An Event-Based Monitoring Service for Networks on Chip

CALIN CIORDAS and TWAN BASTEN
Eindhoven University of Technology
and
ANDREI RĂDULESCU, KEES GOOSSENS, and JEF VAN MEERBERGEN
Philips Research

Networks on chip (NoCs) are a scalable interconnect solution for multiprocessor systems on chip. We propose a generic reconfigurable online event-based NoC monitoring service, based on hardware probes attached to NoC components, offering run-time observability of NoC behavior and supporting system-level debugging. We present a probe architecture, its programming model, traffic management strategies, and a cost analysis. We prove feasibility via a prototype implementation for the Æthereal NoC. Two MPEG NoC examples show that the monitoring service area, without advanced optimizations, is 17–24% of the NoC area. Two realistic monitoring examples show that monitoring traffic is several orders of magnitude lower than the 2GB/s/link raw bandwidth.

Categories and Subject Descriptors: B.4.3 [**Input/Output and Data Communications**]: Interconnections (Subsystems); C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors)—*Interconnection architectures (e.g., common bus, multiport memory, crossbar switch)*; C.5.4 [**Computer System Implementation**]: VLSI Systems

General Terms: Design

Additional Key Words and Phrases: Networks-on-Chip, monitoring, debugging

1. INTRODUCTION

1.1 Problem Statement

Due to the ever-increasing miniaturization of transistors, very complex chip designs are becoming possible. For both physical and complexity reasons, future

Authors' addresses: C. Ciordas and T. Basten, Design Methodology for Electronic Systems Group, Eindhoven University of Technology, P.O. Box 513, 5600MB Eindhoven, The Netherlands; A. Rădulescu, K. Goossens, and J. van Meerbergen, Philips Research Laboratories, Prof. Holstlaan 4, 5656AA Eindhoven, The Netherlands; Correspondence; email: c.ciordas@tue.nl.
Based on "An Event-Based Network-on-Chip Monitoring Service" by Calin Ciordas, Twan Basten, Andrei Rădulescu, Kees Goossens and Jef van Meerbergen, which appeared in the *Proceedings of the 2004 IEEE High Level Design Validation and Test Workshop (HLDVT 2004).* © 2004 IEEE.

chip designs will inherently be multiprocessor systems, consisting of hundreds of IPs communicating with each other.

The design of those large-scale chips needs to be structured. Advanced interconnects such as NoCs [Benini and De Micheli 2002; Dally and Towles 2001; Goossens et al. 2002; Guerrier and Greiner 2000; Millberg et al. 2004; Karim et al. 2002] have been proposed in this trend. They help decouple computation from communication and offer well-defined interfaces [Benini and De Micheli 2002; Rijpkema et al. 2003; Goossens et al. 2003], enabling IP reuse, and structuring the design process. NoCs' scalability [Rijpkema et al. 2003; Guerrier and Greiner 2000] makes them the preferred interconnect solution for tomorrow's large-scale chips.

The increasing complexity of next generations of chips implies the need for debugging support. The differences between NoC-based and traditional interconnect solutions require the development of novel debugging techniques. Observability is a key issue in embedded systems debugging. Without proper computation and communication observability of the system, the task of debugging is impossible. The basic requirement for debugging is the existence of a monitoring system: is it possible to observe the internals of a system while it is running? This article, while recognizing the importance of the debugging task, focuses on the monitoring task.

1.2 Related Work

Computation observability received a lot of attention in the literature. Philips's real-time observability solution, SPY [Vermeulen et al. 2001], allows the non-intrusive output of internal signals on 12 chip pins. The internal signals are grouped in sets of 12 signals and are hierarchically multiplexed on the 12 pins. The observed signals are a design-time choice.

ARM's embedded trace macrocell (ETM) [ARM 2002] offers real-time information about core internals. It is capable of non-intrusively tracing instructions and data accesses at core speed. The trace is made available off-chip by means of a trace port up to 16 bits.

The NEXUS standard [IEEE-ISTO 5001TM 2003] proposes a debug interface for real-time observability of standard defined features. It proposes a standard port while letting the implementation details to the users.

Whereas computation observability in the context of multiprocessor SoCs with deeply embedded cores has been studied and many solutions have been proposed, the on-chip *communication observability* has been mostly ignored. This is because computation observability covers most of bus-based chips. With the emerging trend of NoC-based SoCs, the on-chip communication becomes more sophisticated, relying on run-time programmable solutions. Programmable NoC solutions can be error prone, for example, introducing congestion points or deadlock, leading to a need for debugging. In NoCs, there is no central point of communication/arbitration, like in busses; multiple truly parallel communication paths exist, adding to the overall complexity. In light of the trend towards NoCs, we need communication observability.

The test and verification implications of using NoCs have been inventoried in Vermeulen et al. [2003]. However, today, in the research community, focus is on the design [Bolotin et al. 2004; Kumar et al. 2002; Dally and Towles 2001; Guerrier and Greiner 2000; Millberg et al. 2004; Goossens et al. 2005], analysis [González Pestana et al. 2004; Poplavko et al. 2003] and use [Goossens et al. 2004; Murali and De Micheli 2004] of NoCs. Currently, there is no support for communication observability in NoC-based SoCs. Today, to the best of our knowledge, there are no NoC monitoring systems.

This article builds on an earlier conference article [Ciordas et al. 2004]; it presents the programming model of the monitoring service in more detail, and provides more information on the implementation, including cost estimations.

1.3 Contribution

This article proposes a NoC run-time monitoring system for NoC-based SoCs to serve as a prerequisite for debug. The main monitoring requirements are scalability, flexibility, nonintrusiveness, real-time capabilities and cost. Our solution to the run-time monitoring problem for NoC-based SoCs is a generic NoC monitoring service. This generic NoC monitoring service can be instantiated for any existing NoC. The NoC monitoring service can be configured from any NoC master network interface. The NoC service is based on hardware probes attached to network components, that is, routers and network interfaces. We assume that a parametrized model for NoC components is available, such that the number of ports of these components can easily be instantiated. The service allows:

(1) The nonintrusive capturing of run-time information and functional data in the NoC-based SoCs.
(2) The event-based modeling of monitoring information to be generated in the hardware probes. Events and their attributes are run-time configurable. This approach allows on-chip abstraction of data to reduce the cost of transporting the information.
(3) On-the-fly setup, control and use of the monitoring architecture. The NoC itself is used for all these purposes. No separate communication infrastructure is needed. Configuration and monitoring traffic is managed at run-time.

We prove the concept via an implementation for the Æthereal NoC [Goossens et al. 2002; Rădulescu and Goossens 2004; Rijpkema et al. 2003]. We make an inventory of the costs involved. We present two different MPEG NoC examples and we show that the monitoring system can be realized within 17–24% of the NoC area. By NoC area, we mean the interconnect network area only, not including the area of IPs or other parts of the chip. Based on the area cost for a commercially available monitoring solution for ARM embedded cores, we believe a target of 15–20% area cost is acceptable. Our initial implementations suggest that this target is realistic. Furthermore, we show that, depending on the desired amount of information detail, the additional traffic introduced by the monitoring service can be several orders of magnitude lower than the usual

NoC bandwidth, as exemplified by a NoC-based SoC reconfiguration example and a connection sniffing example.

1.4 Overview

The rest of the article is organized as follows. Section 2 presents the Æthereal NoC concepts. The general concepts of our NoC monitoring service are explained in Section 3. Our event model, a generic NoC event taxonomy and an instance of it applied to the Æthereal NoC are presented in Section 4. In Section 5, the generic concepts of the monitoring probe architecture and the Æthereal probe details are explained. The corresponding programming model is presented in Section 6. Section 7 explains several traffic management options for monitoring, covering configuration traffic as well as data traffic. An inventory of all costs involved is presented in Section 8, together with two MPEG NoC examples to study the area cost of the monitoring system on the NoC, and two examples to quantify possible additional traffic introduced by the NoC monitoring service. We end with conclusions and a discussion of future work.

2. ÆTHEREAL NOC

Several NoCs [Benini and De Micheli 2002; Dally and Towles 2001; Goossens et al. 2002; Guerrier and Greiner 2000; Millberg et al. 2004; Karim et al. 2002] have been proposed, using different topologies and routing strategies. NoCs are generally composed of network interfaces (NIs) [Rădulescu et al. 2005] and routers (R) [Rijpkema et al. 2003]. NIs implement the NoC interface to IP modules. Routers transport data from NI to NI.

The Æthereal NoC [Goossens et al. 2002; Rădulescu et al. 2005; Rijpkema et al. 2003] runs at 500 MHz and offers a raw bandwidth of 2GB/s per link in a 0.13 micron CMOS technology. For Æthereal, the topology can be selected arbitrarily by the designer.

The Æthereal NoC provides transport layer services to IPs in the form of connections, for example, point-to-point connections or multicast, which can be either guaranteed throughput (GT) or best effort (BE). Guarantees are provided by means of slot reservations in routing tables of routers and NIs. Slot reservations are programmed at run-time. Connections have properties such as data integrity, transaction ordering or flow control. Data integrity guarantees that the data is not altered in the NoC. Transaction ordering guarantees that the order of separate communications is preserved per connection. Connection flow control guarantees that the data that is sent will fit in the buffers at the receiving end, to prevent data loss and network congestion.

3. NOC MONITORING SERVICE

3.1 Monitoring Service Concepts

The monitoring service is offered by the NoC itself, in addition to the communication services [Rădulescu and Goossens 2004] offered to IPs. The NoC monitoring service, illustrated in Figures 1 and 2, consists of configurable monitoring probes (P) attached to NoC components, that is, routers or NIs, their

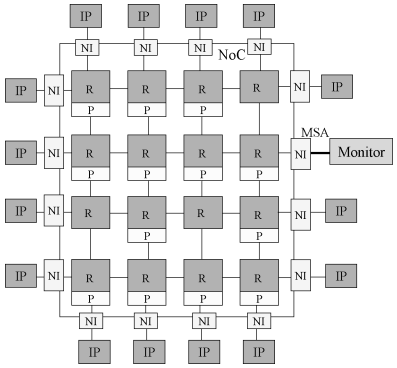

Fig. 1. Centralized monitoring service.

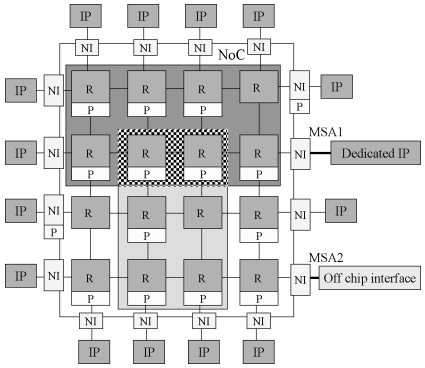

Fig. 2. Distributed monitoring service.

associated programming model, and a monitoring traffic management strategy. Please note that we have chosen to add probes to routers and NIs, because this gives access to the internals of these components. As an alternative one could consider attaching probes to links but limiting observability to the information passing the links.

Event Model. All the monitored information is modeled in the form of events. An event model specifies the event format, for example, timestamped events or not. Currently, we focus only on timestamped events. An event taxonomy helps to distinguish different classes of events and to present their meaning.

Monitoring Probes. The monitoring probes are responsible for collecting the required information from NoC components. The probes, Ps in Figures 1 and 2, capture the monitored information in the form of timestamped events. Multiple classes of events can be generated by each probe, based on a predefined instance of an event model. Monitoring probes are not necessarily attached to all NoC components. For example, the top-right router in Figure 1 has no probe attached. The placement of probes is a design-time choice and is related to the cost versus observability trade off.

Programming Model. The programming model describes the way in which the monitoring service is being set up or torn down. It consists of a sequence of steps for configuring the probes and the means of implementing those steps. Probes are programmed via the NoC, for example, using memory-mapped I/O [Rădulescu et al. 2005; Philips Semiconductors 2002]. The monitoring service can be configured at run-time, by any master IP connected to the NoC from the corresponding NI, called the *monitoring service access point* (*MSA*), for example, MSA1 and MSA2 in Figure 2.

Traffic Management. Traffic management regulates the traffic from the MSA to the probes, required to configure the probes, and the traffic from the probes to the MSA, required to get the monitoring information out of the NoC. Already available NoC communication services or dedicated solutions, for example, a separate bus, can be used for the traffic management for monitoring.

3.2 Distributed vs. Centralized NoC Monitoring Service

We propose a monitoring service that can be configured as a distributed or a centralized service, during run-time, at arbitrary moments in time. In a centralized monitoring service, as shown in Figure 1, the monitoring information from the selected probes is collected in a central point, in this case a monitor, through a single MSA. For small NoCs, a centralized monitoring service is possible and convenient. However, the convergence of monitoring data to a central point may become a bottleneck in large NoCs.

In a distributed monitoring service, the monitoring information is collected for different subsets of NoC components at different points through multiple MSAs. In this way, bottlenecks are removed and we achieve scalability.

Figure 2 shows a distributed monitoring service composed of two subsets of components. One connects directly to a dedicated monitoring IP through MSA1. The second connects, indirectly through a router that is not part of the subset, to an off-chip interface through MSA2. The subsets can be programmed at run-time offering increased flexibility. Hence, a probe can be part of one subset at one time and of a different subset at other times; see the probes attached to the routers in the middle of the figure. Monitoring information can be either used on-chip, for example, by the dedicated IP in Figure 2, or it can be sent off-chip either directly through an off-chip link or via a memory.

4. EVENT MODEL

4.1 Events

An event [Mansouri-Samani and Sloman 1996] is a happening of interest, which occurs instantaneously at a certain time. In our view, an event is a tuple:

Event = (identifier, timestamp, producer, attributes).

The mandatory event *identifier* identifies events belonging to a certain class of events and is unique for each class. The *timestamp* defines the time at which the producer generates the event. The *producer* is the entity that generates the event. *Attributes* are the useful payload of events. Each attribute is present in the form:

Attribute = (attribute identifier, value).

An attribute consists of its *attribute identifier* and its *value*. The attributes and the number of attributes may depend on the event type.

4.2 NoC Event Taxonomy

In the following, we present a taxonomy of NoC events. The term user is used to identify an IP. In general, we can group NoC events in five main classes: user configuration events, user data events, NoC configuration events, NoC alert events, and monitoring service internal events.

The criteria used for this taxonomy are graphically presented in Figure 3. Within the NoC, we can have either user traffic or NoC internal traffic. Furthermore, there can be control traffic or regular data traffic. These two dimensions

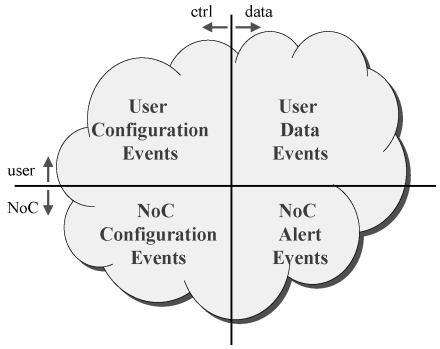

Fig. 3. NoC event space.

define four of the five mentioned classes. All these classes represent data of the monitored system. The fifth class of events is the class of monitoring service internal events. Whether data is generated by the monitoring service or by the monitored system can be seen as a third dimension. The resulting taxonomy covers all relevant groups of events and is general enough to be valid for different types of NoCs, although event types may need to be redefined for each specific NoC. The reason to choose these classes is that each of them may be useful in different debugging tasks. Also, other dimensions can be added to this taxonomy in order to refine it. One such dimension could be the grouping of events according to the component where they are generated, for example, in the probes attached to routers or NIs.

User Configuration Events. A NoC is used by IPs to communicate to each other. User configuration events expose this communication, presenting a system-level view of it, hiding NoC specific details. User configuration events can show for example that processor X is communicating with processor Y via a connection. These events can be very detailed, for example exposing the properties of the connection, or very abstract showing just the timing of the communication and the communicating parties. This class of events may be useful to check the system level interaction of components.

User Data Events. This class of events is the class that allows the sniffing or spying of functional data from the NoC. The sniffing itself can be from the NoC elements or from the links. Sniffing may be required for example to check whether the transmitted data, such as a memory address, is exactly the intended data. It can allow for example sniffing of flits, network packets or complete messages depending on the NoC and the purpose sniffing is used for. This class of events may be useful to check data details of the specific interaction of components.

NoC Configuration Events. To achieve interprocessor communication, the NoC must be programmed or configured, statically or dynamically, in a centralized or distributed way. NoC configuration events expose this configuration of the network, enabling the system debugger to trace configurations, allowing it, for example, to observe the setup of a specific connection. Examples of such

events are the fact that a new entry in a routing table has been completed or that the routing table is full. Attributes of these events can be for example the party that wrote the routing table entry. These events are particularly useful for NoC debugging and optimizations.

NoC Alert Events. After programming, network problems can arise. Problems like buffer overflow, congestion, starvation, livelock or deadlock can appear. In real-time systems, missing a hard real-time deadline is a serious error. Therefore, it is imperative to monitor the network behavior and be alerted if signals of overload or misbehavior appear. An example of an event fitting into this class is the continuous lack of progress in a nonempty router queue indicating that the NoC might be congested or even deadlocked. An attribute of such an event can be the number of cycles the specific queue was idle. Performance analysis events, like average queue fillings or other metrics, can be included here also. Performance debugging may typically need this class of events.

Monitoring Service Internal Events. This class of events contains all the events used by the monitoring service for its own purposes, such as synchronizing or ordering of events, or to signal extraordinary behavior of the monitoring service, for example, monitoring data loss.

4.3 Æthereal Events

In principle, many instantiations of the above taxonomy for any given NoC are possible, depending on the level of abstraction of the defined events and the monitoring purpose. This section presents part of an instantiation of the taxonomy for the Æthereal NoC. We present examples of Æthereal NoC events in each of the corresponding event classes. As producer and timestamp are always present, we only give the identifier and the attributes for each event.

User Configuration Events. Interprocessor communication via the Æthereal NoC is performed by means of connections. The *Connection Opened* event shows when a certain connection has been opened. The attributes are the connection identifier, the type of the connection, for example, narrowcast, the ports between which the connection exists, the path of the connection and whether it is a GT or a BE connection. A *Connection Closed* event shows when a connection is torn down. Its attribute is the connection identifier.

User Data Events. Sniff events for the Æthereal NoC refer to sniffing flits, either BE or GT. Flits are sniffed from the router queues. Sniffing multiple flits can emulate sniffing a complete packet or even complete messages. *BE Sniff* and *GT Sniff* events are generated when a BE flit and GT flit are sniffed, respectively. Their attributes are the identifier of the queue from which the flit was sniffed and the BE or GT flit itself.

NoC Configuration Events. The *Reserve Slot* event shows when a certain slot in the slot table of a router or network interface has been reserved. The attributes are the slot number and its value. The *Free Slot* event shows that a slot table in the router has been freed. Its attribute is the slot number. A

System packet arrived event shows when a programming packet addressed to a router has arrived there. Its attribute is the entire packet. The purpose of this attribute is to trace a system packet and to see what actions will occur because of it. For example, a setup system packet can go through multiple routers and program all of them.

NoC Alert Events. The *Queue filling* event is specified taking into account the number of queues a router has. In case of a four-port router, we have four attributes, namely, the queue fillings in absolute numbers for each of the four queues. The *Queue full for X cycles* event has as attributes the queue identifier and a value for X. The queue identifier pinpoints the specific queue while the X attribute is a number specifying the number of clock cycles the queue stayed full. A *Queue resuming sending* event, with attribute queue identifier, shows that the queue has resumed sending packets after it has been idle for some time. An *End-to-end credit 0* event is a flow control event showing when the remaining buffering credit for a certain connection is zero, leading to a blocked IP. Its attribute is the connection identifier.

Monitoring Service Internal Events. Each probe in a NoC can generate events. The total order of events for one event generator is given by the timestamp. For area efficiency reasons, a timestamp is necessarily limited to a specific maximal value. After reaching this value, the timestamp counter wraps itself. A *Synchronization* event is always produced when the event counter wraps. The event has no attributes and is only required to allow the proper synchronization for one probe. Currently, Æthereal works with a totally synchronous NoC. The event definition for the Æthereal setup allows to reliably reconstruct the overall partial order of monitoring events. The methods described in this paper will also work with asynchronous NoCs but the timestamping policy will be influenced.

5. MONITORING PROBES

5.1 Generic Architecture

The generic NoC monitoring probe architecture consists of three components, see Figure 4: a sniffer (S), an event generator (EG) and a monitoring network interface (MNI). The probe has as input a number of signals obtained by the sniffer from the router. Based on these signals, the probe generates timestamped events through the EG, which must be sent to the MSA. This means the events must be packetized and sent through the network. This is what the MNI is doing. The means of implementing the sniffer, EG and MNI blocks and their architectural details, for example, timestamping policy, are NoC dependent.

5.2 Æthereal Probe Architecture

This section presents the architecture of the monitoring probes for the Æthereal NoC. Monitoring probes are implemented in hardware and are a design-time choice. The system designer has to decide what level of monitoring is desirable

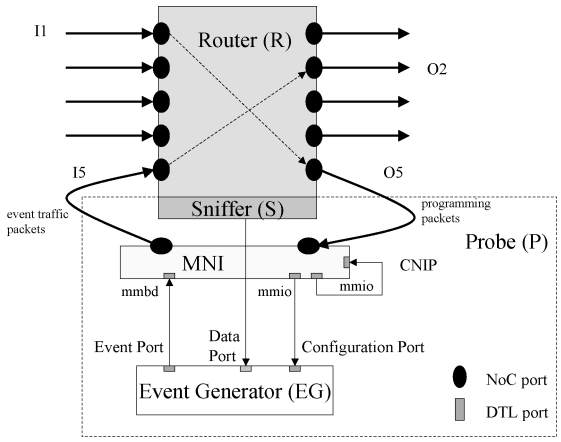

Fig. 4. Æthereal probe architecture.

and affordable. The proposed monitoring architecture is suitable for multiple levels of abstraction.

An Æthereal probe is attached to a router or to a NI. For simplicity, we restrict ourselves to routers in the following. For NIs, there are no conceptual differences.

The Æthereal monitoring probe can be seen in Figure 4. It features a modular design, and can be used without changing the design of the router or NI. In the following, we briefly present each of its components. The basic scenario is very simple: programming packets are coming through the NoC on any of the router ports, for example, I1 in Figure 4. They are transferred by the router from I1 to output O5. The MNI depacketizes the programming packets and configures itself via CNIP and the EG via the Configuration Port. The EG generates events and transfers them to the MNI, via the Event Port, where they are packetized. Packets are sent to the input I5 of the router to be sent to the MSA, via, for example, O2 in Figure 4.

5.2.1 *Sniffer.* The task of the sniffer is to get info from the router and offer it as input data to the EG. Sniffing the signals is not intrusive. The input data are signals obtained by means of SPY-like [Vermeulen et al. 2001] mechanisms. The SPY approach allows a limited set of signals to be monitored in real time, while the chip is in the application, using a multiplexer tree of dedicated wires attached to points of interest in the circuit/SoC. Sniffers can be attached either to routers or to the links between them. We attach the sniffer directly to routers because then we can also have access to the router internals. The sniffer delivers the signals to the EG data port.

5.2.2 *Event Generator.* The task of the EG is to generate timestamped events based on the input received from the sniffer. The EG can generate instances of multiple event types. Event types supported are a design-time choice, their selection is a run-time choice. The EG passes the generated events to the MNI.

The general format of the Æthereal NoC event is:

Identifier	Timestamp	Producer	Attributes
8 bits	16 bits	8 bits	$\geq$0 words

Currently, the Æthereal event *identifier* is an 8-bit code. Æthereal events are not all of the same size. Event attributes can be enabled or disabled allowing any combination of existing attributes for one event. The number and size of attributes is determined by the identifier.

We use a 16-bit *timestamp*. This timestamp is obtained by taking advantage of the 8-bit counter in the router used for slot table iterations. As an 8-bit counter is considered too small for the timestamp, for synchronization reasons, it is extended with another 8-bit counter in the event generator itself. In this way, we can iterate 256 revolutions of 256 slots, without wrapping the timestamp counter.

The *producer* specifies the router that has the probe generating the event attached to it. Currently, we are using an 8-bit code, allowing for 256 producers, which fits today's requirements.

The identifier, timestamp and producer together consist of one (32-bit) word matching the Æthereal link width. The *attributes* can have any maximum size in principle. In our current implementation, our longest event with all attributes enabled is 5 words.

EGs contain an on/off switch, masks for selection of events, their corresponding attributes and activation time, the timestamping unit and a queue for events. Currently, we use a 10-word queue to accommodate two maximum-size events.

The current Æthereal EG has three ports, see Figure 4: one data port, one configuration port and one event port. The data port gets the input from the sniffer. The programming port, a memory-mapped I/O slave, is connected to the MNI output port. The event port, a master memory mapped block data is connected to the MNI input port. The generated events are posted in the internal queue and from there they are passed to the network interface.

5.2.3 *Monitoring NI.* The MNI is a standard NI; we call it the MNI in order to help distinguish it from the other NIs present in the NoC. The events generated in the EG are transferred to the MNI. The MNI packetizes events and sends them via the NoC to MSAs like any other data. The MNI can be configured by any master attached to the NoC through its configuration port CNIP [Rădulescu et al. 2005] in order to setup the connection for the monitoring packets.

The MNI has two Network Interface Ports (NIPs) for communication with the EG: one master memory-mapped I/O port and one slave memory-mapped block data port. The master port connects to its slave pair in the EG and the slave port connects to its master pair in the EG. This MNI has also one bidirectional port to communicate with the router, and is independent of the events fed to it. Packets are queued internally in the MNI queue and then sent to the router.

The design of the MNI makes it possible to treat the EG like any other IP connected to the NoC, which has advantages in the design of the monitoring service and in the co-design of the system and its debug support.

6. PROBE PROGRAMMING MODEL

6.1 Generic Model

The previous section explains the architectural features of the probes. This section presents the associated programming model.

In general, it is important to decide when a probe can be configured. Assuming probes are physically present in the NoC, they can be configured at three possible moments: NoC initialization time, NoC reconfiguration time, or run-time.

Our goal is to make the service available at arbitrary moments at run-time. The monitoring probes are programmed using the NoC itself. It has been shown [Rădulescu et al. 2005] how to configure the NoC at run-time, using the NoC itself. Similar techniques can also be used for programing the probes, requiring no additional communication infrastructure.

The programming of the probes must include the selection of desired events to be generated, including their desired attributes, the selection of the probes to generate the events, a means to enable or disable probes, a way of timing and the setup of traffic monitoring connections.

6.2 Æthereal Model

For Æthereal, we are able to configure the probes at any of the above mentioned moments using memory-mapped I/O read/writes as in AXI [ARM 2003], OCP [OCP International Partnership 2001] or DTL [Philips Semiconductors 2002]. The EG is a slave with a memory-mapped configuration space slave interface. This means that the registers in EG appear in the general memory map allowing them to be read or written by any master. In this way, we implement the programming of the EGs. This is in line with the de facto programming model for the NIs. An EG can therefore be configured by means of simple memory-mapped I/O write operations. Multiple probes can be programmed independently in parallel. If more probes must be configured, then each can be selected at run-time and each probe must be configured separately.

Programming follows two conceptual programming steps for each probe:

(1) *Monitoring Connection Setup*. Events are generated in the EG and then packetized in the MNI. Packets containing events must reach the MSA where the monitoring service has been requested. This is done by setting standard Æthereal connections from the MNI to the MSA. Note that both the MNI and the MSA are standard NIs supporting such connections.

(2) *Probe Setup*
   (a) *Event Selection*. After a connection has been configured, the desired events to be generated must be enabled. By default, all events are disabled. Events are activated by writing the event masks in the EG. Multiple events can be active simultaneously.

(b) *Select Attributes*. Each of the attributes of an event can be selected. Therefore, for each of the events selected in the previous step, the selection of desired attributes is mandatory. The default is that all attributes are disabled. This means that by default all events will be limited to identifier, producer and timestamp. Attributes are enabled by writing attribute masks in the EG.

(c) *Select Time*. When programming the probe, the programmer can optionally set the time that the event generation should start. This is done by setting the time mask in the EG. This mask will be compared with the 8-bit value of the counter in the EG, and at the first match the event generation will start. Selecting the start time of the event generation guarantees in a synchronous NoC the simultaneous start of the event generation by multiple probes.

(d) *Enable/Disable Probe*. Even after the time selection, the probes have to be enabled. Probes are by default disabled. The user can enable or disable any probe in the NoC. Only when the probe is enabled, the event generation starts. Without the time mask the event generation starts immediately. If multiple probes are enabled, it is not possible to guarantee that they will be simultaneously enabled, because of network latency. If the same time mask is set for all EGs, all EGs will simultaneously start at the first encounter of the time set. Enabling or disabling probes is done by writing the enable/disable mask.

The time required to set up the monitoring system is composed of the time required for setting the connections between the EGs and the MSA, and the time required to configure the probes. As an example, the setup time for one connection with a path of length four is 90 ns [Rădulescu et al. 2005]. The time required for configuring the probes depends on the time required for a write operation to a probe register and the number of registers to be written. For example, doing a write transaction for probe configuration, with a payload of two words, to a probe via a GT connection of length three takes 54 ns. The write transactions for programming the probes can be pipelined. We need further experimentation with realistic applications of the monitoring service to assess the impact of the setup time on the performance of the monitoring service.

7. TRAFFIC MANAGEMENT

7.1 Generic Strategy

The monitoring traffic is composed of the probe configuration traffic and the event traffic.

Probe Configuration Traffic. This is all traffic required to setup and configure the monitoring service. It includes the traffic required to configure the probes and the traffic for setting up connections for the transport of data from the probes' MNIs to the NI port that requested the monitoring service. The probe configuration traffic depends on the number of probes being setup.

Event Traffic. This is all the traffic produced as a result of event generation in probes. The event traffic depends on the number of probes setup as well as on the time a probe is enabled.

The monitoring traffic can use existing NoC communication services or a dedicated interconnect, for example, a debug bus. In case existing NoC services are used, additional traffic is introduced in the NoC but no extra interconnect is needed. In case of a dedicated interconnect, no additional traffic is introduced on the existing NoC but more effort is required to design or use another scalable interconnect.

7.2 Æthereal Strategy

The Æthereal monitoring traffic uses the NoC itself and it is based on the existing Æthereal communication services. In this way, no separate interconnect, for control as well as for use, is required for the monitoring probes. There are several choices:

Using GT Services. All the monitoring traffic uses GT connections. A connection is set up between the MSA and the MNI of the specific probe. Each probe in the system uses its own GT connection. In this way, even if the network is congested, monitoring traffic can still reach the MSA at a guaranteed data rate, offering a real-time behavior of the monitoring service. BE user traffic can use the reserved slots when no monitoring traffic is present. It is the safest option from the debugging point of view, but it may interfere with the setup of new GT connections and BE traffic not related to monitoring but to the monitored system. Currently, we use GT services as the implementation choice.

Using BE Services. All monitoring traffic uses BE connections. In case of congestion, it may not be possible for monitoring traffic to reach the MSA at a predefined data rate. The use of BE services is the least intrusive for existing traffic because it does not interfere with GT traffic, but it may interfere with BE traffic. Note that GT debug connections as discussed in the previous option, interfere with user GT connections in the sense that they limit the slot table allocation for the latter.

Using GT and BE Services. When configuring the monitoring service, if more probes are used, it is possible to use either GT or BE for each probe, in order to balance the overhead of the monitoring service. For the distributed monitoring service shown in Figure 2, for example, the traffic from all the probes in the subset of the dedicated IP can use GT services and the traffic from all the probes in the subset of the off-chip interface can use BE services.

8. MONITORING SERVICE COST

8.1 Cost Inventory

The NoC monitoring service provides run-time observability of the NoC behavior. However, this capability does not come for free for the NoC or SoC designer. This section presents an inventory of costs associated with it: area, traffic, and energy.

Area. We target an area budget for the NoC monitoring service of 15–20% of the NoC area. NoC area does not include the area of the IPs or other parts of the chip, it refers to the NoC interconnect area only. Looking at the area cost of commercially available run-time monitoring solutions for computation cores like ARM's ETM has led us to this area budget. The area cost of ARM's ETM10 [ARM 2002] is 1.12 mm^2 compared to the 6.9 mm^2 area of one ARM1022E core with 2x16K cache, for which it was designed. In this case, the area cost for monitoring the computation at run-time is 17.4%, not including the cost of an Embedded Trace Buffer (ETB) that may be required. For communication observability, for example, our monitoring system, we believe a similar area budget is reasonable.

Looking at Figure 4, we get a basic idea of the components that must be physically added to the routers for the NoC monitoring system. All these components have an impact on the area:

(1) Figure 4 suggest that an extra bidirectional router port is required to connect the probe. This means that all probed routers will have a higher arity. For example, moving from arity 4 to arity 5 for an Æthereal router means increasing router area from 0.11 mm^2 to 0.13 mm^2, in a 0.13-micron technology. As we will see later, in Section 8.2.3, it turns out that we can often optimize the monitoring system in such a way that an extra port is not needed.

(2) A network interface with two network interface ports (NIPs)(mmio and mmbd in Figure 4) and one configuration port (CNIP) needs to be added per probed router. The cost of such an Æthereal NI is 0.07 mm^2 in a 0.13-micron technology. For our monitoring solution, this can be optimized; see Section 8.2.3 for more details.

(3) A Sniffer is required in order to get information from the routers in a non-intrusive way. Details of the SPY mechanisms and their area costs are presented in Vermeulen et al. [2001]. The costs are determined by the number of signals the designer wants to be monitored. Thus, it makes for example a difference whether a designer wants to monitor a single router queue at a time or all queues simultaneously.

(4) An Event Generator creates events based on the signals monitored by the sniffer. The cost of the EG depends on the number and type of events desired. Therefore, EGs can vary from very simple ones, coming at basically no area cost, to very complex ones with significant area cost. For example, an estimation of area cost for one EG in the form of a watchpointing unit for four Æthereal router links, together with its corresponding Sniffer, is 0.028 mm^2, in a 0.13-micron technology. This estimate is based on the cost of a 128-bit Sniffer, corresponding to sniffing the four 32-bit router links, a watchpointing unit with four 32-bit comparators, and one 128-bit control register, meaning that all the Æthereal router links can be watchpointed simultaneously.

Traffic. The NoC monitoring system presented uses the NoC for transporting the events it generates. Therefore, the NoC monitoring traffic coexists in

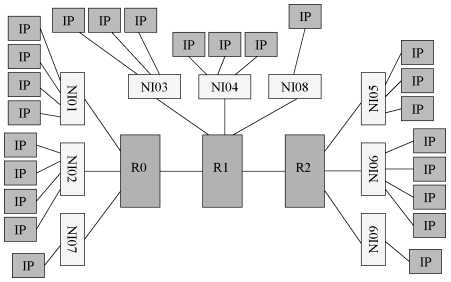

Fig. 5. MPEG NoC1.

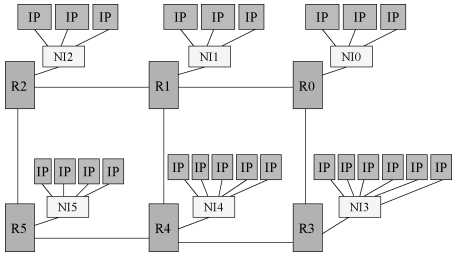

Fig. 6. MPEG NoC2.

the NoC with the user traffic. In this way, it uses NoC resources. The NoC monitoring traffic should be as low as possible compared to the user traffic. The previously mentioned area costs, are the directly visible costs of the NoC monitoring system. It may be possible to also have a hidden cost: the NoC designer may have to overdesign the NoC in order to be able to accommodate the monitoring traffic besides the user traffic.

Energy. The energy consumed by the NoC monitoring system consist of a variable part related to the monitoring traffic and a constant part required by the probes to function.

In the remainder of this section, we further quantify the area and traffic costs. The energy aspect is left for future work.

8.2 Area

8.2.1 *MPEG Examples.* In order to study the area cost, we start with two NoC instances for an MPEG codec with 24 IPs including 3 memories, 21 GT connections with bandwidth varying from 54 to 120 MB/s. The two examples were designed, evaluated and presented in Goossens et al. [2005].

The first example, NoC1 in Figure 5 is a 3×1 mesh. Each router has three NIs attached to it, each NI connecting multiple IPs to the NoC. The original area of this NoC is $1.86\,\text{mm}^2$. The area does not include the IPs connected to the NoC.

The second example, NoC2 in Figure 6 is a 2×3 mesh. Each router has one NI connected to it and one or more IPs are connected to each NI. The original area of this NoC is $2.35\,\text{mm}^2$, again not including the IPs.

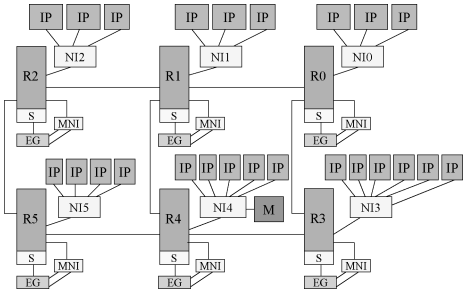

Fig. 7. NoC2 extended with probes, centralized service.

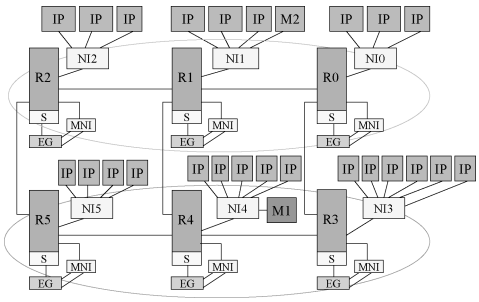

Fig. 8. NoC2 extended with probes, distributed service.

The NoCs used in the two examples have different topology, sizes, and number of NIs attached to the routers. Both NoC instances are automatically generated using the Æthereal design flow [Goossens et al. 2005]. Results are SystemC and synthesisable RTL VHDL, directly usable in a back-end design flow. NoC1 of Figure 5 was subsequently manually optimized for area.

8.2.2 *Adding the Probes. Naive Version.* We have added the monitoring probes to the already presented examples. We have implemented a centralized monitoring service for both NoC examples. Figure 7 shows the architectural implications for NoC2. A distributed monitoring service, see Figure 8, has been developed for NoC2. In the centralized monitoring service of Figure 7, all monitoring traffic goes to the IP called M. In the distributed monitoring service, the traffic from probes attached to routers R0, R1 and R2 goes to IP M2, and the traffic from the probes connected to routers R3, R4, and R5 goes to IP M1.

The area of NoC1 extended with probes is 2.26 mm^2. The area of NoC2 with probes is approximatively 3.09 mm^2. The centralized version is slightly smaller than the distributed one but that is not visible in the area numbers with two decimals accuracy. The monitoring service needs a NIP for each MSA. The area results do not include the area for the sniffers and the area for the EGs, but only the area of the extra cost parts in the routers and MNIs. As already mentioned, the size of the sniffer and the EG heavily depends on the events that should be supported.

Taking into account the area estimate of 0.028 mm^2 for the EG in the form of a complex watchpointing unit with its associated sniffer, as described in

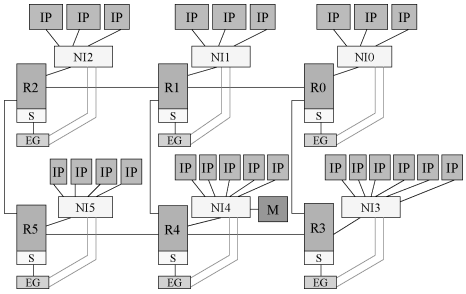

Fig. 9. NoC2 with optimized probes, centralized service.

Section 8.1, the area of NoC1 extended with probes becomes 2.34 mm^2 and the area of NoC2 with probes becomes 3.26 mm^2.

8.2.3 *Improved Version.* Observing the probed MPEG NoCs from Figures 7 and 8 we see that on the probed routers we already have at least one NI besides the MNI of the probe. As we have previously mentioned, the MNI is a common NI parametrized for monitoring purposes. It is therefore possible and straightforward to merge the MNI with one of the NIs already connected to the routers, reducing the area cost, see Figure 9. The resulting NIs are larger in area than the NIs not merged with their MNI counterpart but we can reuse the NI present together with its configuration port and one of the router ports which already connects the NI. One router port is therefore also spared when compared to the naive solution, further reducing the area cost. Please note that this optimization is only possible because we have reused the NI design for the MNI. The resulting area of the optimized NoC2 as presented in Figure 9 is approximatively 2.75 mm^2. This area result does again not include the area of the sniffers and EGs. Taking into account the area estimate for the watchpointing EG with its associated sniffer, like in the previous section, the area of NoC1 extended with probes becomes 2.17 mm^2 and the area of NoC2 with probes becomes 2.92 mm^2

8.2.4 *Area conclusions.* To conclude the area section, we summarize the results:

Area (mm^2)	NoC1(8 slots)	NoC1(16 slots)	NoC2
initial	1.86	1.94	2.35
naive	2.26 (+21%)	2.26 (+16%)	3.09 (+31%)
naive+EG/S	2.34 (+26%)	2.34 (+21%)	3.26 (+39%)
improved	2.09 (+12%)	2.09 (+8%)	2.75 (+17%)
improved+EG/S	2.17 (+17%)	2.17 (+12%)	2.92 (+24%)

It turns out that the original hand optimized NoC1 cannot accommodate the monitoring traffic with the built-in 8-slot routing tables. Therefore, the number of slots had to be increased from 8 to 16, leading to an increase in the NoC1 area from 1.86 mm^2 in the original version to 1.94 mm^2. If we consider the original NoC1 instance with 8 slots as the starting point of comparison, the result is an increase in area with 12% (not counting the EG/Sniffer components); otherwise,

the result is an increase in area with 8%. Taking into account also the EG/S area estimates, area increases with 17% and 12% respectively.

NoC2 can accommodate the debug traffic without modifications. Compared to the original version, we see an increase in area of 17%. Taking into account also the EG/S area estimates, area increases with 24%. Please note that the 24% increase for the NoC2 case is only 0.57 mm^2.

These examples show that:

(1) The area cost of the NoC monitoring service based on the NoC itself, not including the area of the sniffer and the EG, are contained within an average of 14.5% (average of 12% for NoC1 and 17% for NoC2). This area cost is not affected by the area of the EG and sniffer. The sniffer and EG sizes depend on the events desired to be monitored and therefore can substantially vary.

(2) The total area costs of the NoC monitoring service including the watchpointing EG and sniffer estimates are sustainable, and in the order of 17–24% compared to the whole NoC area. The monitoring system for NoC1 (17%), fits into the proposed area budget. The monitoring system for NoC2 (24%), exceeds the proposed area budget. On average (20.5%), the area cost is very close to the proposed area budget of 15–20%. Together with further optimizations, it should be possible to bring the total area cost for NoC2 monitoring system as well as for other examples within the proposed NoC monitoring service total area budget of 15–20%.

The presented area costs are first results and can be improved. No optimizations, except the most straightforward ones were made. However, such optimizations are possible; for example, not probing each router but using a smart placement of the probes can drastically reduce the overall area cost of the monitoring system, making the NoC monitoring service area cost even more acceptable. It is beyond the purpose of this article to present these optimizations.

8.3 Event Traffic

Media processing SoCs for Set-top Box applications or digital TV consist of audio encoding or decoding, for example, AC3, and video processing functions, for example, H263 or MPEG-2 [Goossens et al. 2004]. We briefly look into the monitoring event overhead for such a media processing SoC using a NoC.

From Goossens et al. [2004], we learn that the typical task graph of such SoC has approximately 200 connections all over the NoC. When the application task graph changes, a partial or complete reconfiguration of the NoC is needed. A complete reconfiguration means that all the connections are torn down and a new set of connections is set up. Reconfiguration is required at a maximum rate of once per second. A partial reconfiguration means that only part of connections, for example, half, are torn down and a similar number of connections are set up.

In case we monitor reconfiguration, we focus only on two events, namely *OpenConnection* and *CloseConnection* events. The average cost of these events is two (32bit) words. Each probe monitors the *OpenConnection* and *CloseConnection* events, with all attributes enabled. One flit comprises three words, one

being the header and two being the useful payload. For a complete reconfiguration, the total useful event payload is 3.2 KB, leading to an event traffic of 4.8 KB, that is:

$$200(\text{connections}) \times 2(\text{events}) \times 1(\text{flit}) \times 3(\text{words}) \times 4(\text{bytes}) = 4800 \text{ bytes}$$

In case we monitor the functional data over one connection, for example, a reserved 30 MB/s GT connection, used for writes to a memory, with an actual usage of 28.6 MB/s, we focus on the *GTSniff* event. In this case, as we monitor only one connection, the source, the identifier and the timestamp of the events are not necessary. An event will be the flit itself. We are interested in the functional data passing the connection. The 30-MB/s GT user connection uses 50.25-MB/s traffic in the NoC, considering all the overhead caused by the transaction protocol used (commands and addresses are added to messages) and the packetization. In this way, we sniff 50.25 MB/s of data, which is the payload for the debug connection. Assuming the debug connection writes this data into memory, through the MSA, commands and addresses are added bringing the total required before packetization to 62.3 MB/s, and to 83 MB/s afterwards it. These numbers can be improved by removing the packetization overhead from the sniffed connection (the sniffed packet headers).

These examples show that the traffic related to the monitoring service is sustainable by mature NoCs if events are carefully selected and enabled at the right time. For example, the raw bandwidth of the Æthereal NoC is 2 GB/s per link and the monitoring traffic from our reconfiguration monitoring is 4.8 KB/s, approximately six orders of magnitude less, while the monitoring traffic from our connection sniff example is 62.3 MB/s, two orders of magnitude less. More enabled events would lead, of course, to more traffic.

9. CONCLUSION

In this article, we have presented the concepts of a NoC monitoring service, the first one described in the scientific literature. This monitoring service offers communication observability at run-time. It can be used for example for on-chip or off-chip application and system-level debugging, but also for run-time performance analysis. The monitoring service can be configured and used at arbitrary moments during run-time, offering increased flexibility.

The monitoring service is integrated in the NoC and uses the NoC communication services for configuration as well as for the event traffic. It can be instantiated automatically together with the NoC, saving design time. The monitoring service consists of probes attached to NoC components, allowing easy scalability of the service. The generic architectural concepts of the probe feature a programmable modular design composed of sniffer, monitoring network interface and event generator, providing flexibility to target the service to the monitoring task at hand.

Proof of concept is achieved via implementation for the Æthereal NoC. Probes model the monitored information in the form of timestamped events. We have presented our event model, a generic event taxonomy for NoCs and one of the possible instantiations for the Æthereal NoC. Run-time programmability of the

probes is achieved via memory-mapped configuration ports in the probe. Traffic management is achieved by reusing the guaranteed communication services of the NoC. This provides potentially nonintrusive real-time monitoring.

The cost of the monitoring traffic is low, being several orders of magnitude lower than the bandwidth available in the NoC, for two realistic examples in a typical media processing SoC. The area cost for the NoC monitoring service targets a 15–20% of the total NoC area. Initial experiments on two different MPEG NoCs show that this target is realistic.

The generic concepts presented allow to retarget the NoC monitoring service to other NoCs. Almost all NoCs have network interfaces and routers as basic building blocks. They all provide some sort of communication services and support a certain design flow to instantiate these blocks and services. Furthermore, they all provide means of configuration or reconfiguration. In order to instantiate the proposed NoC monitoring service for another NoC, these concepts can be reused leading to a reasonable implementation effort.

In summary, the proposed monitoring service satisfies all requirements set out in Section 1.3. Future work includes further cost analysis and optimization of the NoC monitoring service, and integrating computation observability with the NoC monitoring service in order to achieve a complete SoC monitoring solution. Since not all monitoring tasks require the same monitoring functionality, we also plan to look into functionality vs. cost trade-offs.

ACKNOWLEDGMENTS

The authors wish to thank Bart Vermeulen for the valuable feedback he provided on an early draft of this article.

REFERENCES

ARM. 2002. *Embedded Trace Macrocell*. www.arm.com.

ARM 2003. *AMBA AXI Protocol Specification*. ARM.

BENINI, L. AND DE MICHELI, G. 2002. Networks on chips: A new SoC paradigm. *IEEE Comput. 35*, 1, 70–80.

BOLOTIN, E., CIDON, I., GINOSAR, R., AND KOLODNY, A. 2004. QNoC: QoS architecture and design process for network on chip. *J. Syst. Archit. 50*, 2–3 (Feb.), 105–128. (Special issue on Networks on Chip).

CIORDAS, C., BASTEN, T., RĂDULESCU, A., GOOSSENS, K., AND VAN MEERBERGEN, J. 2004. An event-based network-on-chip monitoring service. In *Proceedings of the International High-Level Design Validation and Test Workshop (HLDVT)*. 149–154.

DALLY, W. J. AND TOWLES, B. 2001. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the Design Automation Conference (DAC)*. 684–689.

GONZÁLEZ PESTANA, S., RIJPKEMA, E., RĂDULESCU, A., GOOSSENS, K., AND GANGWAL, O. P. 2004. Cost-performance trade-offs in networks on chip: A simulation-based approach. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*. 764–769.

GOOSSENS, K., DIELISSEN, J., GANGWAL, O. P., GONZÁLEZ PESTANA, S., RĂDULESCU, A., AND RIJPKEMA, E. 2005. A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC design and verification. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*. 1182–1187.

GOOSSENS, K., DIELISSEN, J., VAN MEERBERGEN, J., POPLAVKO, P., RĂDULESCU, A., RIJPKEMA, E., WATERLANDER, E., AND WIELAGE, P. 2003. Guaranteeing the quality of services in networks on chip. In *Networks on Chip*, A. Jantsch and H. Tenhunen, Eds. Kluwer, Chap. 4, 61–82.

GOOSSENS, K., GANGWAL, O. P., RÖVER, J., AND NIRANJAN, A. P. 2004. Interconnect and memory organization in SOCs for advanced set-top boxes and TV—Evolution, analysis, and trends. In *Interconnect-Centric Design for Advanced SoC and NoC*, J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, Eds. Kluwer, Chap. 15, 399–423.

GOOSSENS, K., VAN MEERBERGEN, J., PEETERS, A., AND WIELAGE, P. 2002. Networks on silicon: Combining best-effort and guaranteed services. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*. 423–425.

GUERRIER, P. AND GREINER, A. 2000. A generic architecture for on-chip packet-switched interconnections. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*. 250–256.

IEEE-ISTO 5001TM. 2003. *The NEXUS 5001 Forum Standard for a Global Embedded Processor Debug Interface*. www.nexus5001.org.

KARIM, F., NGUYEN, A., AND DEY, S. 2002. An interconnect architecture for networking systems on chips. *IEEE Micro 22*, 5 (Sept.), 36–45.

KUMAR, S., JANTSCH, A., SOININEN, J.-P., FORSELL, M., MILLBERG, M., OBERG, J., TIENSYRJA, K., AND HEMANI, A. 2002. A network on chip architecture and design methodology. In *Proceedings of the Symposium on VLSI*. 105–112.

MANSOURI-SAMANI, M. AND SLOMAN, M. 1996. A configurable event service for distributed systems. In *Proceedings of the International Conference on Configurable Distributed Systems*. 210–220.

MILLBERG, M., NILSSON, E., THID, R., KUMAR, S., AND JANTSCH, A. 2004. The Nostrum backbone—A communication protocol stack for networks on chip. In *Proceedings of the International Conference on VLSI Design*. 693–696.

MURALI, S. AND DE MICHELI, G. 2004. Bandwidth-constrained mapping of cores onto NoC architectures. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*. 896–901.

OCP INTERNATIONAL PARTNERSHIP. 2001. Open core protocol specification.

PHILIPS SEMICONDUCTORS. 2002. *Device Transaction Level (DTL) Protocol Specification Version 2.2*.

POPLAVKO, P., BASTEN, T., BEKOOIJ, M., VAN MEERBERGEN, J., AND MESMAN, B. 2003. Task-level timing models for guaranteed performance in multiprocessor networks-on-chip. In *Proceedings of the International Compilers, Architecture, and Synthesis for Embedded Systems*. 63–72.

RĂDULESCU, A., DIELISSEN, J., GONZÁLEZ PESTANA, S., GANGWAL, O. P., RIJPKEMA, E., WIELAGE, P., AND GOOSSENS, K. 2005. An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network programming. *IEEE Trans. CAD Integ. Circ. Syst. 24*, 1 (Jan.), 4–17.

RĂDULESCU, A. AND GOOSSENS, K. 2004. Communication services for networks on chip. In *Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation*, S. S. Bhattacharyya, E. F. Deprettere, and J. Teich, Eds. Marcel Dekker, 193–213.

RIJPKEMA, E., GOOSSENS, K. G. W., RĂDULESCU, A., DIELISSEN, J., VAN MEERBERGEN, J., WIELAGE, P., AND WATERLANDER, E. 2003. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*. 350–355.

VERMEULEN, B., DIELISSEN, J., GOOSSENS, K., AND CIORDAS, C. 2003. Bringing communication networks on chip: Test and verification implications. *IEEE Commun. Mag. 41*, 9 (Sept.), 74–81.

VERMEULEN, B., OOSTDIJK, S., AND BOUWMAN, F. 2001. Test and Debug Strategy of the PNX8525 Nexperia Digital Video Platform System Chip. In *Proceedings of the International Test Conference (ITC)*. 121–130.

Received November 2004; revised March 2005 and May 2005; accepted July 2005