

Chapter 2

SERVICE-BASED DESIGN OF SYSTEMS ON CHIP AND NETWORKS ON CHIP

Kees Goossens, Santiago González Pestana,
John Dielissen, Om Prakash Gangwal, Jef van Meerbergen,
Andrei Rădulescu, Edwin Rijpkema, and Paul Wielage
Philips Research Laboratories, Eindhoven, The Netherlands
{ Kees.Goossens,Santiago.Gonzalez.Pestana }@Philips.com

Abstract: We discuss why performance verification of systems on chip (SOC) is difficult, by means of an example. We identify four reasons why building SOCs with predictable performance is difficult: unpredictable resource usage, variable resource performance, resource sharing, and interdependent resources. We then introduce the concept of a service, aiming to address these problems, and describe its advantages over “ad-hoc” approaches. Finally, we introduce the ÆTHEREAL network on chip (NOC) as a concrete example of a communication resource that implements multiple service levels.

Keywords: System design, embedded system, system architecture, real time, network on chip, quality of service, performance analysis, best effort.

1. INTRODUCTION

Moore’s Law results in increasing computational power, which enables sophisticated functions to be incorporated in ever-smaller devices. Consumer electronics is shifting from discrete tethered devices to pervasive systems embedded in every-day objects. The increased interaction with the real world (e.g. managing the intelligent home, as opposed to e.g. a stand-alone personal computer) requires real-time reactions, a high degree of reliability, and, for user comfort, predictable behaviour.

Moreover, as the computational power of these systems grows, more advanced algorithms are introduced, such as MPEG4 (moving-picture experts group) and 3D graphics. These algorithms make use of the increased flexibility (software-programmability) of embedded systems. Combining variable

resource requirements (computation, storage, and communication) with the robust and predictable behaviour required by embedded consumer-electronics devices is the challenge that we address in this chapter.

The embedded systems just described are implemented using one or more chips, which together contain one or more *systems on a chip* (SoC). SoCs are composed of hardware components (*intellectual property*, or IP), which are interconnected by a communication infrastructure, here assumed to be a *network on a chip* (NoC).

Designing a SoC is an expensive undertaking, requiring large hardware and software design teams. The bulk of the effort of SoC design resides not in the design of the IP, but in their composition or integration into a larger, working whole. Verifying that the ensemble of IP behaves correctly with the required functionality and real-time performance is the bottle neck in SoC design.

In this chapter we advocate that the notion of *services* can ease system design. Computation, communication, and storage services enable the construction of modular SoCs, allowing compositional verification.

Overview In the following section we describe and analyse the problem of performance verification of SoCs. Building on the notions of resources, their usage and performance, we show that unpredictable resource usage, variable resource performance, and resource sharing, complicate the construction of predictable systems. In Section 3 we define the service concept, describe its advantages over “ad-hoc” approaches, and show how it addresses the performance verification problems identified earlier. In Section 4 we describe a concrete application of the concepts. The *ÆTHEREAL* NoC implements two communication service levels (Goossens, Dielissen, van Meerbergen, Poplavko, Rădulescu, Rijpkema, Waterlander and Wielage, 2003; Rădulescu and Goossens, 2004). We describe their implementation, intended usage, and how they tackle the problems listed above. In Section 5 we reflect and conclude.

2. RESOURCES: THEIR PERFORMANCE AND USAGE

IP *re-use* addresses the so-called design-productivity gap by using IPs in derivative and multiple designs. This approach works well for components, such as peripherals, memories, programmable processors, communication infrastructure, and real-time operating systems (RTOS). *Platforms* (Keutzer, Malik, Newton, Rabaey and Sangiovanni-Vincentelli, 2000), such as Philips’s Nexperia (de Oliveira and van Antwerpen, 2003), provide the next level of re-use, by defining interfaces and protocols to connect the re-usable components (both hardware and software).

As an example, consider one of Philips's largest SoCs to date, PNX8550, shown in Figure 2-1 (Goossens, Gangwal, Röver and Niranjana, 2004), which exemplifies the Philips Nexperia platform. Many parts of its design are re-

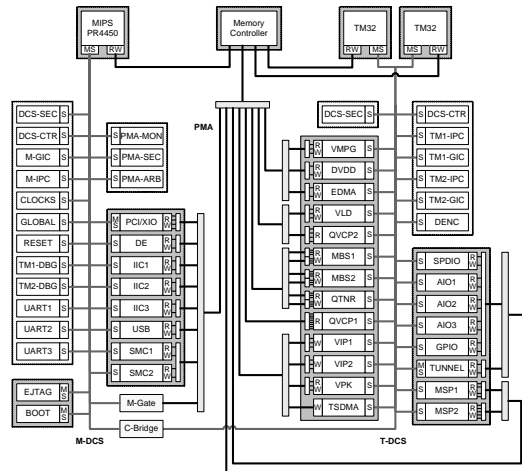


Figure 2-1. Simplified block diagram of PNX8550.

used from earlier designs, or are standard components that can be found in the library of Nexperia-compliant IP. They are combined using the device-transaction-level (DTL, 2002) hardware and TriMedia streaming software architecture (TSSA) software protocols, and use the PSOS operating system, as prescribed by the platform.

PNX8550 implements many set-top-box and video-enhancement functions, which require predictable real-time audio and video streaming. *Performance verification* is the difficult task of ensuring that the assembly of the large number of computation, storage, and communication IP meets all real-time constraints under all circumstances.

Although platforms have made assembling a SoC much easier, insufficient steps have been taken in applying the same ideas to performance verification of the resulting SoC. We tackle this issue by enriching the platform concept with services, to allow more explicit descriptions of, and reasoning about performance.

2.1 Why is Performance Verification Difficult?

Consider PNX8550, and assume that both TriMedia processors (TM32 in Figure 2-1) have a cache, and run multiple real-time tasks scheduled by a RTOS. If we want to know the time it takes a task on one of the TriMedia processors to communicate with a task on the other TriMedia using shared external memory,

there are several issues to consider. First, each task shares its TriMedia with other tasks. PSOS supports task preemption, and its arbitration mechanism therefore determines the delay before a task is active. Second, the caches may or may not contain the instructions and/or data of the task in question, resulting in a varying delay before the requested data is produced. Moreover, the cache is shared with other tasks on the same TriMedia. For example, an interrupt prior to swapping in the task could have flushed the cache, delaying the task's start. Third, the communication between the tasks uses the external memory. The memory is attached to the memory controller, which is again a shared resource with a sophisticated arbiter. Depending on the arbitration scheme employed, the write and read latencies and bandwidths may be influenced by other traffic contending for the external memory, such as the MIPS and the streaming traffic from the pipelined memory-access (PMA) interconnect (Goossens et al., 2004). As a result of these phenomena, computing the communication performance (latency and bandwidth) between the two tasks is non-trivial.

Resources and Users In the above example, we can identify several kinds of *resources*: *computation* (TriMedia, MIPS, IP such as the quality temporal noise reduction or QTNR), *communication* (device-control-and-status (DCS) and PMA interconnects), and *storage* (caches, off-chip and on-chip memories). These are used by several types of *users*: tasks (of computation), communication connections (of communication), and buffers for intermediate results or for communication (of storage).

In the example there are four independent factors complicating the performance analysis, as shown in Table 2-1: unpredictable resource usage, variable resource performance, users sharing resources, and (inter)dependence of multiple resources. We discuss each in turn below.

Table 2-1. Independent factors complicating performance analysis.

	user	resource
uncertainty	unpredictable usage	variable performance
multiple	shared resource	resource dependence

2.1.1 Unpredictable resource usage.

Algorithms defined by newer data compression standards, such as MPEG, are increasingly dynamic. For example, MPEG2's data compression allows variable bit rates, and MPEG4 uses dynamic object creation. As a result, the usage of resources (computation, communication, and storage) to encode or decode is variable. Figure 2-2 shows an abstract example of time-varying usage of a resource ("instantaneous usage").

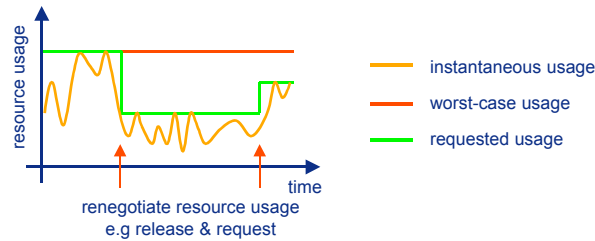


Figure 2-2. Variable resource usage and requested performance.

As an example, compressed MPEG2 data streams usually contain so-called I frames followed by several B and P frames. I frames are larger than B and P frames, and require more computation to decode them. However, I frames are required to decode B and P frames, and hence the memory requirements (size and bandwidth) to decode B and P frame are larger than those for I frames. However, a set-top box, for which PNX8550 is designed, must display a constant number of pictures per second on the TV screen, regardless of the content MPEG stream. Thus, even in a SoC with predictable resources, if their usage is variable, care has to be taken to ensure results with constant quality.

2.1.2 Variable resource performance.

As shown abstractly in Figure 2-3, resources themselves can have varying performance. The “instantaneous performance” of a resource, such as instructions per second, can vary over time for architectural reasons, as we describe below.

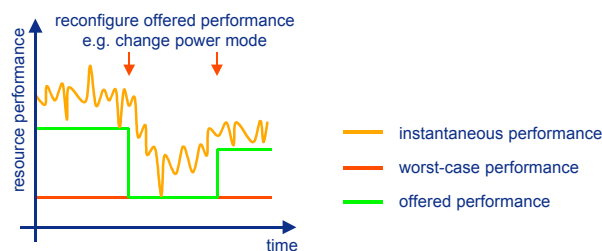


Figure 2-3. Variable resource performance and offered performance.

Computation resources, such as a MIPS, can have variable performance due to low-power (sleep) modes, which introduce a wake-up delay. In addition, techniques to reduce power consumption, such as voltage and/or frequency

scaling, give rise to multiple steady-state performance levels. As a result, the time it takes to perform a given fixed computation can vary.

Storage resources provide two examples of architectural variability: caches and volatile memories. A cache returns the requested data after a variable time. If the requested data is in the cache, it is returned quickly, but in the case of a cache miss, the data is returned after a much longer time. A cache therefore improves the *average* performance, but not the *guaranteed* (worst-case) performance. Although cache has a known deterministic algorithm, it is difficult to characterise its performance.

Volatile memories, such as dynamic random-access memories (DRAM), lose their data after some time, unless it is refreshed. The periodic refresh takes a long time, compared to a single memory access. This can result in unpredictable access times to the memory, depending on whether a read access is delayed by a refresh or not. Moreover, the order of read and write transactions, as well as the order in which transactions access the memory banks has a large impact on the memory's net bandwidth. The *nett bandwidth* is the number of user data words per second as opposed to the number of cycles per second that the memory is occupied (*gross bandwidth*). Memory controllers therefore often reorder transactions to maximise the (average) net memory bandwidth. As a result, the memory bandwidth and latency that a user experiences is dependent on his transactions (e.g. mix of reads and writes). Although the memory controller uses a known deterministic algorithm, the resulting average performance is difficult to characterise, like for caches, described above.

It is important to note that the variation in resource performance is *not intrinsic*, but is a consequence of the resource architecture. (Single-event upsets, such as alpha particles are an exception. They must be dealt with using error-correcting techniques.) The variable resource performance can be due to the resource's internal behaviour that, however, affects the user (e.g. processor sleep modes and SDRAM refresh), or it can be user dependent and difficult to capture (e.g. caches, or memory transaction reordering). In Section 3.2.2 we show examples of how architectures can be made predictable, and how their behaviour can be made to depend more clearly on the behaviour of users, using services.

2.1.3 Shared resources.

Consumer-electronics SOCs such as PNX8550 must deliver huge computational performance at low cost to the end-user. The number of computation, communication, and storage resources must therefore be minimised, and be used efficiently. Often this entails *sharing* a resource between multiple users.

For example, the programmable processors (TriMedia, MIPS) are shared between multiple tasks, and often include a real-time operating system (RTOS)

with some arbitration policy, which makes a task's execution time dependent on other tasks. Moreover, modern processors pipeline instructions, perform speculative execution, and so on. As a result, executing a given number of instructions may take different lengths of time, depending on e.g. the interleaving with other instruction streams. As a result, computing the number of (millions of) instructions per second (MIPS) as opposed to processor clock speed, may be difficult.

External memories are expensive because they raise the cost of the chip package, by introducing extra pins. In PNX8550, therefore, a single large external memory is used for the communication between streaming IPs. The scarce memory bandwidth is shared to near its capacity by the programmable processors and streaming IP. As we saw in the previous section, the memory's net bandwidth is strongly impacted by the order of read and write transactions, as well as the order in which transactions access the memory banks. The memory controller therefore reorders transactions of the multiple users to maximise the net memory bandwidth. As a result, the memory bandwidth and latency that a single user experiences is dependent on not only his, but also on other users's transactions.

Finally, shared "single-hop" communication infrastructures between multiple IP, such as busses and switches, contain a single arbiter, and face similar issues. For example, the DCS busses in PNX8550 use round-robin arbitration, and the PMA communication infrastructure uses a multi-level arbitration scheme (described in Goossens et al., 2004).

In all cases, many different arbitration policies are possible, when sharing a resource, e.g. first-come first-serve, round-robin, time-division-multiple-access (TDMA), and rate-monotonic scheduling. From the perspective of a single resource user, they introduce uncertainty regarding the resource performance. For example, the latency or bandwidth of a memory or bus may vary, depending on the behaviour of other users. This complicates the construction of a predictable SoC as a whole. Therefore, to design a predictable SoC it is helpful if shared resources use arbitration mechanisms amenable to analysis, e.g. TDMA (Rijkema, Goossens, A. Rădulescu, van Meerbergen, Wielage and Waterlander, 2003), deadline-monotonic scheduling (Audsley, Burns, Richardson and Wellings, 1991). Next to this, to facilitate reasoning about performance, each user of a resource is preferably presented with a view on the resource that is independent of other users.

2.1.4 Dependence of multiple resources.

In the example at the start of this section, the two communicating tasks each used two shared resources: the programmable processor (computation) and the external memory (storage). (The communication infrastructure is not

shared for the programmable cores.) To reason about task-to-task communication performance, we must reason about all arbiters that are involved: it is the composition of the arbiters that determines the *end-to-end* performance. For example, a bandwidth guarantee of x bytes/sec on both processors and the external memory (ignoring caches), does *not* guarantee that the end-to-end (i.e. task-to-task) bandwidth is x . In the worst case, mismatched arbitration can cause starvation, resulting in zero bandwidth. The presence of “gates” or “bridges” in an architecture (see, e.g. Figure 2-1) couple arbiters of different resources, and are an indication that these issues could arise. (In fact, the gate and bridge in PNX8550 are well-behaved (Goossens et al., 2004).)

We use the term (*inter*)*dependence* for the effect that arbiters of different resources interact in an unforeseen or unintended manner, possibly degrading end-to-end performance. All shared resources that are used by a single user must be taken into account in an end-to-end performance analysis. Several approaches tackling this analysis are being investigated (Sha and Sathaye, 1993; Richter, Jersak and Ernst, 2003; Bekooij, Moreira, Poplavko, Mesman, Pastrnak and van Meerbergen, 2004), and they rely on expressing the user behaviour (e.g. worst-case execution time) and local arbitration policies in a single formalism for end-to-end reasoning.

As a special case, when the resources are of the same type, more specialised approaches exist. In particular, NoCs are “multi-hop” communication infrastructures, meaning that they are composed of multiple routers (or switches), each with their local arbiter. Fundamentally, this leads to the problem of interfering arbiters identified above. There is a great deal of research on providing end-to-end service guarantees in computer networks (Zhang, 1995; Rexford, 1999), and NoCs (Rijkema et al., 2003; Goossens et al., 2003; Millberg, Nilsson, Thid and Jantsch, 2004; Liang, Swaminathan and Tessier, 2000) to which we return in Section 4.

For the construction of predictable SoCs, it must be possible to clearly describe and manage the interrelations and interdependencies between the behaviours of multiple resources. We believe that services, defined in the next section, provide a first step towards this goal.

2.2 Conclusions

We identified four reasons why performance verification is difficult: unpredictable resource usage, resources with variable performance, sharing of resources, and dependencies between multiple resources. These causes are independent and several of them usually act simultaneously, as we saw in the example of Section 2.1.

The first reason, unpredictable resource usage, is often externally imposed (external standards). Many algorithms, however, are, or can be made pre-

dictable, perhaps at some cost. We believe that service (levels), introduced below, can be used to characterising resource usage, in order to limit the effects of unpredictable resource requirements. The remaining reasons are due to architectural choices, which are under our own control. Service-based design, introduced in the next section, can help in making the right choices.

3. OFFERING AND USING SERVICES

In the previous section we identified four reasons why performance verification of SoCs is difficult, based on resources and resource users. In this section we describe and contrast two approaches to build SoCs: ad hoc and based on services. We motivate why we believe the latter has many advantages.

Ad-Hoc Systems The ad-hoc approach basically consists of instantiating a number of resources, adding arbiters to those that are shared. Performance verification is then difficult for the following reasons.

- To verify the performance of a SoC it must be considered in its entirety. It is not possible to consider the constituent resources in isolation because their behaviours are (inter)dependent and can interfere with one another, as we saw in Section 2.1.4.
- To accurately understand the complete SoC behaviour, current practice uses simulation of all (interdependent) resources in full detail. However, accurate simulation of the complete SoC is slow, which limits the number and length of simulations that can be performed. Moreover, simulation can only cover a small part of all possible SoC states and inputs (traces). It may be difficult to force a SoC to be in its worst state (e.g. longest latency) with simulation, especially if the worst state is unknown in advance. As a result, the observed worst case of the simulated traces can be much smaller than the real worst case. This could lead to underdimensioned resources (such as communication buffers), and a SoC that will not function correctly under all circumstances.
- If, during the performance verification process, a SoC is found to not meet its specification, a simulation trace does not necessarily give insight in how to remedy the problem. The most obvious cure, increasing the number of (shared) resources, may actually decrease the performance.
- It is not easy to make ad-hoc SoCs robust. Activation of a new user (e.g. a picture-in-picture in a set-top box) may cause a working SoC to fail completely, instead of affecting only the new user. We shall return to this issue below, in Section 3.2.1.

Below, we propose a compositional solution that is based the concept of services, to characterise and decouple the behaviour of both resources and users.

3.1 Services

We compose a SoC of resources such as processors, memories, and NoCs. These resources offer *services*,¹ which are requested and used by users. A user service request includes *attributes* to specify the desired service *level*. Examples of computation, communication, or storage service attributes are (Rădulescu and Goossens, 2004):

- Uncorrupted completion, e.g. of a write transaction to a memory, or its transport by the communication resource. If an action completes, then it is guaranteed to be correct.
- Guaranteed completion. This is not automatic; e.g. a task may be blocked until a minimum amount of memory is available, and in a NoC data may be dropped in case of congestion.
- (Minimum) capacity, e.g. amount of buffering, the number of simultaneous users of a resource.
- Ordering: is there any ordering between subsequent actions? Examples are read transactions from one master IP to multiple slave IPs, which in a NoC can come back out of order, and also multiple computations which can finish out of order on a processor.
- (Minimum) average throughput, measured in instructions per second for computation resources, and bytes per second for memory and communication resources.
- (Maximum) bound on the completion time. Guaranteed completion is defined as an unspecified completion time less than infinity; here the maximum is finite and known in advance. Examples are the latency of a read transaction on the memory, or its transport by the communication resource.
- (Maximum) variation in completion time (jitter), which is important for real-time audio and video.

These service attributes can be combined to specify a particular service level, e.g. a communication connection between two IPs could be lossless, ordered, with 100 Mbyte/sec average throughput, and with a maximum latency of 0.8 microseconds.

A resource can offer different services levels (or differentiated services, Kumar, Lashman and Stiliadis, 1998) to different users at the same time. For example, a NoC may offer communication services with different latency, throughput and jitter levels, e.g. for control traffic (low latency, low throughput) and

¹Or: resources are used to offer services. In this chapter a narrow view on services is taken, by restricting them to a single kind of resource (computation, storage, or communication). It is possible and useful to generalise services to use multiple resources, as well as lower-level services. Examples are database, printing, or secure-storage services.

streaming traffic (high throughput, low jitter). It is fruitful to offer different services levels simultaneously to increase the resource utilisation, as argued in (Goossens et al., 2003; Rijpkema et al., 2003).

Most services must be *negotiated* (Figure 2-4): a user must specify and request his desired service level from the resource. A service level describes both



Figure 2-4. Users (request service level) negotiate with resources (offer service level).

the performance offered by a resource to a user (e.g. “the NoC has only lossless, ordered connections available with at most 10 Mbyte/sec average throughput”), as well as the (potentially different) performance requested by a user (“the current task graph requires three connections with 5 Mbyte/sec average throughput but without loss or ordering constraints”). If the resource commits to the request, then the service is then guaranteed to be available until the user releases the service, when he no longer needs it. Otherwise the resource rejects the request, and the user must give up or retry with different (lower) service requirements. Note that a service is either committed to (i.e. guaranteed) or not. A resource cannot renege on its commitment.

Services must be negotiated because the resource must ensure that its capacity (storage size, instructions per second, bytes per second, etc.) is not over-subscribed, to avoid invalidating the services it has already committed to. Resources manage their number of users by performing admission control, which is why users must specify their required services in advance. Moreover, after admission, users must be prevented from using more than their allocated share of the resource (Otero Pérez, Rutten, van Eijndhoven, Steffens and Stravers, 2005).

The service concept is well established: it originated in protocol communication stacks, e.g. OSI (Rose, 1990) and has been extended to cover resource discovery, leases, etc. in approaches such as Sun’s Java Jini (Jin, 2001), and HAVi (HAV, 2000; Lea, Gibbs, Dara-Abrams and Eytchison, 2000). A *lease* is a service that is valid for a certain amount of time (we have assumed it will remain valid until the user releases it). This is more robust, in case the resource user does not correctly release resources (e.g. in the case of unreliable com-

munication between user and resource, or malicious or fault users), or when resources are inherently unreliable. Although currently not required for SoCs, we anticipate that these techniques will be applicable in the long term.

3.2 The Advantages of Using Services

We will now describe how services are used to ease each of the four obstacles to building predictable SoCs, identified in Section 2.1: unpredictable resource usage, variable resource behaviour, shared resources, usage of multiple dependent resources (Table 2-1). Table 2-2 outlines how services address each case; a fuller description is given below.

Table 2-2. Services simplify performance analysis.

	user	resource
uncertainty	<i>characterise</i> unpredictable usage	<i>abstract</i> variable performance
multiple	<i>virtualise</i> shared resources	<i>decouple</i> resource behaviours

3.2.1 Services characterise unpredictable resource usage.

Quality of service is the process whereby a trade off is made between the available resources and the requests to implement the functionality (quality) required by the user (Figure 2-4). For example, suppose that a set-top box displays a high-definition film, when the user requests a picture in picture (PIP) (Otero Pérez, Steffens, van der Stok, van Loo, Alonso, Ruíz, Bril and Valls, 2003). With the resources available in the SoC it may not be possible to honour this request. The first possible course of action (“ad-hoc,” common in personal computers) is to activate the PIP anyway. This will result in a mode where neither the high-definition film nor the PIP are displayed correctly, and the result can be anything from a “blue screen” (crashed system) to a garbled screen. Alternatively, in a service-based SoC, the quality-of-service manager requests the additional services required by the PIP (e.g. additional memory bandwidth) from the appropriate resources. If not all resources commit to the requested services, then the high-definition film and PIP can not be activated simultaneously. The SoC could inform the user that this is the case. (Note that the high-definition film has been running undisturbed throughout this process.) Another option would be to change the high-definition film to a standard-definition film (requiring fewer resources), freeing enough resources to also support the PIP.

Here we are concerned not so much with quality of service, but rather how to enable it. Services form the basis, by abstracting variable resource usage to requested services for users (see Figure 2-2), and by abstracting variable

resource performance to offered services (see Figure 2-3) for resources. We first discuss the former, the next section describes the latter.

Different service levels abstract instantaneous resource requirements of the user. This allows less frequent negotiation (“negotiated usage” versus “instantaneous usage” in Figure 2-2), at the cost of claiming too many resources. The limits of this trade off are continuous negotiation (returning to “instantaneous usage”) and worst-case design with no negotiation (“worst-case usage”). Two renegotiations are shown in Figure 2-2: the first reduces the negotiated usage, and the second increases it.

Services simplify the interface and corresponding interaction between user and resource because the requirements of the user are requested using abstract service levels, instead of detailed descriptions of actual instantaneous usage. This simplifies the implementation of resources.

Moreover, as the PIP example demonstrates, SoCs are more robust when using services because it is possible to verify in advance that a mode change will succeed, without disturbing active functions.

3.2.2 Services abstract variable resource performance.

Different service levels offered by a resource can also abstract its variable resource performance. As an example, in Figure 2-3 the instantaneous actual performance of a resource (e.g. voltage-controlled processor) may be variable and difficult to capture exactly. The offered performance therefore offers simplified view on the resource (e.g. piece-wise constant). Two reconfiguration points are shown, which could correspond to an adaptation of voltage to change processor speed.

Services offer an abstract view on resource performance, to make it simpler for users to claim the performance they desire. For example, a NoC user could ask for a connection with 100Mbyte/sec average throughput and a maximum latency of 2 microseconds. The NoC translates this abstract request for net bandwidth (user data per second) to its internal representation of gross bandwidth (which takes into account, e.g. packetisation and flow-control overhead, and the number and spacing of TDMA slots). The underlying NoC arbitration policy (TDMA or otherwise) and architecture (flow control or not), etc. that implement the services are hidden from the user because they are irrelevant to him. The translation from gross resource performance to what is offered net to the user may not be easy, as we have seen in Section 2.1.2. In Section 4 and elsewhere in this volume (Gangwal, Rădulescu, Goossens, González Pestana and Rijpkema, 2005) we describe in more detail the *ÆTHEREAL* NoC where this translation has been implemented successfully.

Abstract services also make QoS independent of particular resource implementations. QoS managers match the requested user services with the offered

resource services. After resources have committed to providing services they must not renege on its commitment, because this makes the notion of negotiation superfluous, and makes it hard for the QoS manager to offer a reliable service to end users. Taking processor power management as an example, Simunic, Boyd and Glynn, 2004 describes how resources can autonomously change their performance (e.g. frequency) to optimise a power budget. This impacts the service levels users receive. Instead resources should regulate their performance in concordance with its users. A good example is the autonomous islands of performance of Meijer, Pessolano and Pineda de Gyvez, 2004, where a resource's performance (operating frequency) is specified by the user, and the resource internally finds an optimal operating point (using adaptive voltage scaling and adaptive body bias) that guarantees the requested performance, even under (varying) environmental conditions (such as silicon processing variations, and voltage drops). Predictable system-level power and performance management can be built on top of these islands of performance (Hu and Marculescu, 2004).

3.2.3 Services virtualise shared resources.

In Section 2.1.3 we discussed how sharing a (constant-performance) resource can result in a variable performance for a single user. However, when, in a service-based SoC, a resource commits a particular service level to a user, it guarantees that the service is available to the user independent of other users of the resource. Thus, every user has his own *virtual* resource, with a performance that has been agreed upon during negotiation.

As a result, the users can be simpler because they have fewer failure modes. A user can be affected by the other users only during negotiation for a service (when the resource rejects the request), instead of any point in time (as happened in the ad-hoc implementation of the PIP example of Section 3.2.1). Services can thus isolate users from one another. This avoids the need for cooperation between users (such as required by e.g. the internet's transmission control protocol), and can make the SoC more robust against erroneous or misbehaving users (Kumar et al., 1998).

3.2.4 Services decouple usage of multiple resources.

In Section 2.1.4 we observed that when a user uses multiple shared resources, unforeseen interactions (dependencies) between these resources can affect the end-to-end performance the user obtains. The previous section showed that services decouple (or isolate) users of a single shared resource, and that each user can reason about his services independent of other users. As a result,

when a user uses multiple shared resources, he can reason about all resource reservations independently (they are decoupled).

However, as discussed in Section 2.1.4, and as is shown elsewhere (Bekooij et al., 2004), resource requirements are interdependent when end-to-end performance guarantee must be given that involve multiple resources. For example, when two tasks on different processors communicate via shared memory, the processor bandwidth, memory bandwidth, and memory buffer size are interdependent.

Although services do not remove this interdependence, there are several advantages when they are used. First, resource performance is reasoned about in terms of abstract net service levels rather than the actual detailed resource implementation. Second, users of shared resources can be considered independently because they each have their own virtual resource. Both cases reduce the complexity of the QoS manager, which can use data-flow (Bekooij et al., 2004), and other (Richter et al., 2003) techniques to compute the resource reservations that ensure end-to-end (e.g. task-to-task) performance guarantees.

3.3 Conclusions

In this section we introduced the notions of services and service levels. A requested service level serves to abstract or simplify the description variable resource requirements of a user (Figure 2-2) by hiding internal details and dynamism. Similarly, an offered service level serves to abstract or simplify the description of the variable offered performance of a resource (Figure 2-3). Figure 2-4 then shows how a QoS manager matches the requested and offered services, using negotiation. Abstract, implementation-independent services are an important enabler for effective QoS.

Services *decouple* the multiple users of a single resource (Section 3.2.3), as well as the multiple resources used by a single user (Section 3.2.4). As a result, resource users can be simpler, and SoCs can be made more robust. Although resource interdependencies are not eliminated by services, they become explicit and more abstract.

Service-based design can reduce functional and performance verification of the complete SoC in several ways. First, users and resources are specified in terms of their services. For example, a communication or storage resource can be specified to support a certain number of users with particular service levels (e.g. with net bandwidths). Following this, they can be independently designed, implemented, and their function and performance verified, because their specifications and implementations do not depend on other users or resources. Users implement their functionality making use of (building on top of) services provided by the resources. It is easier to reason about abstract services provided by resources than about their combined implementations. After

integrating the verified user and resource implementations, the SoC as a whole must be verified. Because resources are known to be correct, system verification can take place at the level of services offered by the resources, and not performed on the ensemble of all user and resource implementations (the ad-hoc approach). This compositional method is also known as *assume-guarantee reasoning* (Henzinger, Qadeer and Rajamani, 2000), because by guaranteeing the performance or behaviour of components (service providers), this guarantee can be used as a safe assumption in the performance analysis in the larger SoC using it (service users). Services naturally provide the abstraction for the guarantee step.

In the next section we will show how the $\text{\AE}THER\text{\AE}AL$ NoC can be automatically generated, programmed, and verified because it has been designed with these concepts in mind.

4. CASE STUDY: THE $\text{\AE}THER\text{\AE}AL$ NETWORK ON CHIP

The communication infrastructure is key in any platform (SgROI, Sheets, Mihal, Keutzer, Malik, Rabaey and Sangiovanni-Vincentelli, 2001) because it integrates all IP into a larger SoC, and because it is the locus of the platform communication protocols. The communication infrastructure is therefore a natural place to initiate a service-based design method. In this section we discuss how Philips's $\text{\AE}THER\text{\AE}AL$ NoC (Goossens et al., 2003) attempts to solve the issues raised in Section 2 by introducing communication services, as described in Section 3.

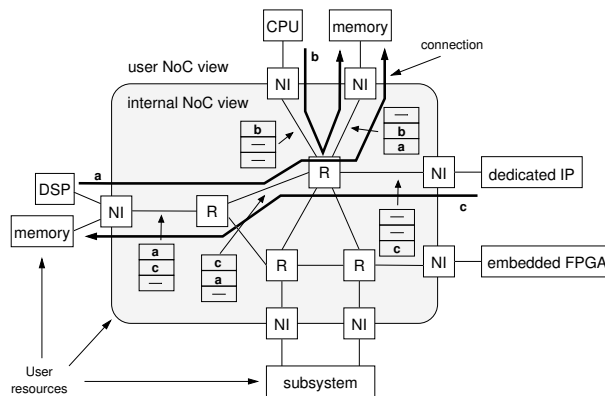


Figure 2-5. SoC composed of heterogeneous IP interconnected by a NoC.

Figure 2-5, shows the basic architecture of a NoC. There are two different points of view: (a) that of the NoC user, where the whole NoC can be seen as

a single resource providing communication services to different users, and (b) the internal NoC view consisting of multiple interacting resources.

A NoC is composed of two components: routers (R) and network interfaces (NI), see Figure 2-5. Routers transport data within the NoC. NIs convert the IP view on communication (e.g. read and write transactions) to the NoC's internal view (e.g. packets, flow control). Importantly, the NIs also implement the service abstraction, reducing the NoC's internal multiple-resource view to a NoC user's single-resource view.

In the remainder of this section we describe *ÆTHEREAL*'s service-based communication model, which comprises *best-effort* (BE) and *guaranteed-throughput* (GT) service levels. We explain their characteristics and intended uses, and how they aim to enable service-based SoC design.

4.1 The *Æ*thereal Communication Model

As discussed above, in the NoC internal view, *ÆTHEREAL* is a multi-hop interconnect, i.e. it contains multiple components (routers and NIs). Each of these components has a constant performance (e.g. every NoC link has 2Gbyte/sec bandwidth, Rijpkema et al., 2003). The NoC is shared by multiple users, who may have variable resource requirements.

ÆTHEREAL offers communication services, and comprises the *best-effort* (BE, Section 4.1.1) and *guaranteed-throughput* (GT, Section 4.1.2) service levels. These service levels have different characteristics and intended uses. The BE service level exhibits several of the problems listed in Table 2-1, whereas the GT service level does not. However, as argued in Goossens et al., 2003, it is advantageous to offer both service levels to increase resource utilisation and hence reduce cost.

Communication services are provided on connections (Rădulescu and Goossens, 2004). A connection specifies the communication between one master (e.g. the digital-signal processor DSP of Figure 2-5) and one or more slaves (e.g. distributed shared memories). Figure 2-5 shows three example connections. The user indicates the required service level per connection by specifying communication attributes, as described in Section 3.1. A BE connection offers uncorrupted, lossless, ordered communication, to which GT connections add minimum throughput, maximum latency, and maximum jitter.

As discussed in Section 2.1, the translation from the user view on performance to the NoC view on performance may be far from trivial. We illustrate this for NoCs in Figure 2-6. A user of a NoC most often reasons in terms of application data, such as bits per second of an MPEG stream ("net bandwidth"). Assuming this data is memory-mapped, the IP uses read and write transactions to access the data, and a command and address are added to the application data. The NI convert these transactions into packets, by chopping it into pieces

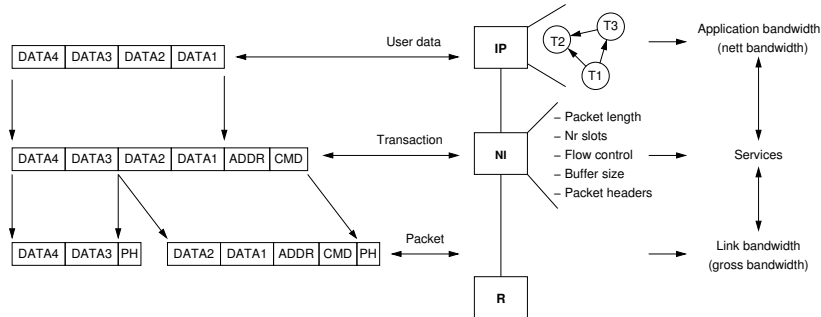


Figure 2-6. From user view to NoC view on communication.

and adding a header. Packets may be of different lengths, and the NoC may also internally generate packets that are not visible to the user, e.g. for flow control. As a result, the gross bandwidth to be claimed inside the NoC will be more than the requested bandwidth for the application data. The strength of $\text{\AE}THER\text{\AE}AL$ NoC is that the communication services include this nett to gross bandwidth translation (described in detail in Gangwal et al., 2005).

The following two subsections describe the BE and GT service levels, and how they enable the move from the internal NoC view on communication to the user's view on communication, by solving the problems described in Table 2-1 (resource sharing, interdependent resources).

4.1.1 The best-effort service level.

We first describe what the BE service level consists of, and how it is implemented. Then we list which of the problems of Table 2-1 are present, and finally motivate the reasons for offering BE service level.

BE connections implement uncorrupted, lossless, ordered data transport (transaction completion is a result of absence of data loss). This is implemented by a common NoC architecture (Rijkema et al., 2003): a packet-switched NoC, with input-queued routers using worm-hole routing and round-robin arbitration. Packets are never dropped, and credit-based end-to-end flow control is used to avoid congestion. Packet ordering is ensured by deterministic source routing.

A NoC will be shared by multiple users, and their packets may clash inside the NoC. To solve this contention, routers and NIs use local round-robin arbitration. However, when a connection uses multiple routers or NIs, the combined effect of multiple interdependent arbiters becomes difficult to characterise. Input queuing causes interdependencies between different connections (called head-of-line blocking), and worm-hole routing causes interdependencies between arbiters of different routers. As listed in Table 2-3, these are examples

Table 2-3. The best-effort service level.

problem (cf. Table 2-1)	BE service level
unpredictable resource usage	not addressed
variable resource performance	not applicable
resource shared by multiple users	local round-robin arbitration
multiple interdependent resources	local round-robin arbitration

of resource sharing and interdependent resources. Note that unpredictable resource usage is not addressed by the BE service level, and that each of the routers and NIs has a constant performance.

As a result, end-to-end (IP-to-IP) service guarantees such as throughput, latency, and jitter can not be given. Thus, only simulation can be used to correctly dimension a NoC (including its topology, buffer sizes, etc.) for given application requirements (throughput, latency, etc.).

Nonetheless, *ÆTHEREAL* offers the BE service level for a number of reasons. First, it enables a NoC to be used where user resource requirements can not be characterised well, or are highly variable. Moreover, not all applications require real-time guarantees, such as web browsing or graphics. By using the BE service level the NoC resources can be dimensioned for the average instead of worst-case communication requirements. This allows a higher resource utilisation, potentially using fewer resources, i.e. a smaller NoC. The BE service level therefore trades real-time performance for higher resource utilisation.

4.1.2 The guaranteed-throughput service level.

We first describe what the GT service level consists of, and how it is implemented. Then we list how the problems of Table 2-1 are addressed.

The GT service level adds minimum throughput, and maximum latency and jitter bounds to the BE service level. This is implemented by a NoC architecture first introduced by *ÆTHEREAL* (Rijkema et al., 2003): a global distributed TDMA arbitration scheme that emulates pipelined time-division-multiplexed circuit-switched connections. *ÆTHEREAL* implements the *global* TDMA arbitration in a *distributed* manner (using only local synchronisation). This scheme eliminates contention, and hence ensures minimal buffering in routers (one-flit input queues for worm-hole routing). Figure 2-5 shows an example NoC with three GT connections, labelled a, b, and c, with the corresponding TDMA tables and slot reservations.

A NoC will be shared by multiple users, as is the case for the BE service level. However, the GT service level avoids contention in the NoC by means of global TDMA arbitration. The same scheme also eliminates resource interdependencies due to head-of-line blocking and worm-hole routing. As a result,

Table 2-4. The guaranteed-throughput service level.

problem (cf. Table 2-1)	GT service level
unpredictable resource usage	must be characterised
variable resource performance	not applicable
resource shared by multiple users	analysable global TDMA arbitration
multiple interdependent resources	analysable global TDMA arbitration

throughput, latency, and jitter guarantees can be derived as described elsewhere in this volume (Gangwal et al., 2005). As listed in Table 2-4, this solves resource sharing and interdependent resources that plagued the BE service level. When a GT connection is requested, the required throughput, latency, and jitter must be specified, to reserve communication resources (essentially, buffers and slots in the TDMA tables), in contrast to a BE connection. Thus, the resource usage must be characterised by the user, as discussed in Section 3.2.1. Finally, note that each of the routers and NIs has a constant performance.

```
# User view: GT service-level request:
open_connection("decoder.mc", "mem.p2", "GT",
               72 Mbyte/sec, 2.5 ms, 16 byte, 72 Mbyte/sec, 1.7 ms, 16 byte)
# throughput, latency, burst size for read & write
```

```
# Internal NoC view: GT reservation:
open_connection("decoder.mc", "mem.p2",
               "GT", "22-32", "3 1 0", 33, "GT", "7-13", "1", 60)
# type, slots, path, credits for request & response
```

Figure 2-7. Service level versus GT connection reservation.

As a result, end-to-end (IP-to-IP) service guarantees such as throughput, latency, and jitter can be given on GT connections. The GT service level first enables the transition from an internal NoC view to a service-based user view (Figure 2-6). That is, the internal structure of the NoC is hidden for the user, and the collection of resources (routers and NIs) behaves as a single resource. A single global arbitration scheme (TDMA) implements resource sharing, and resource interdependencies are eliminated. Second, building on top of this, the user's (nett) requirements are translated to internal NoC (gross) resource reservations by the NoC, as advocated in earlier sections. A simplified example for a single connection is shown in Figure 2-7. A user specifies a connection from a master to a slave with required nett bandwidth and latency constraints, for given burst sizes, as shown in the top half of the figure. This is translated to the internal resource reservation view, consisting of slots, path, credits, etc., shown in the lower half of the figure.

Disadvantages of the GT service level include the need to characterise user communication requirements in advance. Between negotiation points (Fig-

ure 2-2), resources are reserved for the worst-case, potentially increasing the NoC size. The GT service level therefore trades real-time performance for possibly higher resource requirements.

4.1.3 Combining BE and GT services levels.

In the two preceding sections we have introduced the BE and GT service levels. The former aims for high resource utilisation for which it sacrifices throughput and latency guarantees. The latter aims for real-time performance guarantees, potentially at the cost of more resources (a larger NoC). By offering both service levels, *ÆTHEREAL* resources are reserved as required for GT connections, but unclaimed or unused GT bandwidth is used by BE connections. As a result, real-time (GT) services and good resource utilisation (low cost) are combined (Rijpkema et al., 2003; Goossens et al., 2003).

4.2 The *Æthetical* Design Flow

The *ÆTHEREAL* NoC design flow (Goossens, Dielissen, Gangwal, González Pestana, Rădulescu and Rijpkema, 2005) comprises design-time NoC generation (i.e. dimension and generate the NoC hardware based on user requirements), NoC configuration (compute the resource reservations from the user requirements, as shown in Figure 2-7), NoC simulation, and NoC performance verification (for GT connections). User requirements are usually stated as a collection of modes (or use cases) that the SoC must support, and NoC configuration therefore usually proceeds at the granularity of modes rather than connections. Figure 2-8 shows an example of performance verification. Connection 2 corresponds to the decoder.mc to mem.p2 connection of Figure 2-7. For each connection the computed resource reservations (number of TDMA slots), specified and available (minimum) bandwidth and (maximum) latency are shown, as well as the specified and required buffer sizes in the NoC. NoC generation, configuration, and verification for GT connections is performed on the basis of analytical models. Hence, simulation is only required if BE connections are used.

5. CONCLUSIONS

In this chapter we described and analysed the problem of performance verification of SoCs. We identified four reasons why building SoCs with predictable performance is difficult (Table 2-1): unpredictable resource usage, variable resource performance, resource sharing, and interdependent resources. We introduced the concept of a service, aiming to address these problems, and described its advantages over “ad-hoc” approaches. Finally, we introduced the *ÆTHEREAL* NoC as a concrete example of a communication resource that implements multiple service levels.

ConnId	Trans	Slot Table Size = 128		Throughput (Mbytes/sec)		Latency (ns)		BufferSize (Words)											
		Forward Allocated Slots	Reverse Allocated Slots	Spec	Avail	Spec	Max	Forward Master			Forward Slave			Reverse Slave			Reverse Master		
								Spec	Max	Slack	Spec	Max	Slack	Spec	Max	Slack	Spec	Max	Slack
0	read	11	7	72.00	104.17	2500.00	2418.00	40	40	0	33	33	0	20	20	0	21	21	0
0	write	11	7	72.00	89.46	1700.00	1584.00	40	40	0	33	33	0	20	20	0	21	21	0
1	read	11	7	72.00	104.17	2500.00	2418.00	40	40	0	33	33	0	20	20	0	21	21	0
1	write	11	7	72.00	89.46	1700.00	1584.00	40	40	0	33	33	0	20	20	0	21	21	0
2	read	11	7	72.00	104.17	2500.00	2418.00	40	40	0	33	33	0	20	20	0	21	21	0
2	write	11	7	72.00	89.46	1700.00	1584.00	40	40	0	33	33	0	20	20	0	21	21	0
3	read	18	11	120.00	161.46	2500.00	2424.00	56	56	0	54	54	0	28	28	0	33	33	0
3	write	18	11	120.00	145.62	1700.00	1572.00	56	56	0	54	54	0	28	28	0	33	33	0
4	read	11	7	72.00	104.17	2500.00	2430.00	40	40	0	33	33	0	20	20	0	21	21	0
4	write	11	7	72.00	89.46	1700.00	1590.00	40	40	0	33	33	0	20	20	0	21	21	0

Figure 2-8. Performance verification output example.

ACKNOWLEDGEMENTS

The authors thank Liesbeth Steffens and Clara Otero Pérez for their extensive and constructive feedback.

References

- Audsley, N. C., Burns, A., Richardson, M. F. and Wellings, A. J., 1991, Hard real-time scheduling: The deadline monotonic approach, in W. A. Halang and K. Ramamritham (eds), *Real-Time Programming*, pp. 127–132.
- Bekooij, M., Moreira, O., Poplavko, P., Mesman, B., Pastrnak, M. and van Meerbergen, J., 2004, Predictable embedded multiprocessor system design, *Proc. Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*, LNCS 3199, Springer.
- de Oliveira, J. A. and van Antwerpen, H., 2003, The Philips Nexperia digital video platform, in G. Martin and H. Chang (eds), *Winning the SoC Revolution*, Kluwer Academic.
- DTL, 2002, *Device Transaction Level (DTL) Protocol Specification. Version 2.2*.
- Gangwal, O. P., Rădulescu, A., Goossens, K., González Pestana, S. and Rijpkema, E., 2005, Building predictable systems on chip: An analysis of guaranteed communication in the æthereal network on chip, In this volume.
- 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, *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*.
- 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 A. Jantsch and H. Tenhunen (eds), *Networks on Chip*, Kluwer, chapter 4, pp. 61–82.
- Goossens, K., Gangwal, O. P., Röver, J. and Niranjana, A. P., 2004, 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 (eds), *Interconnect-Centric Design for Advanced SoC and NoC*, Kluwer, chapter 15, pp. 399–423.
- HAV, 2000, *The HAVi Specification. Version 1.0*.
- Henzinger, T. A., Qadeer, S. and Rajamani, S. K., 2000, Decomposing refinement proofs using assume-guarantee reasoning, *Proc. of Int'l Conference on Computer Aided Design (ICCAD)*, pp. 245–252.
- Hu, J. and Marculescu, R., 2004, Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints, *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*.
- Jin, 2001, *Jini Architecture Specification, Version 1.2*.
- Keutzer, K., Malik, S., Newton, A. R., Rabaey, J. M. and Sangiovanni-Vincentelli, A., 2000, System-level design: Orthogonalization of concerns and platform-based design, *IEEE Trans. on CAD of Integrated Circuits and Systems* **19**(12), 1523–1543.
- Kumar, V. P., Lashman, T. V. and Stiliadis, D., 1998, Beyond best effort: Router architectures for the differentiated services of tomorrow's internet, *IEEE Communications Magazine* pp. 152–164.
- Lea, R., Gibbs, S., Dara-Abrams, A. and Eytchison, E., 2000, Networking home entertainment devices with HAVi, *IEEE Computer* **33**(9), 35–43.
- Liang, J., Swaminathan, S. and Tessier, R., 2000, aSOC: A scalable, single-chip communications architecture, *Proc. Int'l Conference on Parallel Architectures and Compilation Techniques (PACT)*.
- Meijer, M., Pessolano, F. and Pineda de Gyvez, J., 2004, Technology exploration for adaptive power and frequency scaling in 90nm CMOS, *Proc. Int'l Symposium on Low Power Electronics and Design (ISPLED)*, pp. 14–19.
- Millberg, M., Nilsson, E., Thid, R. and Jantsch, A., 2004, Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip, *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*.

- Otero Pérez, C. M., Steffens, L., van der Stok, P., van Loo, S., Alonso, A., Ruíz, J. F., Bril, R. J. and Valls, G., 2003, QoS-based resource management for ambient intelligence, *in* T. Basten, M. Geilen and H. de Groot (eds), *Ambient Intelligence: Impact on Embedded System Design*, Kluwer, pp. 159–182.
- Otero Pérez, C., Rutten, M., van Eijndhoven, J., Steffens, L. and Stravers, P., 2005, Resource reservations in shared-memory multiprocessor SOCs, *In this volume*.
- Rexford, J., 1999, *Tailoring Router Architectures to Performance Requirements in Cut-Through Networks*, PhD thesis, University of Michigan, department of Computer Science and Engineering.
- Richter, K., Jersak, M. and Ernst, R., 2003, A formal approach to MpSoC performance verification, *IEEE Computer* **36**(4), 60–67.
- Rijpkema, E., Goossens, K., A. Rădulescu, J. D., 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, *IEE Proceedings: Computers and Digital Technique* **150**(5), 294–302.
- Rose, M. T., 1990, *The Open Book: A Practical Perspective on OSI*, Prentice Hall.
- Rădulescu, A. and Goossens, K., 2004, Communication services for networks on chip, *in* S. S. Bhattacharyya, E. F. Deprettere and J. Teich (eds), *Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation*, Marcel Dekker, pp. 193–213.
- SgROI, M., Sheets, M., Mihal, A., Keutzer, K., Malik, S., Rabaey, J. and Sangiovanni-Vincentelli, A., 2001, Addressing the system-on-a-chip interconnect woes through communication-based design, *Proc. Design Automation Conference (DAC)*, pp. 667–672.
- Sha, L. and Sathaye, S. S., 1993, A systematic approach to designing distributed real-time systems, *Computer* **26**(9), 68–78.
- Simunic, T., Boyd, S. P. and Glynn, P., 2004, Managing power consumption in networks on chips, *IEEE Transactions on VLSI Systems* **12**(1), 96–107.
- Zhang, H., 1995, Service disciplines for guaranteed performance service in packet-switching networks, *Proceedings of the IEEE* **83**(10), 1374–96.