An Initiative towards Open Network-on-Chip Benchmarks

Cristian Grecu[1], Andrè Ivanov[1], Partha Pande[2], Axel Jantsch[3], Erno Salminen[4], Umit Ogras[5], Radu Marculescu[5]

[1]University of British Columbia, [2]Washington State University, [3]Royal Institute of Technology, [4]Tampere University of Technology, [5]Carnegie Melon University

Abstract

Measuring and comparing performance, cost, and other features of advanced communication architectures for complex multi core/multiprocessor systems on chip is a significant challenge which has hardly been addressed so far.

This document outlines the top-level view on a system of benchmarks for Networks on Chip (NoC), which intends to cover a wide spectrum of NoC design aspects, from application modeling to performance evaluation and post-manufacturing test and reliability.

For performance benchmarking it describes requirements and features for application programs, synthetic micro-benchmarks, and abstract benchmark applications. Then, it proposes ways to measure and benchmark reliability, fault tolerance and testability of the on-chip communication fabric. This paper introduces the main concepts and ideas for benchmarking NoCs in a systematic and comparable way. It will be followed up by a report that will define a benchmark framework and the syntax of interfaces for benchmark programs that will allow the community to build-up a benchmark suite.

1. Introduction

This paper motivates the need for a Network-on-Chip benchmarking, outlines the compelling features of a NoC benchmarking environment, and describes an initiative toward establishing some widely accepted and useful benchmarks.

1.1. The Problem

The practical implementation and adoption of the NoC design paradigm faces multiple unresolved issues related to design methodology/technology and analysis of architectures, test strategies and dedicated CAD tools. To advance and accelerate the state of the art of the NoC paradigm R&D, the community is in need of widely available reference benchmarks [1].

Classic benchmarks for multiprocessor systems [2][3] are application-oriented, and cannot be used directly for communication-intensive architecture such as NoCs. Moreover, the nature of the applications running on NoC-based designs is expected to be more varied and heterogeneous compared to typical applications for multiprocessor computers.

The current SoC benchmark circuits, for example ITC 2002 [4], contain only a very limited number of blocks, are targeted at test development only, and do not reflect the high level of integration specific to the NoC scenarios. We are promoting a collaborative initiative to develop a set of NoC benchmarks that will foster improved and accelerated developments in this field.

In view of the propriety issues involved, we propose that the interested community work toward the development of a set of synthetic benchmarks characterized by the following sets of orthogonal parameters:

i) A set of relevant **metrics** and the associated **measurement methodologies**

ii) A set of parameterized **reference inputs** for the NoC benchmarks, consisting of:
   a. NoC functional cores composition (# of Processing Elements - PEs -, number and size of memories, number of I/Os)
   b. interconnect architectures;
   c. data communication requirements;

The term *synthetic* refers to some level of abstraction, for example a task graph with known computation times and communication loads instead of actual application code. Of major importance is the benchmark measurement methodology, which defines parameters of interest and their points of measurement (in time and space) relative to the structure and representation of the NoCs. For coherent, reproducible evaluation and comparison of research data, adequate metrics must be available.

These sets of parameters (a, b, c), when combined, would suffice to yield a meaningful and useful representative set of NoC characteristics. E.g., core composition would characterize the NoC with respect to the number of processing elements, memory elements, I/Os. For testing purposes, additional information would be required, for example, the test strategy (e.g., BIST, scan, etc…) for each functional core and test related parameters (e.g., number of scan chains, scan chains lengths, number of test patterns, number of I/Os, etc.). This type of information is mainly intended to allow the development and comparison of system level test synthesis and scheduling techniques.

The interconnect architecture is intended to characterize NoCs with respect to the data transport capabilities of the communication fabrics. The proposed interconnect architectures to date can be classified into one of the following: cube-based topologies, tree-based topologies, irregular ones, and their different combinations.

Data communication requirements would define the communication needs of the synthetic NoC. This set of parameters consists of inter-core bandwidth/latency, data integrity requirements, and spatial/temporal traffic distributions.

1.2. Benchmarks Characteristics

The problem of NoC design is extremely complex and it can only be solved by identifying and parameterizing the elements that potentially define a NoC, their properties, and their interactions. As such, it is clear that a NoC benchmark has to be applied on more than a simple topological description of a certain circuit (such as in the case of well-known ISCAS85 or ITC 2002 benchmarks), but rather on a combination of hardware blocks (functional IP cores), a NoC fabric (routing elements and wires) with a certain topology/architecture, and a specification for the traffic of data through the entire system. These elements can be thought to be orthogonal, and different benchmark scenarios can be created by their superposition.

A minimal list of properties and features that a NoC benchmarking environment should exercise includes the following:

- Network size (small, medium, large)
- IP core composition (amount of processing, memory cores, other)
- Topology (regular, irregular)
- Traffic characteristics (spatial and temporal)
- QoS requirements (best-effort, guaranteed bandwidth, guaranteed latency)

The different characteristics of the NoC benchmarks with respect to the above properties will allow the evaluation of the NoC performance parameters, among the most important ones being the throughput, latency, power/energy dissipation, and silicon area.

1.3. The Benefits

The NoC benchmarks would envisage benefits similar to those ensuing from our field or related fields. Examples include:

Improved sharing and comparison of R&D results

The NoC design is still in its infancy and, generally, companies and institutions are not open to share specifications, models, and other proprietary data regarding NoCs. A set of academic, synthetic benchmarks can be shared and used without these limitations. The existence of an open format for benchmarks specifications makes possible for interested research groups to contribute with relevant models and test cases.

Increased healthy competitiveness between R&D

The use of different metrics and measurement methodologies complicates comparison. For example, some research groups tend to emphasize just the implementation related issues, such as area or power, whereas the others provide only a set of (anecdotal) performance values.

Increased reproducibility of results and commonality for comparative purposes

Comparison of results is only possible if input data and measurement are fully reproducible. Often, researchers tend to use proprietary test cases, not always fully characterized and therefore not entirely reproducible. Common benchmarks allow fair and consistent comparison of different approaches. Reproduction also requires thorough documentation of measurement settings.

Accelerated development and analysis

Usually, designers build testbenches based on the initial specifications to verify the functionality and performance of complex systems. The existence of standardized input data and hardware models can speed-up the initial design and performance estimation phase.

Better scalability compared to application benchmarks

Application benchmarks offer the best accuracy but are difficult to port for different systems, and their simulation is time intensive, compared to synthetic benchmarks. They also scale poorly with system size, for example the number of tasks, which defines the maximum number of processing elements, is fixed. Synthetic benchmarks are more suitable for benchmarking purposes since they can exhibit properties of particular fixed size application benchmarks, but can scale with system size while still retaining these properties,.

2. Performance Benchmarks

Since many different parameters and factors influence the performance perceived for a particular application, we need a versatile set of devices to analyze, measure and compare the performance of different NoCs. The overall performance for a particular application is the ultimate criteria. However, there are many different factors influencing this figure such as algorithm design, functional partitioning, resource allocation, mapping, communication services, buffering, flow control, routing algorithms, physical design, clocking strategies, and so on. To be able to accurately analyze and assess individual factors we need a more sophisticated set of characteristics.

In the following two types of benchmarks are defined.

Benchmark programs are programs or models resembling real applications, which exercise the entire communication architecture and design methodology. Hence, they can be used to assess the suitability of a particular NoC for a given application or application domain.

Micro-benchmarks are synthetic benchmarks, which exercise one or a few specific aspects of a communication architecture. They can be used to gain insight into a particular design feature or to compare and rate a number of alternative techniques, e.g. several routing algorithms. The most common example is uniform random traffic.

2.1. The Hardware Platform Model

The model of hardware platform consists of a set of Resource Models (RMs) that are connected to the communication network, as shown in Figure 1.

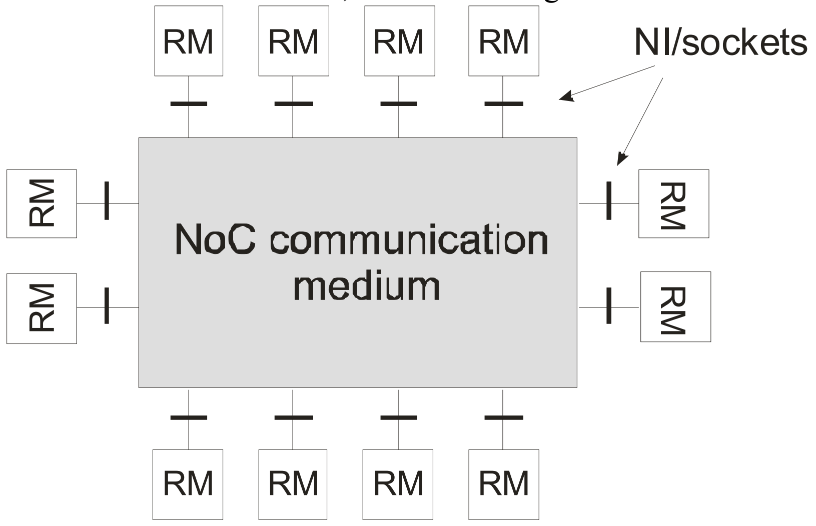

Figure 1: A number of resource models (RMs) connect to the communication network.

A RM is a model written in C, C++, SystemC, VHDL or Verilog. It may be arbitrarily simple or complex. It models the functionality of a resource in the system which communicates with other resources over the network. It may represent a PE (such as processor with software, a dedicated hardware block, a DSP block), an on- or off-chip memory, or any other resource in the system. The main requirement is that it complies with one of the defined interface protocols like OCP [5], AXI [7], or similar (see Sec. 2.6) and that functional tasks may be mapped onto RMs. The functionality may be presented either as benchmark programs or as micro-benchmarks.

2.2. *Benchmark Programs*

Programs are modeled as program written in C, SystemC, VHDL or any other language but without considering the final implementation, be it any combination of HW and SW. For benchmark programs the figure of merit is defined by the benchmark itself because it may be different for different benchmarks. In many cases the end-to-end delay and the overall throughput of the system will be the most important figures of merit. However, some benchmarks may focus on other aspects. For instance, for a benchmark representing a hard real-time system, the most important figure is the number of deadline misses. For most embedded systems the memory size has to be minimized. Consequently, benchmarks that represent those applications will focus also on the amount of buffer space needed at the network interfaces.

A benchmark program provides the following information:

Functionality:

- Application model
- Mapping and scheduling of application tasks to the resource models (RMs)
- Set of RM as C, C++, SystemC, VHDL or Verilog models

Usage:

- Directions to connect the RMs to the network
- Instructions for configuration and compilation
- Deadlines or other non-functional requirements of the application (optional)

Topology and Mapping (optional):

- Size and topology of the network, the structure of the routers
- Binding of a RM to a position in the network

Task description is independent of the underlying hardware (RM and network) and it defines the temporal and spatial properties of the traffic. Tasks are explicitly mapped to resources. In simplest case, there is exactly one task per RM but there can also be multiple tasks per RM. A resource translates the operation count in application model into time units. For example, if a simple resource model can perform 2 operations/cycle, a task with 500 operations takes 500/2=250 cycles to execute on that RM. Given the cycle time, this can be translated to, for example, nanoseconds.

The usage information gives all necessary directions to connect the RMs to the terminals of the network, initialize them, run the benchmark and interpret the results. For instance a benchmark program may require an initialization phase of 1µs followed by an evaluation phase of 5 µs.

The topology and mapping information is optional. However, if it is not provided the evaluation includes also the design methodology and the techniques for network configuration, resource allocation and resource binding. If a benchmark program targets the benchmarking of the network architecture (topology, routing, switching, etc.) it has to provide the topology and mapping information because different mappings may lead to radically different performance results for the same network architecture. If this information is not provided, the benchmark program can be used to evaluate the combined effect of a topology configuration, mapping, and the communication network itself.

2.3. *Micro-Benchmarks*

Micro-benchmarks intend to exercise a NoC in a very specific way or measure a single particular aspect. Hence, they offer insights on a specific property and facilitate the analysis and design of a communication infrastructure. A single micro-benchmark provides only a very limited view and does not allow for far-reaching conclusions about the suitability for an application domain. However, a set of well designed micro-benchmarks can give both a broad and detailed understanding of a given communication network. Since even a large set of micro-benchmarks are not guaranteed to represent a real application well, micro-benchmarks are complementary to benchmark programs. While benchmark programs evaluate the combined effect of many aspects of the platform as well as of the application, micro-benchmarks isolate individual properties and allow for a deeper point analysis.

Micro-benchmarks should cover a number of different aspects which are outlined in the following.

Packets and Transactions:

The packet is the data unit of the network and it has limited size. By measuring delay, bandwidth, jitter, power consumption, etc. of individual packets, the network itself, its routing, switching, buffering and flow control mechanisms are evaluated. Distinction should be made between the latency in the network and the delay for network access.

Transactions are higher level communication activities and evaluate packetization, end-to-end flow control, streaming capabilities and similar services offered by the network. In addition to the network, transactions also evaluate the interface blocks between the network and the resources. Examples of transactions are: memory read and write, read and write bursts, transmission over an open connection or stream, opening and closing of connections. Micro-benchmark defines the transactions which translate into network packets.

Unloaded and loaded case:

In the unloaded case, individual packets or transactions are measured without any network contention. This gives data about minimum delay and peak performance.

The loaded case investigates the network behavior when many independent packets and transactions compete for the same resources. Congestion, arbitration, buffering and flow control policies will be exercised.

Temporal and spatial distribution:

In the loaded case, different traffic scenarios have to be considered. The obvious and most widely used is the random, uniform traffic where nodes communicate with each other with equal probabilities. However, micro-benchmarks need to stress the network with various different traffic scenarios to study how the performance depends on the traffic patterns.

We need to differentiate between temporal and spatial distribution of traffic generation. The temporal distribution determines how an individual RM generates traffic over time. In particular, bursty traffic scenarios have to be covered adequately. The spatial (i.e. target) distribution governs the spatial traffic pattern: who communicates with whom. The benchmarks should cover traffic scenarios with localized traffic, hot-spot patterns and other typical patterns that represent important application characteristics.

Best effort and guaranteed services:

Micro-benchmarks measure the performance of best effort traffic. Guaranteed throughput services are supposed to provide guarantees on minimum throughput and maximum latency, which should be verified by some other means. However, it is important to study the effect of resource allocation by guaranteed services on best effort traffic performance.

Network size:

To study the scalability of the communication networks of various sizes with up to several hundred nodes shall be exercised by scaling the micro-benchmark accordingly.

The set of micro-benchmarks shall systematically exercise all important aspects of a NoC. It will give the NoC developer insight and guidelines for improvement. It will also give the NoC user a detailed understanding of the NoC behavior, its strengths and weaknesses.

In addition to stochastic benchmarks, deterministic micro-benchmarks can be devised to exercise specific corner cases and for measuring and studying special situations in a reproducible way.

2.4. Communication-centric Application Modeling

Synthetic benchmarks and traffic generators are useful for exercising various aspects of the communication network and evaluating different network configurations. However, they must be able to capture the control and data dependencies between the tasks. For example, in an MPEG encoder, the variable length encoding operation can start only after the discrete cosine transform and quantization tasks are completed. Therefore, simple stochastic models for RMs with constant transfer probabilities are likely to result in inaccurate estimates. Capturing the control and data dependencies becomes vital especially, when one targets a specific class of applications. A complete implementation of a real application will obviously capture all the dependencies; however, it may be very complicated and time-consuming. Moreover, the behavior of a task as a traffic source/sink, i.e., its model of communication, as opposed to the details of the computation it performs, is sufficient for the evaluation of the communication architecture. For this reason, we propose to utilize Finite State Machines (FSM's) that mimic the tasks of a real application in terms of communication.

Overview

In the context of embedded systems, applications are usually modeled as communication task graphs (CTG) [6]. The tasks in the CTG exhibit both control and data dependencies.

The control dependencies imply that one task cannot be executed before its predecessor tasks are completed, while the data dependencies indicate communication between two tasks. Furthermore, CTGs are widely accepted and there are publicly available benchmarks [3][6]. Therefore, we utilize CTGs as an entry point to the proposed methodology.

Basically, each PE in the network will be an FSM generated automatically from a CTG and mapping information. The PE will be connected to a router via bidirectional links and comply with common interface protocols. Instead of generating a packet randomly, the PEs will follow the rules specified by the underlying CTG, as explained below.

Data generation/consumption

Each PE is a Finite State Machine with the following information:

Task list (i.e. mapping):
The tasks that are mapped to corresponding PE. This provides the information regarding the *sender list* (the list of PE that can send data to the current PE) and the *destination list* (the list of the PEs to which the current PE sends data).

Control information:
The information regarding the data dependencies. For example: *"If* enough data is received from source *i*, *then* initiate a transaction with destination *j*". For detailed modeling, the control information includes also the internal state of the task, for example its execution count so far.

Processing time (P):
The amount of time it takes for the PE to complete a certain task. After receiving enough data, the PE will execute P operations (modeled as waiting for certain period), before generating a response.

Transaction data amount (D):
The size of the transaction that will be generated after processing.

Based on the control and data dependencies specified by the underlying CTG, there may be a number of different PE types:

SISD (Single input, single destination):
This is the simplest case. The PE may receive data only from one PE. After processing the input, the resulting transaction is sent only to a single destination.

MISD (*Multi*-input, *single* destination):
In this case, more than one PE may send data to this PE. This situation corresponds to a scenario, where inputs from multiple sources are needed to perform a certain type of task. After the task is completed, the response is sent to a single destination. One can further divide MISD into two classes: dependent or independent inputs. For the case with dependent inputs, all the inputs have to be received before the processing can take off, while for the independent inputs a subset of inputs is sufficient for processing.

SIMD (Single input, multi destination):
This type of PE receives input from a single source, but the packet generated after receiving this input is sent to multiple destinations.

MIMD (Multi-input, multi destination):
This type of PE reflects the most general case and the others are simply sub-cases of MIMD. The PE can both receive from and send to multiple PEs. The inputs can be either dependent or independent as in MISD case.

Overall operation

The PEs (their FSMs) describing the target application are automatically generated from the CTG of the application, available RMs, and applied mapping. Then, the PEs and routers are connected to each other as specified by the network topology. The PE that implements the root task in the CTG initiates the application by injecting the input data to the network. The input PE repeats this process at a period specified by the CTG.

The operation of a sample PE is depicted in Figure 2. All PEs, except the one that implements the root task, are initially in the waiting state (State 1 in Figure 2). As soon as all necessary data are received, the PE enters the processing state (State 2 in Figure 2). The PE stays in this state until the processing is completed. This duration is determined by the type of the task (i.e. the underlying task graph) and the type of the processing element. This state is followed by State 3, where the PE generated the output data and sends it to the PEs in the destination list. After this, the PE enters again the waiting state.

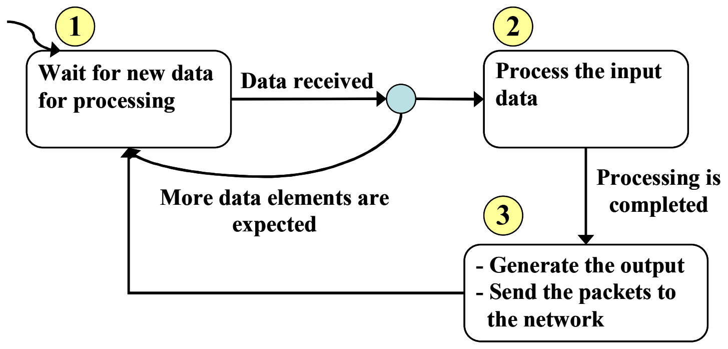
1
Wait for new data for processing
Data received
2
Process the input data
Processing is completed
More data elements are expected
3
- Generate the output
- Send the packets to the network

Figure 2: The finite state machine describing a sample processing element.

To summarize, the PEs mimic the real application as the generated data traffic matches the actual implementation of the application. As a result, the following performance measures can be obtained, and different network configurations can be evaluated accurately.

Application run-time:

The time it takes to complete the application, i.e. time to process a certain amount of input data. We note that this metric can be reliably computed only when the data and control dependencies within the application are captured as in the proposed method. This method also gives insight to utilization level of the PEs and how well the deadlines are met.

Application throughput:

The minimum period at which the input PE injects data determines the maximum application throughput. If the input PE injects data at a faster rate, the bottleneck PE among the downstream nodes cannot accept it at the same pace and the data will be blocked after the available buffering resources are exhausted.

2.5. *Measurement Points*

In addition to defining how to exercise a network, it is also important to unambiguously define where to measure the parameters of interest.

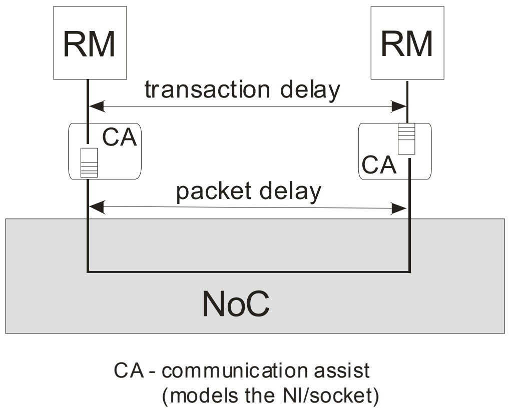

Figure 3: Points of measurement.

Figure 3 shows two different delays that have to be distinguished. The packet delay denotes the delay of a packet between the moment its header is injected into the network and the moment when the last flit leaves the network. In the communication assist (CA) usually some buffering is done when the packet cannot enter the network immediately. Also, for transactions and longer messages the CA performs packetization, de-packetization, reordering of received packets, and connection management. All packet-related measurements shall be done between the CA and the network fabric, while all transaction-related measurements shall be done between CA and the RM. In addition to average values, both minimum and maximum latencies are measured at the same time. In certain cases, the variation in latency (i.e. jitter) is also important.

2.6. Interfaces and Sockets

To make a benchmark reusable for analyzing different networks, both the benchmark and the network have to adhere to a standard interface definition. Thus, the benchmark initiative will define the interfaces and protocols that networks under test and benchmark programs have to adhere to. Similarly, the file format for delivering the benchmarks will be defined.

Protocols such as OCP serve this purpose very well, and their use simplifies not only the benchmarking, but also the design of NoC-based systems. The OCP protocol is core-centric and interconnect agnostic, which allows the network interfaces (NI) to actually deliver to the NoC fabric a set of standard communication signals (defined by the OCP socket) organized in packets and enhanced with the set of data that is specific to the particular network protocol.

The OCP-IP socket is openly available, popular, and has sufficient high level and advanced concepts, such as multiple open, pipelined and out-of-order transactions. Therefore, it seems suitable for interfacing with NoCs. All benchmark programs or micro-benchmarks accepted in the NoC benchmark suite shall adhere to the OCP-IP protocol (or similar).

The corresponding benchmark program will thus measure the performance of NIs from the OCP socket to the point where packetized data is injected into the NoC interconnect. This allows the benchmark to be independent of the functional cores and concentrate on assessing the communication performance and parameters of the NoCs.

3. Benchmarks for NoC Test and Reliability

One of the most important requirements for the NoC design methodology to be widely adopted is to be 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).

To date, the NoC test methodologies are mainly concerned with the testing of the functional IP cores, using the communication infrastructure as a Test Access Mechanism (TAM)[8]. The missing link for a complete test solution is the test strategy for the interconnect infrastructure itself. The common characteristics of NoC interconnect architectures is that the functional blocks communicate with each other through intelligent switches and links [10][11]. Hence the (post-manufacturing) test strategies of NoC-based interconnect infrastructures need to address two problems:

- testing of the switch blocks;
- testing of the inter-switch interconnect segments.

The NoC interconnects are characterized by poor controllability and observability, due to the fact that they are deeply embedded and spread across the chip. Pin-count limitations restrict the use of I/O pins dedicated for the test of the different components of the data-transport medium. The NoC fabric is a mix of active and passive components that are exposed to a multitude of faults of very diverse nature. The NoC switches contain both memory elements and logic blocks, for which the fault mechanisms and models can be significantly different. However, they need to be integrated in a consistent, streamlined manner. On the other hand, the inter-switch links are sensitive to interconnect-specific faults such as opens, shorts, delay-faults, and crosstalk faults.

A distinct direction for benchmarking NoCs is reliability benchmarking. In a broader sense, the reliability issues encompass aspects such as tolerance to post-manufacturing faults and transient errors, tolerance to process, voltage, and temperature (PVT) variations, resilience to transient errors [12].

3.1. Test Access Mechanisms (TAM)

The problem of test access mechanism design for NoCs can be formally described as follows: given a set of functional cores and an on-chip communication fabric, design a hardware mechanism that transports test input data from an on- or off-chip source to the functional cores and the to the communication infrastructure, and delivers the test output data from the cores and NoC to an on- or off-chip test sink. This problem is somewhat similar to the more general TAM design for SoC core-based test; however, the existence of a complex, distributed data transport mechanism (the NoC infrastructure), raises two issues.

First, how this infrastructure can be tested, and second, what is the relationship between the NoC infrastructure and the TAM infrastructure. The latter question appears when one considers that the NoCs are particular forms of SoCs, for which test methodologies (and cored-based test in particular) are mature enough to provide a safe, if not fully optimized, solution. Consequently, a NoC could be considered as a classic SoC, and tested using the core-based approach [11], with the communication fabric tested as a separate core. On the other hand, since a NoC will already possess a dedicated data transport mechanism, it makes sense to reuse that for purpose of test data transport. While this is not the place to discuss one solution in particular, it is obvious that a test benchmark has to be transparent to the particular TAM used by a target NoC.

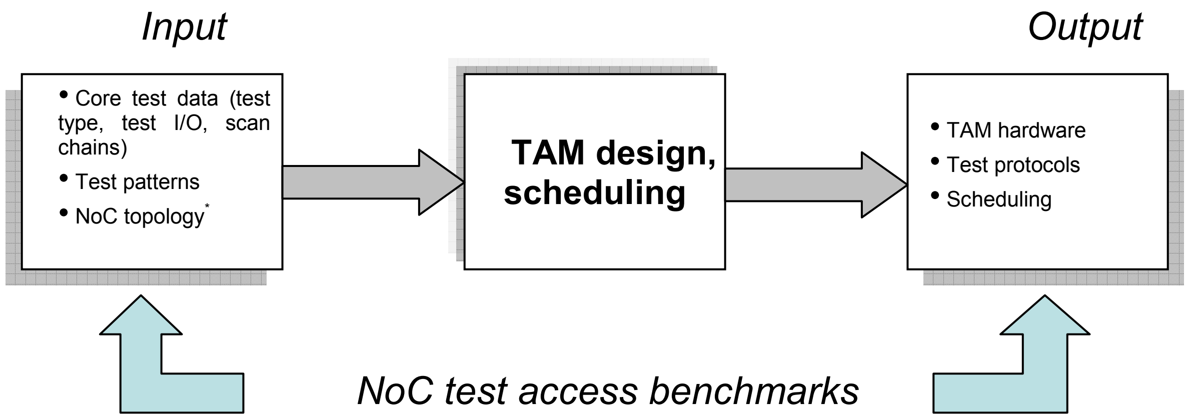

Figure 4: Benchmark data for NoC TAM design (* - optional)

With these considerations, the overall benchmark approach can be represented as in Figure 4, where the data required as input for the benchmark consists of:

- test information for each functional core (test type - scan, BIST -, number of test I/Os, number and configuration of scan chains);
- test patterns for the functional cores and the NoC infrastructure;
- optional, information on the topology of the NoC architecture.

3.2. Test Scheduling

The goal of manufacturing test is to ensure that the NoC is fabricated correctly, with respect to a specified set of faults. To achieve this, for each core, a set of test patterns are generated (on- and/or off-chip) and applied to the cores' test inputs. Subsequently, test output data is collected from the test outputs and transported to the test sink for comparison with the expected outputs. For large numbers of cores, this activity can become extremely time-intensive, and this can raise the cost associated to manufacturing test to prohibitive amounts. Therefore, test scheduling has the objective of minimizing the test time, and implicitly the total test cost. Additional constraints that must be considered here are test power and test area. The power dissipated during testing must be carefully estimated, such that the power budget of the NoC is not exceeded and the test area overhead is within acceptable limits.

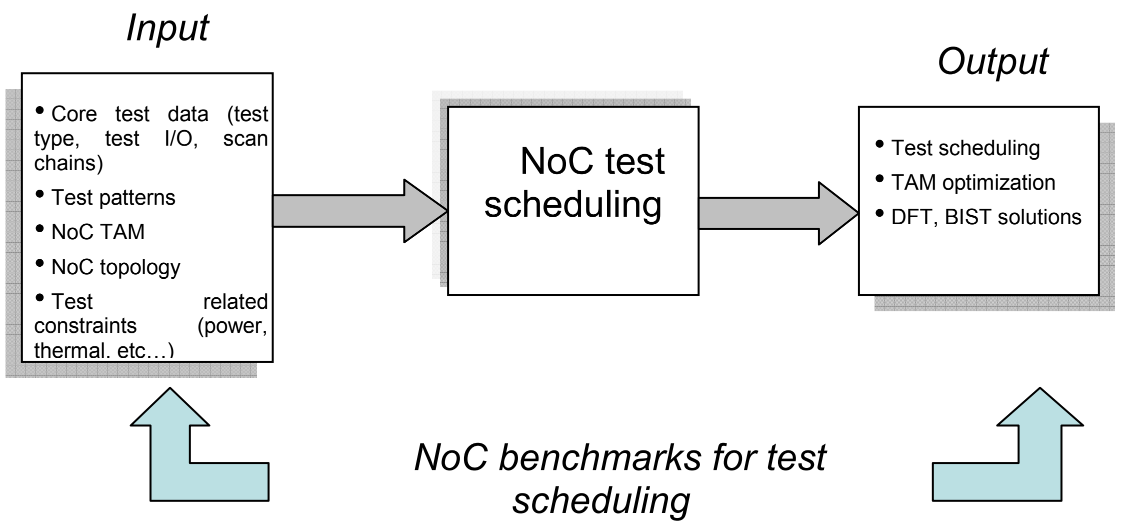
Input
• Core test data (test type, test I/O, scan chains)
• Test patterns
• NoC TAM
• NoC topology
• Test related constraints (power, thermal, etc…)
NoC test scheduling
Output
• Test scheduling
• TAM optimization
• DFT, BIST solutions
NoC benchmarks for test scheduling

Figure 5: Benchmark data for NoC test scheduling

Figure 5 illustrates the inputs needed to benchmark the NoC test scheduling. The data that must be delivered as input contains each functional core's test information (test type, number of test I/Os, nature and length of scan chains), individual test patterns for each core and for the NoC communication infrastructure, NoC topology, and other constraints such as test power budget, thermal characteristics, etc.

3.3. On-line Testing

Transistor sizes used in current fabrication technologies and increased levels of integration expose the NoCs to a multitude of transient faults during their life-time. Among the most significant causes we can enumerate EM noise, cosmic radiations, PVT variations, and many others. Moreover, many of the possible NoC applications are in fields like communications, avionics, defense, where reliable and safe operation of the devices is one of the most important design parameters. In order to monitor continuously their operation and ensure that malfunctions can be detected and compensated for, it is a common practice to perform some amount of testing on-line, without disrupting the operation of the devices. Different NoC designs may be more or less suited for on-line testing, and specific techniques must be developed to perform on-line testing in NoCs. Therefore, an important component of the benchmarking must be evaluating the ability of a NoC to be tested on-line.

3.4. Benchmarks for Fault Tolerance and Reliability

NoCs are particularly suited for implementation of fault-tolerant techniques, due to their inherent parallelism and potential for reconfigurability. Defect/fault tolerant techniques can be implemented at different levels, from hardware redundancy to software-based error recovery schemes. Adaptive routing algorithms combined with error control mechanisms show great promise in achieving fault-tolerant on-chip communication. However, the impact on NoC power and performance can be prohibitive, since extra-hardware or traffic management schemes required for correcting faults and re-ordering of the packets will generally tax the power budget. A meaningful benchmark for

NoC reliability must provide QoS- and power-constrained application data and assess the impact of particular fault-tolerance mechanisms on NoC implementations.

A critical requirement for determining the efficiency of different fault tolerant designs is the availability of relevant, quantitative metrics. A fault tolerant NoC must be able to recover from failures of the data transport mechanism. Fault recovery performance refers to the time required to detect and recover from a NoC fault (e.g. a crosstalk fault on an interconnect link, a failed memory buffer, etc.). If the maximum time to perform a fault recovery can be bounded while still meeting the system performance requirements, then the network fault recovery mechanism can be used successfully. The fault tolerance metrics must be independent of specific hardware features or NoC architectures. They should allow NoC fault recovery performance to be assessed from an application point of view.

A comprehensive fault tolerant approach consists of five key elements: *avoidance, detection, containment, isolation, and recovery*. They may be adopted individually or as hierarchical combinations. The effectiveness of the corresponding implementations can be estimated using quantitative metrics generally accepted for distributed communication systems.

We define a set of fault tolerance metrics to be used for assessing the reliability of a NoC subsystem:

- Detection latency (T_{det}): the amount of time (in cycles) between the moment a fault appears and the moment it is detected.
- Recovery time (T_{rec}): the amount of time (in cycles) that passes between the detection and recovery of a fault.
- Availability: the ratio between the amount of time the NoC subsystem is fully functional, and the total operating time, including detection and recovery latencies.

The three parameters defined above have direct impact on global performance figures of merit of the NoC, specifically in terms of QoS: T_{det}, T_{rec} define the lower limit of the achievable latency, and therefore the level of QoS that can be guaranteed upon occurrence of faults. These metrics complement the application metrics (latency, throughput) and resource metrics (area, power consumption) measured in presence of detection/recovery techniques.

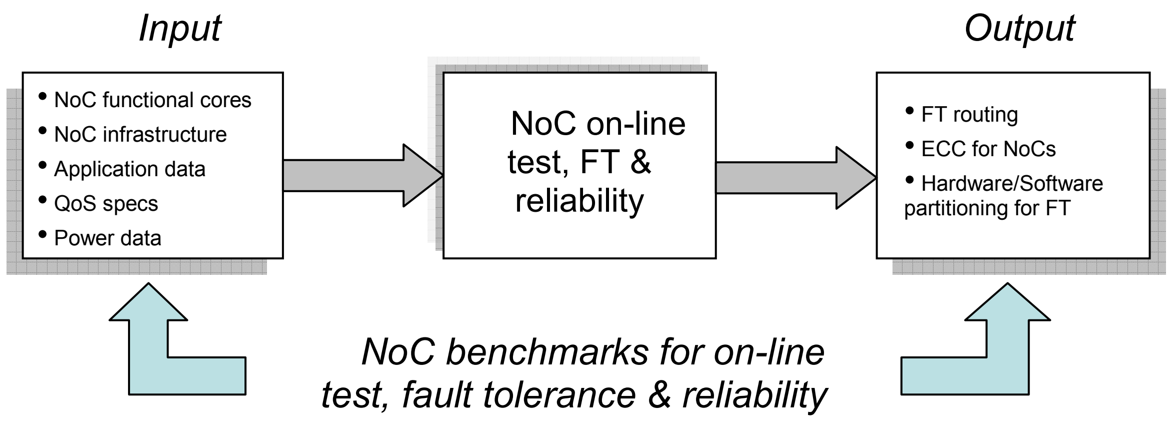
Input
Output
• NoC functional cores
• NoC infrastructure
• Application data
• QoS specs
• Power data
NoC on-line test, FT & reliability
• FT routing
• ECC for NoCs
• Hardware/Software partitioning for FT
NoC benchmarks for on-line test, fault tolerance & reliability

Figure 6: NoC benchmarking for on-line test, fault tolerance and reliability.

4. Future Directions

In order to provide a consistent and reliable means for results sharing and comparison, the NoC benchmarks must be provided in a format that is simple, flexible, non-ambiguous, and allows for future improvements. Additionally, the IP-sensitive nature of such benchmarks when reflecting commercial designs must be protected, so that the benchmarks can remain open to the academic/industrial communities. Another requirement for the benchmarks and their format is modularity: according to their place in the NoC design/test flow, some benchmarks may be the output of a design/test step whose input is a different benchmark. When developing the benchmarking methodology and formats, one should consider how different benchmarks could possibly interact with each other.

The input formats and detailed benchmarking methodologies are the object of a second document that the NoC Benchmarking Workgroup will make available. Interested parties wishing to provide feedback or contribute with specifications and models are invited to contact the group members.

Acknowledgements

The authors would like to thank OCP-IP for its continued support and contribution. The authors also thank other workgroup members and industry partners for the fruitful discussion and their useful comments during this initiative.

5. References

[1] E. Salminen, T. Kangas, T. D. Hämäläinen, J. Riihimäki, "Requirements for Network-on-Chip Benchmarking", Norchip, Oulu, Finland, November 21-22, 2005, pp. 82-85.

[2] The Standard Performance Evaluation Corporation, SPEC, *http://www.spec.org/hpg/*

[3] R. Dick, Embedded System Synthesis Benchmarks Suites (E3S) *http://www.ece.northwestern.edu/~dickrp/e3s/*

[4] ITC'02 SOC Test Benchmarks, http://www.hitech-projects.com/itc02socbenchm/

[5] Open Core Protocol Specification, Release 2.1, OCP-IP, 2005, *http://www.ocpip.org/socket/ocpspec/*

[6] R. P. Dick, D. L. Rhodes and W. Wolf, "TGFF: task graphs for free," Proc. Intl. Workshop on Hardware/Software Codesign, March 1998.

[7] AMBA AXI Protocol Version 1.0 Specification, ARM Limited, 2004.

[8] Y. Zorian, E. J. Marinissen, S. Dey, "Testing Embedded Core-Based System Chips," IEEE International Test Conference (ITC'98), pp. 130-143, 1998.

[9] A. Jantsch and H. Tenhunen, editors, *Networks on Chip*, Kluwer Academic Publishers, 2003.

[10] P. P. Pande, C. Grecu, A. Ivanov, R. Saleh, G. De Micheli, "Design, Synthesis, and Test of Networks on Chips," IEEE Design and Test of Computers, vol. 22, no. 5, pp. 404-413, Sept/Oct, 2005.

[11] Y. Zorian, D. Gizopoulos, C. Vandenberg, P. Magarshack, "Guest Editors' Introduction: Design for Yield and Reliability," IEEE Design and Test of Computers, vol. 21, no. 3, pp. 177-182, May/Jun, 2004.

[12] V. lyengar, K. Chakrabarty, E.J. Marinissen, “Test Access Mechanism Optimization, Test Scheduling, and Tester Data Volume Reduction for System-on-chip” IEEE Transactions on Computers, vol. 52, issue 12, Dec 2003, pp:1619 – 1632.