AN EVENT-BASED NETWORK-ON-CHIP MONITORING SERVICE

Calin Ciordas † *Twan Basten* † *Andrei Rădulescu* ‡ *Kees Goossens* ‡
Jef van Meerbergen †‡
† *Eindhoven University of Technology, Eindhoven, The Netherlands*
‡ *Philips Research Laboratories, Eindhoven, The Netherlands*
c.ciordas@tue.nl

Abstract

Networks on chip (NoCs) are a scalable interconnect solution for large scale multiprocessor systems on chip (SoCs). However, little attention has been paid so far to the monitoring and debugging support for NoC-based systems. We propose a generic on-line event-based NoC monitoring service, based on hardware probes attached to NoC components. The proposed monitoring service offers run-time observability of NoC behavior and supports system-level and application debugging. The defined service can be accessed and configured at run-time from any network interface port. We present a probe architecture for the monitoring service, together with its associated programming model and traffic management strategies. We prove the feasibility of our approach via a prototype implementation for the Æthereal NoC. The additional monitoring traffic is low; typical monitoring connection configuration for a NoC-based SoC application needs only 4.8KB/s, which is 6 orders of magnitude lower than the 2GB/s per link raw bandwidth offered by the Æthereal NoC.

1 Introduction

Due to the ever increasing miniaturization of transistors, very complex chip designs are becoming possible. For both physical and complexity reasons, future chip designs will inherently be multiprocessor systems, consisting of hundreds of modules (IPs).

Design of large-scale chips needs to be structured resulting in the decoupling of the communication from computation. Advanced, scalable interconnects such as NoCs [2, 3, 7, 8, 10, 13, 16] have been proposed. They help decoupling computation and communication and offer well defined interfaces [2, 5, 16], enabling IP reuse.

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. Monitoring translates to the observability problem: is it possible to observe the internals of a system while it is running? In this paper we focus on the monitoring task, while recognizing the importance of the debugging task.

Related Work. *Computation observability* received a lot of attention in the literature. Philips' real-time observability solution, SPY [18], allows the nonintrusive 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) [1] 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 of up to 16 bits.

The NEXUS standard [9] 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 active run-time programmable solutions. Programmable NoC solutions can be error prone, e.g. introducing congestion 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 exists, adding to complexity. In the context of the trend towards NoCs, we need communication observability.

Today, in the research community, the focus is on the design [3, 8, 11, 13], analysis [4] and use [6, 14] of NoCs. There is no support for communication observability in NoC-based SoCs. Today, to the best of our knowledge, there are no NoC monitoring systems.

Contribution. This paper proposes a NoC run-time monitoring system for NoC-based SoCs. 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 NoC. The NoC monitoring ser-

vice can be configured from any NoC master network interface. The NoC service is based on hardware probes attached to network components, i.e. routers or network interfaces. The service allows:

(1) The non-intrusive capturing of run-time information and functional data in 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) Run-time 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 this concept via an implementation for the Æthereal NoC [7, 16]. For a typical NoC-based SoC application we show that the additional traffic introduced by the monitoring service is 6 orders of magnitude lower than the usual NoC user traffic.

Overview. Section 2 presents the Æthereal NoC concepts. Section 3 explains the general concept of our NoC monitoring service. Section 4 presents our event model, a generic NoC event taxonomy and an instance of it for the Æthereal NoC. 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 monitoring data traffic. Section 8 presents an example to quantify possible additional traffic introduced by the NoC monitoring service. We end with conclusions.

2 Æthereal NoC

Several NoCs [2,3,7,8,10,13] have been proposed, using different topologies and routing strategies. NoCs are generally composed of network interfaces (NIs) [17] and routers (R) [16]. NIs implement the NoC interface to IP modules. Rs transport data from NI to NI.

The Æthereal NoC [7, 16] runs at 500 MHz and offers a raw bandwidth of 2GB/s per link in a 0.13 micron 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, e.g. point-to-point connections or multicast, which can be either guaranteed throughput (GT) or best effort (BE). Connections have properties such as data integrity, transaction ordering or flow control. Guarantees are provided by means of slot reservations in routing tables of Rs and NIs. Slot reservations are programmed at run-time. 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.

A transaction takes place between multiple IPs via the NoC, and is composed of multiple messages. Each message is composed of several packets, that in turn consist of one or more flits. Flits are the minimal transport unit.

3 NoC Monitoring Service

The monitoring service is offered by the NoC itself, in addition to the communication services offered to IPs. The NoC monitoring service consists of configurable monitoring probes (P) attached to NoC components, i.e. Rs or NIs, their associated programming model, and a monitoring traffic management strategy.

Event model. All the monitored information is modeled in the form of events. An event model specifies the event format, e.g. 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 the collection of 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. The placement of probes is a design-time choice and is related to the cost versus observability tradeoff. E.g. the topright R in figure 1 has no probe attached.

Programming model. The programming model describes the way in which the monitoring service is being setup 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,e.g. using memory mapped I/O [17] [15]. The monitoring service can be configured at run-time, by an IP from any NI, called the monitoring service access point (MSA).

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, e.g. GT or BE, or dedicated services can be used for the traffic management.

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

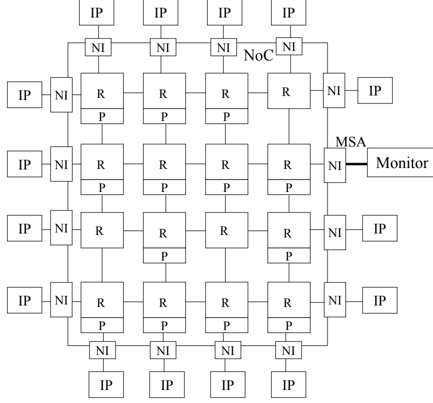

Figure 1. Centralized Monitoring Service

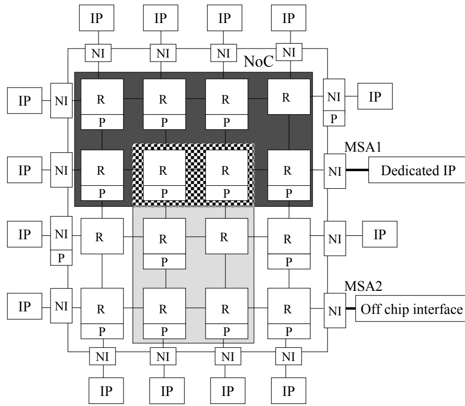

Figure 2. Distributed Monitoring Service

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 R which 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 Rs in the middle of the figure. Monitoring information can be either used on-chip, e.g. 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 [12] 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)

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, exemplified with Æthereal events. The term 'user' is used to identify an IP. 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. This 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.

User Configuration Events. A NoC is used by IPs to communicate to each other. User configuration events expose the configuration of this communication at the level of the user API. Æthereal examples are *Connection Opened* and *Connection Closed* events. The attributes are the connection identifier, the type of the connection, e.g. narrowcast, the ports between which the connection exists, the path of the connection and whether it is a GT or a BE connection.

User Data Events. This class of events 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. Æthereal sniff events, *BE Sniff* and *GT Sniff* inspects flits, either BE or GT. Sniffing multiple flits can emulate sniffing a complete packet or even complete messages. Their attributes are the queue identifier of the queue from which the flit was sniffed and the BE or GT flit itself.

NoC Configuration Events. To achieve interprocessor communication, the NoC must be configured. NoC configuration events expose this configuration of the network, enabling the system debugger to trace it. Æthereal *Reserve Slot* and *Free Slot* events show when a certain slot in the slot table of a R or NI has been reserved, respectively freed. The attributes are the slot number and its value.

NoC Alert Events. In a faulty or ill programmed NoC, problems like buffer overflow, congestion, starvation, livelock or deadlock can appear. Therefore, it is imperative to monitor the network behavior for signals of overload or misbehavior. The Æthereal *Queue filling* event is specified taking into account the number of queues R has. In case of a four port R, we have four attributes, namely the fillings of the four queues. An *End-to-end credit 0* event is a flow control event showing when no buffer credits for a certain connection remain, leading to a blocked IP. This could be an indication of a possible problem. Its attribute is the connection identifier.

Monitoring Service Internal Events cover all the events used by the monitoring service for its own purposes,

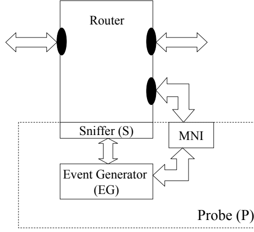

Figure 3. NoC Probe Architecture

such as synchronization or ordering of events, or to signal extraordinary behavior of the monitoring service, e.g. monitoring data loss. For Æthereal, the total order of events for one event generator is given by the timestamp. For efficiency 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 methods described in this paper will also work with asynchronous NoCs but the timestamping policy will be influenced. The event definition for the Æthereal setup allows to reliably reconstruct the overall partial order of monitoring 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.

5 Monitoring Probes

The **generic probe architecture** proposed consists of three components, see Figure 3: a sniffer (S), an event generator (EG) and a monitoring network interface (MNI). EG and MNI and their architectural details, e.g. timestamping policy, are NoC dependent. The EG behaves like any other IP connected to the NoC. Monitoring probes are hardware implemented and are a design-time choice. The system designer can decide what level of monitoring is desirable or affordable.

We present the S, MNI and EG details for the **Æthereal NoC probes**. A probe is attached to a R or to a NI. For simplicity, we restrict ourselves to Rs in the following. For NIs, there are no conceptual differences. The probe features a **modular design**, and can be used without changing the design of the R or NI. Figure 4 presents the detailed architecture of the probe. Programming the probe is very simple: programming packets come through the NoC on any of the R ports, e.g. I1 in Figure 4. They are transferred by the R from I1 to its last output O5. The MNI depacketizes these packets and configures itself via Configuration Network Interface Port(CNIP) and the EG via Configuration Port subsequently. Using sniffed data the EG generates events and transfers them to the MNI, via the Event Port, where they are packetized. Packets are sent to the last input I5 of the R

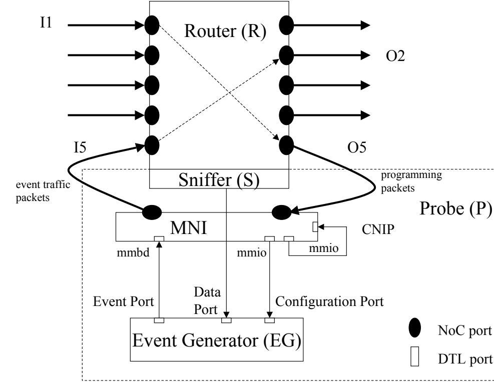

Figure 4. Æthereal Probe Architecture

to be sent to the MSA, via e.g. O2 in Figure 4.

Sniffer. The task of the S is to get info from the R and offer it as input data to the EG. These data are signals obtained by means of non-intrusive SPY [18] like mechanisms. Ss can be attached either to Rs or to the links between them. We attach the S directly to Rs because we can have access also to the R internals. The S delivers the signals to the EG data port.

Event Generator. The task of the EG is to generate timestamped events based on the input received from S. 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. 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 given by the identifier. We use a 16 bit **timestamp**. The **producer** specifies the R that has the probe generating the event attached to it. Currently, we are using an 8 bit code, allowing for 256 producers. The identifier, timestamp and producer together form one (32bit) word matching the Æthereal link width. The **attributes** can have any size in principle. In our current implementation, our longest event with all attributes enabled is 5 words.

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 configuration port, a slave memory mapped I/O port, is connected to the MNI output port. The event port, a master memory mapped block data port is connected to the MNI input port. The generated events are posted in a 10 words EG queue and from there are passed to the MNI. In case an MNI does not accept any more data, events are dropped, in the EG at creation.

Monitoring Network Interface. The MNI is a simple standard NI. The events generated in the EG are transferred to the MNI. The MNI packetizes these 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 [17] in order to set up

the connection for the monitoring packets. The MNI has 2 ports 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. The MNI has one bidirectional 2GB/s link (I5 and O5 in Figure 4) to communicate with the R; it is independent of the events fed to it. It is very small, with a 0.05mm^2 synthesized area in a 0.13 micron technology. For comparison an arity 5 GT/BE router has an area of 0.26mm^2 [16]. Packets are queued internally in the MNI queue and then sent to the R.

6 Probe Programming 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. Our goal is to make the service available at arbitrary moments during run-time. The monitoring probes are programmed using the NoC itself. It has been shown [17] how to configure the NoC at run-time, taking advantage of the existing NoC memory-mapped interface. The same technique is also used for programming the probes, requiring no additional communication infrastructure.

Æthereal Model. For Æthereal, we are able to configure the probes at run-time using memory-mapped I/O, i.e. DTL [15]. The EG is a slave with a memory-mapped configuration space slave interface connected to the NI. It can be configured by means of simple standard write operations. Multiple probes can be programmed independently in parallel.

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.

(2) Probe setup. The set up is done in four steps: event selection, attributes selection (per event type), start time selection, and enable/disable probe.

7 Traffic Management

The **monitoring traffic** is composed of the probe configuration traffic and the event traffic. **Probe configuration traffic** 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 MNI to the NI port which requested the monitoring service. The probe configuration traffic depends on the number of probes being setup. **Event traffic** 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, e.g. 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 but more effort is required to design or use another scalable interconnect.

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, e.g. all the traffic generated by the probes in Figure 1 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 existing BE traffic and the setup of new GT connections which is done through BE packets. 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. GT debug connections 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 Event Traffic Overhead

Media processing SoCs for Set-top Box applications or digital TV consist of audio encoding or decoding, e.g. AC3, and video processing functions, e.g. H263 or MPEG2 [6]. We present the monitoring event overhead for such a media processing SoC using a NoC.

From [6] 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 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, e.g. half, are torn down and a similar number of connections are set up.

In the case we monitor reconfiguration, we focus only on two events, namely *OpenConnection* and *CloseConnection* events. The average cost of these events is two (32 bit) 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 usefull payload. For a complete reconfiguration the total usefull event payload is 3.2 KB, leading to an event traffic of 4.8KB i.e:

200(connections) x 2(events) x 1(flit) = 1200 words

This example shows that the traffic overhead of the monitoring service is easily sustainable by mature NoCs if events are carefully selected and enabled at the right time. E.g. the raw bandwidth of the Æthereal NoC is 2 GB/s per link and the monitoring traffic from our example is 4.8 KB/s, approximately six orders of magnitude less. More events enabled would lead of course to a greater traffic overhead.

9 Conclusion

In this paper, 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 and can be used either for on-chip/off-chip application and system-level debugging. The monitoring service can be configured and used at arbitrary moments during run-time.

The monitoring service is integrated in the NoC and reuses the NoC communication services for configuration as well as for the monitoring traffic. It can be instantiated automatically together with the NoC, saving design time. The monitoring service consists of probes attached to NoC components. The generic architectural concepts of the probe feature a modular design composed of a sniffer, a monitoring network interface and an event generator.

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 programming of the probes is achieved via memory-mapped configuration ports in the probe. Traffic management is achieved by reusing the communication services of the NoC. The cost of the monitoring traffic is low, being orders of magnitude lower than the user traffic in the NoC in case of monitoring a complete reconfiguration.

References

[1] ARM. *Embedded Trace Macrocell Architecture Specification*. www.arm.com, 2002.

[2] L. Benini and G. De Micheli. Networks on chips: A new SoC paradigm. *IEEE Computer*, 35(1):70–80, 2002.

[3] W. J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proc. Design Automation Conference (DAC)*, pages 684–689. IEEE, 2001.

[4] S. Gonzalez Pestana, E. Rijpkema, A. Rădulescu, K. Goossens, and O. P. Gangwal. Cost-performance trade-offs in networks on chip: A simulation-based approach. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*. IEEE, Feb. 2004.

[5] K. Goossens, J. Dielissen, J. van Meerbergen, P. Poplavko, A. Rădulescu, E. Rijpkema, E. Waterlander, and P. Wielage. Guaranteeing the quality of services in networks on chip. In A. Jantsch and H. Tenhunen, editors, *Networks on Chip*, chapter 4, pages 61–82. Kluwer, 2003.

[6] K. Goossens, O. P. Gangwal, Röver, and A. Niranjan. Interconnect and memory organization in SOCs for advanced set-top boxes and TV — evolution, analysis, and trends. In J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, editors, *Interconnect Centric Design for Advanced SoC and NoC*, chapter 15, pages 399–423. Kluwer, 2004.

[7] K. Goossens, J. van Meerbergen, A. Peeters, and P. Wielage. Networks on silicon: Combining best-effort and guaranteed services. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 423–425, Mar. 2002.

[8] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 250–256. IEEE, 2000.

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

[10] F. Karim, A. Nguyen, and S. Dey. An interconnect architecture for networking systems on chips. *IEEE Micro*, 22(5):36–45, Sept. 2002.

[11] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani. A network on chip architecture and design methodology. In *Proc. Symposium on VLSI*. IEEE, 2002.

[12] M. Mansouri-Samani and M. Sloman. A configurable event service for distributed systems. In *Proc. 3rd Int'l Conference on Configurable Distributed Systems*, pages 210–220. IEEE, 1996.

[13] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. The Nostrum backbone - a communication protocol stack for networks on chip. In *Proc. Int'l Conference on VLSI Design*, pages 693–696. IEEE, 2004.

[14] S. Murali and G. De Micheli. Bandwidth-constrained mapping of cores onto NoC architectures. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*. IEEE, 2004.

[15] Philips Semiconductors. *Device Transaction Level (DTL) Protocol Specification Version 2.2*, 2002.

[16] E. Rijpkema, K. G. W. Goossens, A. Rădulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 350–355. IEEE, Mar. 2003.

[17] A. Rădulescu, J. Dielissen, K. Goossens, E. Rijpkema, and P. Wielage. An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network programming. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*. IEEE, 2004.

[18] B. Vermeulen, S. Oostdijk, and F. Bouwman. Test and Debug Strategy of the PNX8525 Nexperia Digital Video Platform System Chip. In *Proc. Int'l Test Conference (ITC)*, pages 121–130. IEEE, 2001.