, we note that no direct comparison is intended between the two types of architecture considered for test-time evaluation. The NoC structures studied here have very different topologies and sizes (in terms of number of switches).

The reason for choosing these particular topologies and sizes is that they represent two extreme cases with respect to the nature of their respective topologies: The mesh is a more uniform architecture, whereas the BFT is hierarchical. For lack of standard NoC benchmark circuits, we believe these topologies offer a reasonable representation of the practical cases.

The routing policies employed for the unicast and functional multicast cases were the dimensional routing (e-cube) [18] for mesh topologies and least common ancestor for the BFT topologies [10]. For the test-only multicast, the routing was customized according to the algorithm in Section V-B.

Looking at the test times in Table V, we notice that the unicast test and test time reported in [35] are very similar. This is because, in each case, test data are delivered sequentially to each CUT. The test volume of our unicast method is larger than the one in [35] due to the fact that we include test packets for testing the NoC channels. The larger test volume is compensated, however, by minimizing the test-transport time. Compared to [21], the unicast approach appears to perform worse but only because [21] uses a reduced test-data volume (no interconnect test is performed), and second-order effects, such as the effect of scan-input fan-out, are not considered for larger NoCs. Our unicast method has, however, an advantage that is not apparent from Table V: It can deliver test data at the nominal operating speed of the NoC infrastructure regardless of the NoC size. The method presented in [21] may not be able to carry the test data at the NoC nominal frequency, particularly in the case of large-size architectures, when the fan-out of the scan input will increase significantly and the maximum switching rate of the scan-chains accordingly. Comparing the test times with the multicast, the superiority of the multicast-test-data transport is evident. The improvement in test speed range from 2× for the case of small-size NoCs to 34× for large size NoCs. As with the unicast-transport method, the multicast mechanism is able to transport test data at the nominal operating speed of the NoC, thus making possible at-speed transport of test-packet test with no additional cost.

VII. CONCLUSION

We have presented a method for testing the communication fabric of NoC-based MP-SoCs. The novelty of this method lies in the reuse of the NoC fabric for test-data transport in a recursive manner and in exploiting the inherent parallelism of the NoC architectures for speeding the test-data transport and reducing the test time. We developed test-scheduling algorithms for two types of test-data transport: sequential (unicast) and concurrent (multicast). Our methods integrate the test of all types of NoC components (buffers, links, and routing blocks) in a unified fashion. The efficiency of the test method was evaluated for synthetic NoCs of different topologies and sizes. The results of this assessment show significant speedup of test-data delivery as compared to previously proposed NoC test methods, from 2× for small-size NoCs up to 34× for larger networks. A distinct advantage of both unicast- and multicast-test methods is the possibility of delivering test data at the nominal operating speed of the NoC without additional overhead.

As future work, we intend to extend our method to include the functional cores of the MP-SoC and include other constraints such as test power.

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:20:40 UTC from IEEE Xplore. Restrictions apply.

ACKNOWLEDGMENT

The authors would like to thank M. Goessel and E. Sogomonyan of the University of Potsdam for the valuable comments and discussions.

REFERENCES

[1] P. Magarshack and P. G. Paulin, "System-on-chip beyond the nanometer wall," in *Proc. DAC*, Anaheim, CA, Jun. 2–6, 2003, pp. 419–424.
[2] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.
[3] P. Pande, C. Grecu, A. Ivanov, R. Saleh, and G. De Micheli, "Design, synthesis, and test of networks on chip," *IEEE Des. Test Comput.*, vol. 22, no. 5, pp. 404–413, Sep./Oct. 2005.
[4] E. J. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, and C. Wouters, "A structured and scalable mechanism for test access to embedded reusable cores," in *Proc. ITC*, 1998, pp. 284–293.
[5] E. Cota, L. Caro, F. Wagner, and M. Lubaszewski, "Power aware NoC reuse on the testing of core-based systems," in *Proc. ITC*, 2003, pp. 612–621.
[6] C. Liu, V. Iyengar, J. Shi, and E. Cota, "Power-aware test scheduling in network-on-chip using variable-rate on-chip clocking," in *Proc. IEEE VLSI Test Symp.*, 2005, pp. 349–354.
[7] B. Vermeulen, J. Dielissen, K. Goossens, and C. Ciordas, "Bringing communications networks on a chip: Test and verification implications," *IEEE Commun. Mag.*, vol. 41, no. 9, pp. 74–81, Sep. 2003.
[8] M. Nahvi and A. Ivanov, "Indirect test architecture for SoC testing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 7, pp. 1128–1142, Jul. 2004.
[9] M. Cuviello, S. Dey, X. Bai, and Y. Zhao, "Fault modeling and simulation for crosstalk in system-on-chip interconnects," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, San Jose, CA, Nov. 1999, pp. 297–303.
[10] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for MP-SoC interconnect architectures," *IEEE Trans. Comput.*, vol. 54, no. 8, pp. 1025–1040, Aug. 2005.
[11] I. Saastamoinen, M. Alho, and J. Nurmi, "Buffer implementation for Proteo network-on-chip," in *Proc. ISCAS*, 2003, vol. 2, pp. II-113–II-116.
[12] H. Wang, L.-S. Peh, and S. Malik, "Power-driven design of router microarchitectures in on-chip networks," in *Proc. 36th Annu. IEEE/ACM Int. Symp. MICRO*, 2003, pp. 105–116.
[13] A. J. Van de Goor, I. Schanstra, and Y. Zorian, "Functional test for shifting-type FIFOs," in *Proc. Eur. Des. Test Conf.*, Mar. 1995, pp. 133–138.
[14] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. DAC*, Las Vegas, NV, Jun. 18–22, 2001, pp. 683–689.
[15] *Intel IXP2400 datasheet*. [Online]. Available: http://www.intel.com/design/network/products/ npfamily/ixp2400.htm
[16] A. Radulescu, J. Dielissen, K. Goossens, E. Rijpkema, and P. Wielage, "An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network configuration," in *Proc. IEEE DATE*, 2004, vol. 2, pp. 878–883.
[17] P. P. Pande, C. Grecu, A. Ivanov, and R. Saleh, "Switch-based interconnect architecture for future systems on chip," in *Proc. SPIE—VLSI Circuits and Systems*, 2003, vol. 5117, pp. 228–237.
[18] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks—An Engineering Approach*. San Mateo, CA: Morgan Kaufmann, 2002.
[19] C. Grecu, P. Pande, A. Ivanov, and R. Saleh, "Timing analysis of network on chip architectures for MP-SoC platforms," *Microelectron. J.*, vol. 36, no. 9, pp. 833–845, Sep. 2005.
[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, 2001.
[21] A. M. Amory, E. Briao, E. Cota, M. Lubaszewski, and F. G. Moraes, "A scalable test strategy for network-on-chip routers," in *Proc. IEEE ITC*, 2005, pp. 591–599.
[22] Y. Zorian, E. J. Marinissen, and S. Dey, "Testing embedded-core-based system chips," *Computer*, vol. 32, no. 6, pp. 52–60, Jun. 1999.
[23] V. Iyengar and K. Chakrabarty, "Test bus sizing for system-on-a-chip," *IEEE Trans. Comput.*, vol. 51, no. 5, pp. 449–459, May 2002.
[24] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. C-34, no. 10, pp. 892–901, Oct. 1985.
[25] A. DeHon, "Compact, multilayer layout for butterfly fat-tree," in *Proc. 12th Annu. ACM Symp. Parallel Algorithms and Architectures*, 2000, pp. 206–215.
[26] *Synopsys TetraMAX ATPG Methodology Backgrounder*. [Online]. Available: www.synopsys.com/products/test/tetramax_wp.html
[27] *Networks on Chip*, A. Jantsch and H. Tenhunen, Eds. Norwell, MA: Kluwer, 2003.
[28] *IEEE Standard Testability Method for Embedded Core-based Integrated Circuits*, 2005. IEEE Standard 1500-2005.
[29] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Boston, MA: Springer-Verlag, 2005.
[30] F. G. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "HERMES: An infrastructure for low area overhead packet-switching networks on chip," *Integr. VLSI J.*, vol. 38, no. 1, pp. 69–93, 2004.
[31] J. Hu and R. Marculescu, "DyAD—Smart routing for networks-on-chip," in *Proc. DAC*, 2004, pp. 260–263.
[32] M. Nahvi and A. Ivanov, "An embedded autonomous scan-based results analyzer (EARA) for SoC cores," in *Proc. 21st IEEE VLSI Test Symp.*, 2003, pp. 293–298.
[33] C. Grecu, P. P. Pande, A. Ivanov, and R. Saleh, "BIST for NoC interconnect infrastructures," in *Proc. 24th IEEE VLSI Test Symp.*, 2006, pp. 30–35.
[34] C. Grecu, P. P. Pande, B. Wang, A. Ivanov, and R. Saleh, "Methodologies and algorithms for testing switch-based NoC fabrics," in *Proc. 20th IEEE Int. Symp. DFT*, 2005, pp. 238–246.
[35] C. Liu, Z. Link, and D. K. Pradhan, "Reuse-based test access and integrated test scheduling for network-on-chip," in *Proc. DATE*, 2006, pp. 303–308.
[36] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Comput. Surv.*, vol. 38, no. 1, pp. 1–51, 2006.
[37] *International Technology Roadmap for Semiconductors*. 2006 update. [Online]. Available: http://www.itrs.net/Links/2006Update/2006UpdateFinal.htm

Cristian Grecu (S'06) received the B.S. and M.Eng. degrees in electrical engineering from the Technical University of Iasi, Iasi, Romania, and the M.A.Sc. degree from the University of British Columbia, Vancouver, BC, Canada, where he is currently working toward the Ph.D. degree in the Department of Electrical and Computer Engineering.

His research interests focus on design and test of large systems-on-chip, with particular emphasis on their data-communication infrastructures.

André Ivanov (S'81–M'85–SM'95–F'06) received the B.Eng. (with honors), M.Eng., and Ph.D. degrees in electrical engineering from McGill University, Montreal, UC, Canada.

From 1995 to 1996, he spent a sabbatical leave with PMC-Sierra, Vancouver, BC. He has held invited professorship positions with the University of Montpellier II, the University of Bordeaux I, and Edith Cowan University, Perth, Australia. In 2001, he cofounded Vector 12, a semiconductor intellectual property company. He is currently a Professor with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Canada. His primary research interests are in the area of integrated-circuit (IC) testing, design for testability, and built-in self-test for digital, analog, and mixed-signal circuits and systems-on-chip (SoCs). He has published widely in these areas and holds several patents in IC design and test. Besides testing, he has interests in the design and design methodologies of large and complex ICs and SoCs. He has published over 100 papers in conference and journals and is the holder of four U.S. patents.

Dr. Ivanov has served and continues to serve on numerous national and international steering, program, and/or organization committees in various capacities. Recently, he was the Program Chair of the 2002 Very Large Scale Integration Test Symposium (VTS'02) and the General Chair for VTS'03 and VTS'04. He serves on the Editorial Board of the IEEE *Design and Test Magazine and Kluwer's Journal of Electronic Testing: Theory and Applications*. He is currently the Chair of the IEEE Computer Society's Test Technology Technical Council. He is a Golden Core member of the IEEE Computer Society, a Fellow of the British Columbia Advanced Systems Institute, and a Professional Engineer of British Columbia.

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:20:40 UTC from IEEE Xplore. Restrictions apply.

Resve Saleh (M'79–SM'03–F'06) received the B.S. degree in electrical engineering from Carleton University, Ottawa, ON, Canada, and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Berkeley.

He spent nine years as a Professor with the Department of Electrical and Computer Engineering, University of Illinois, Urbana, and one year teaching at Stanford University. He was with Mitel Corporation, Ottawa, ON, Canada, with Toshiba Corporation, Japan, with Tektronix, Beaverton, OR, and with Nortel, Ottawa, ON. He spent a sabbatical year with PMC-Sierra, Burnaby, BC, Canada. He was a Founder of Simplex Solutions which developed computer-aided design software for deep submicrometer digital-design verification. He is currently a Professor and the NSERC/PMC-Sierra Chairholder with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, in the field of system-on-chip design and test. He coauthored a book entitled *Design and Analysis of Digital Integrated Circuit Design: In deep submicron technology* (McGraw-Hill, 2003). He has published over 100 journal articles and conference papers.

Dr. Saleh is a Professional Engineer of British Columbia. He served as General Chair (1995), Conference Chair (1994), and Technical Program Chair (1993) for the Custom Integrated Circuits Conference. He held the positions of Technical Program Chair, Conference Chair, and Vice General Chair of the International Symposium on Quality in Electronic Design (2001) and has served as Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN. He was the recipient of the Presidential Young Investigator Award from the National Science Foundation in the United States in 1990.

Partha P. Pande (S'04–M'05) received the M.S. degree in computer science from the National University of Singapore, Singapore, in 2002 and the Ph.D. degree in electrical and computer engineering from the University of British Columbia, Vancouver, BC, Canada, in 2005.

He is currently an Assistant Professor with the School of Electrical Engineering and Computer Science, Washington State University, Pullman. His primary research interests are in the area of design and test of networks-on-chip, fault tolerance and reliability of multiprocessor systems-on-chip platforms.

Prof. Partha is serving in the program committees of different international conferences, which include the International On-Line Testing Symposium, Asian Test Symposium, and Midwest Symposium on Circuits and Systems.

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:20:40 UTC from IEEE Xplore. Restrictions apply.