Chapter 4

GUARANTEEING THE QUALITY OF SERVICES IN NETWORKS ON CHIP

Kees Goossens, John Dielissen, Jef van Meerbergen,
Peter Poplavko[†], Andrei Rădulescu, Edwin Rijpkema,
Erwin Waterlander, and Paul Wielage
Philips Research Laboratories, Eindhoven, The Netherlands
[†] *Technical University of Eindhoven, Eindhoven, The Netherlands*
Kees.Goossens@philips.com

Abstract Users expect a *predictable quality of service* (QOS) of embedded systems, even for future, more dynamic, applications. System-on-chip designers use networks on chip (NOC) to solve deep submicron problems, and to divide global problems into local, decoupled problems. NOCs provide services through protocol stacks, and introducing *guaranteed* services enables IP re-use and platform-based design. It also provides globally predictable behaviour, as required by the user, when combining local, decoupled solutions. There are several levels of QOS commitment (correctness, completion, completion bounds), with increasing cost. A combination of guaranteed and best-effort (no commitment) services combines their respective attractive features: predictable behaviour, and good average resource utilisation. The ÆTHEREAL NOC is an example of this approach, and forms the basis of a QOS-based design style, as advocated in this chapter.

1. Future applications and systems on chip

In this section we raise the following question: *why and how must systems implementing future applications guarantee the quality of service* (QOS)? We look at each of the components in turn, in the context of embedded systems and systems on a chip (SOC). We then observe that new SOC architectures and design methods divide global problems into local ones, and rely on networks on chip (NOC) to compose local solutions. We answer the question, in Section 2, by presenting a synthesis of a QOS-based design style and networks on chip. Section 3 explains that QOS can be decomposed into several levels of commitment, with different resource requirements. In Section 4 we introduce the ÆTHEREAL NOC, and compare it to other NOCs, in Section 5.

1.1 Characteristics of future applications

We are witnessing the *convergence* of previously unrelated application domains: computation (personal digital assistants, mobile computing), communication (telephone, videophone, networking), and multimedia (audio, photography, video, augmented and virtual reality). Convergence leads to increased functionality and heterogeneity, as previously unrelated functions are combined. Systems become more dynamic, or even unpredictable, as new algorithms seek to take advantage of the disparity between average and worst-case processing. Compressed audio and video data of MPEG2 is an example. Algorithms are also shifting to higher semantic levels, and the semantic complexity of data is less predictable than its volume. An example is the shift from constant pixel processing (scaling, edge enhancement, and so on), to MPEG4 object detection and synthesis, to face recognition and scenario detection. Finally, increased interaction with the environment also makes systems more dynamic. System functionality can be influenced by the current location (affecting communication or computation capabilities), environmental conditions (precipitation and multipath interference can affect mobile communication), and other systems in the proximity (e.g. car guidance, ad hoc networking).

Trends such as ambient intelligence and the networked home make future applications *embedded* and *pervasive*. Moreover, applications acquire a responsibility for the control of physical objects, such as home heating systems, cars, and so on. Real-time performance and safety are critical in many of these applications.

Below, we examine how users interact with these applications, and then what is required to design these systems.

1.2 Quality of service: a user view

Users expect a certain behaviour of applications; in other words, they must be *predictable*. While those expectations may be low, as is often the case for personal computers, a certain fitness for purpose is always assumed. Consumer electronics are subject to higher demands: a television must have a robust user interface and is not allowed to crash or be unresponsive. Expectations are stricter yet for real-time applications (e.g. involving audio and video, or control systems); a television must display at least 50 pictures of a constant quality per second, for example. The essence of quality of service (QoS) is therefore the offering of a predictable system behaviour to the user.

A central question is how to reconcile the dynamic, unpredictable nature of future applications with the requirements for predictable services.

1.3 Implementing future applications on chip

Having identified the characteristics of future applications and their QoS requirements, we now turn to the question of their implementation in SOCs. Moore's law describes the *exponential growth* over time of the number of transistors that can be integrated in an IC. It predicts that chips in 2010 will count over 4 billion transistors, operating in the multi-GHz range [1]. It is this abundance of computational power that has fuelled the convergence of application domains described above. However, Moore's law is only a prediction, and two major obstacles must be solved to make it a reality [2]. First, to use future VLSI technologies, several *deep-submicron problems* must be solved: the increasing disparity between transistor and wire speeds, power delivery and dissipation, and signal integrity. Second, the intrinsic computational power of an IC must not only be used efficiently and effectively, but the time and effort to design a system containing both hardware and software must also remain acceptable. The so-called *design productivity gap* states that the increase in our ability to design SOCs does not match Moore's law. To close the gap, system design methods to implement applications with the latest VLSI technology (including the definition of architectures, mapping applications to architectures, and programming) must harness exponential hardware resource growth in a scalable and modular manner.

The following two examples show that until recently, architectures and design methods at both the deep-submicron and system levels have been often been *global* in nature. This hampers scalability, and in the next section we show that approaches that compose local solutions are becoming popular.

The physical communication between intellectual property blocks (IP) has made use of a mix of local and global wires. When timing constraints of the design must be verified, IPs cannot be checked independently because they are interconnected; correcting a timing violation in one IP may invalidate the timing of another. This process does not necessarily converge to a solution, and is design specific (not re-usable). This global timing closure problem results from a tight (timing) coupling [3].

The second example considers the model of time. Until now, the dominant design style has been synchronous; a global notion of time has been implemented by globally synchronous clocking. Increasing processing variations on a single IC make this style untenable in the future [4]. This will impact clocking regimes, but also the programming model. When using shared memory, which has been the dominant method to communicate between tasks, if tasks are not tightly synchronised (requiring a common, global time frame) then task scheduling may lead to interference and unpredictable behaviour.

1.4 Composing local subsolutions

To avoid the exponential complexity of global methods and solutions, there is increasing interest in subdividing global problems into *local, decoupled* problems, and then composing the local solutions. This requires, foremost, *compositionality* to assemble a global solution from local ones (Figure 4.1). But solutions must also be *scalable* (and likely hierarchical, Figure 4.1(c)) because there will be an exponential number of local solutions. This applies at all

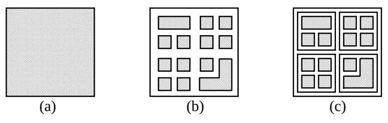

Figure 4.1. Divide global problem (a) into local problems that are composed flat (b) or hierarchically (c). Examples are GALS, local timing closure, local shared memory with global fifo communication or message passing, local busses or switches with global packet switching.

levels of SOC design: lay-out, timing verification, clocking, and programming. This trend commenced some time ago with IP-based re-use, platform-based design [5]. The interest in decoupling and composing local solutions can be observed in hierarchical lay-out and wiring, local clocking strategies such as GALS (globally asynchronous, locally synchronous) [6], software-programmable fixed IPs or tiles [7, 8], and synthesisable tiles for dedicated silicon [9] or FPGAs [10, 11]. At the task level, these scalable system architectures (including e.g. chip multiprocessing [12, 13, 14]) use tasks with local, independent address spaces and time frames that are composed by means of timing-independent fifo channels (for example, Kahn process networks for stream-based processing).

1.5 Networks on chip

All the approaches that advocate local solutions have a common reliance on a scalable and compositional communication medium to efficiently combine the large number of (hardware and software) IPs or subsystems in a working system. This challenge is addressed by networks on chip (NOC), which therefore play a pivotal role in future SOCs. NOCs help to solve both deep-submicron problems (timing closure, wiring, lay-out, etc.) and compositional design methods (services and protocol stacks). More detailed argumentation for the use of NOCs in future SOCs can be found in [15, 9, 16, 17, 2], and elsewhere in this volume.

In the following section we show that NOCs and QoS naturally combine to solve the two problems stated in this section: NOCs combine decoupled local solutions, and a service-based design style makes both the design process and the QoS of the resulting SOC more predictable.

2. NoCs and QoS: a synthesis

We advocate system design centred on NOCs and QoS for three reasons. First, services relieve the inherent tension between the *dynamic nature of future applications* and the user requirement for predictable services. This addresses concerns raised in Sections 1.1 and 1.2. Second, IP re-use and platform-based design aim to re-use applications and architectures by decoupling them. The use of NOCs and their associated network protocol stacks are a way to achieve this (cf. Section 1.5). Finally, it is our tenet that QoS, in particular *guaranteed services*, allow us to move towards *globally predictable, locally predictable* methods and solutions for SOC design. This, in combination with NOCs solves a need of recent architectures and design methods, introduced in Section 1.4. We discuss each point in turn.

2.1 Dynamic applications and predictable QoS

A QoS-based design style alleviates the demanding task for the system *designer*. He or she must plan systems that reliably provide users with the expected QoS for future applications. Moreover, systems must always be cost effective. Higher costs are acceptable when safety is critical, but they are always under pressure for consumer products, whether they be television sets or cars. Systems, especially SOCs, therefore have limited resources, and they must be shared and managed as the application unfolds its dynamic behaviour. Resource management depends on two phases: negotiation to obtain resources, followed by a steady state in which allocated resources are used. As applications become more dynamic, the first phase, *renegotiation*, will become more frequent. Thus, users can be given a predictable QoS by giving resource management and QoS a prominent place in system design.

2.2 Platform-based design, NoCs, and QoS

The aim of *platform*-based design [5, 18] is to reduce the cost of system design through re-use of applications and architectures. A platform decouples applications and system architectures, by defining a template architecture and programming model. In other words, by limiting the freedom in which an application can be implemented on an architecture, the interdependence of application and architecture is concentrated and reduced (Figure 4.2(a)). However, the convergence of applications entails an increasing diversity and dynamics in resource usage (such as communication and computation patterns), and this results in an increasing need for *differentiated services* [15]. It is therefore important that the platform offers the appropriate services (for communication, computation, power management, etc.). As a result, the communication infrastructure is a critical component of a platform, because it must solve the

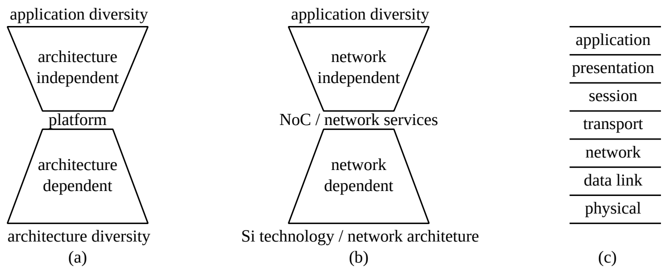

Figure 4.2. Decouple applications and architectures by means of a platform (a), or network (e.g. NOC, Internet) services (b), or network protocol stacks (c).

apparent contradiction of implementing diverse application behaviours with application-dependent IPs in an application-independent manner. Using a NOC for the platform interconnect tackles both problems: it integrates heterogeneous IPs in a standard fashion (in other words, the NOC services largely define the platform, Figure 4.2(a)); and it naturally provides differentiated services by means of a (partially application-dependent) protocol stack [17].

Figure 4.2(b) shows that NOCs offer a small set of services ("NoC services"), on top of which different kinds of communication can be implemented (e.g. shared-memory cache-coherent traffic, message passing). The OSI transport layer [19] is a natural place to separate the application and architecture, because it is the first network-independent layer. The set of services should be small, yet allow the upper layers in the protocol stack to offer differentiated services [20]. This method has been used successfully in computer networks [21], where the internet protocol (IP) plays a similar role. Higher layers offer more specialised protocols such as TCP and UDP, and FTP and HTTP. For NOCs, there are several proposals [22, 23, 24], but convergence has yet to occur.

Concluding, NOCs help IP re-use and platform-based design to combine IPs, and NOC services aid in decoupling applications and architectures.

2.3 QoS enables predictable composition

As we have seen in Section 1.3, to fully exploit Moore's law, it is key to combine and control many local, perhaps autonomous, components in an efficient and flexible manner resulting in the required QOS. To localise timing, communication, data, scheduling, and so on, means that components (subsystems, IPs, tasks, threads, etc.) must first be identified. After composition, their interaction is controlled by means of arbitration, scheduling, or resource management. But we must address the apparent contradiction of providing *globally predictable* QOS (cf. Section 2.1) while current approaches tend to *globally unpredictable, locally predictable* (GULP!) regimes. How can global timing guarantees be given when locally synchronous components are combined asynchronously [6]? How can global performance be quantified when tasks are combined in a latency-insensitive fashion [14, 7]? It is our tenet that QoS, in particular guaranteed services, allow us to move towards *globally predictable, locally predictable* methods and solutions. We elaborate this claim below, by first defining what role QoS plays, then stating how the local solutions (e.g. IP design) benefit, and finally how QoS eases the composition of local solutions.

2.3.1 Services. Effective steering of the limited resources in a NOC (cf. Section 2.1), and hence effective QoS, is contingent on a reliable reaction of resources to instructions. In other words, a predictable or guaranteed behaviour of system components is a prerequisite to providing the user with expected QoS, at whatever level. The essence of QoS-based system design is to restrict the interaction between components to well-defined services. Two phases can be distinguished: (a) negotiation, followed by (b) resource steering or scheduling, and observation or inspection. As an example, consider a microprocessor which negotiates a connection to a memory with guaranteed bandwidth, but without a constraint on transaction ordering. The NOC that offers communication services will honour the request if has sufficient resources available, and will reject it otherwise. Later, the microprocessor may renegotiate its connection, e.g. to a higher bandwidth. The NOC will release the resources of the connection for use by other connections, when it is closed by the microprocessor.

2.3.2 Advantages for local solutions. QoS-based interaction has a number of advantages for IP design. By limiting component interaction to a set of well-defined services, their interfaces are simplified because there are fewer eventualities to take into account. Moreover, failures of service users (IPs) are concentrated at the reconfiguration points: after a service provider (such as a NOC) has committed to the request (such as a connection with bounded jitter), its provision can be relied upon. Similarly, IPs do not interfere with each other because the service provider and user have a local contract. This allows components to be designed and implemented in isolation. An advantageous side effect of negotiation is that the requirements of IPs must be stated explicitly, aiding the design process.

2.3.3 Advantages in composing local solutions. We now consider the impact of QoS-based interaction when combining IPs to obtain a SOC. First, services that are guaranteed to an IP are not affected by other IPs in the network, making reasoning about the IP in isolation possible. This is essential for a *compositional construction* (design and programming) of SOCs.

Moreover, SOCs can be more *robust*, because rather than relying on cooperation to share resources, resource management enforces the contracts between service providers and users. An IP that (maliciously or erroneously) does not adhere to a cooperative protocol, such as TCP/IP, cannot, therefore, disturb the system as a whole [25]. Failure is therefore local in space (one IP) and time (at one of its reconfiguration points).

Next, when components are combined, the performance (i.e. the QoS) of their composition must be validated. This can be done by analysing the complete system implementation. However, this analysis may be too hard, because the behaviour of individual components or their interactions are complex to model accurately (e.g. traffic analysis of NOCs [26], cache behaviour [27]). *Statistical* approaches are therefore frequently used [28]. Unfortunately resource requests often do not fit models (e.g. bursty traffic versus fractal or normal traffic distributions [26]) invalidating the verification. The oxymoron "statistical guarantee" does therefore not guarantee a QoS, but implies a (usually post hoc) analysis relying on a statistical model of the resources and their usage. Instead, we propose to validate the composition of the local solutions at the level of services (their interface), instead of having to open them up, and consider their (global) combination. This *abstraction* is essential in reducing the verification state space.

Finally, services can make QoS provision *architecture independent*, because how a component, such as a NOC, offers its services is not relevant. This removes the need to second-guess the inner workings of a component, because its services describe all that is required to know. Interaction becomes prescriptive (state what is required, i.e. what must happen), rather than prognostic (try to predict the service provider's behaviour) or reactive (act on how the service provider behaves).

We can therefore conclude that by abstracting local IP behaviours to their service requirements or provision, makes the global QoS of their composition more predictable and hence easier to reason about. This simplifies the design of SOCs.

3. On the cost of guaranteeing QoS

We have seen that a QoS-based approach is required to implement future applications, and to offer a predictable performance to users. In essence, offering a QoS requires a commitment. In this section, we present different levels of commitment, and their effect on predictability and cost in terms of resource usage. Although guaranteed services, which offer commitment, have many advantages over so-called best-effort services, which offer no commitment, we show that their combination is beneficial. The ÆTHEREAL NOC, described in the next section, is an example of that approach.

3.1 Different levels of commitment

As has been explained before, to offer a certain QoS with finite resources, the service provider and user negotiate to arrive at a contract, i.e. a *commitment* by the service provider to honour the request. If a commitment has been given, the service is *guaranteed*, otherwise it is a *best-effort* service. Commitment exists at several levels: 1) *correctness* and integrity of the result, if and when it is delivered. Examples are parity checking and error correcting codes for uncorrupted data transmission, and redundant computation with majority voting for safety-critical systems. 2) Promise of *completion* or delivery. This involves the cumulative availability (over time) of sufficient resources (e.g. memory, cpu cycles), and their performance (e.g. absence of deadlock or livelock). Note that a minimum number of resources may be required: for example, an in-place FFT needs to store all samples simultaneously, and a mobile communication may require a minimum battery charge to generate a sufficiently strong signal. 3) *Bounds* on the performance. Examples are the completion time (when is the result available), cumulative time to completion and its variations (e.g. bounded latency and jitter), and (peak and average) energy consumption.

Almost any form of commitment to progress leads to resource allocation (e.g. memory space, cpu cycles, communication bandwidth, battery power) and hence requires resource management. Moreover, levels of commitment depend on those below them. We illustrate this with two examples.

An IP connected to a NOC may not always be able to accept incoming data. Assuming its input buffers are finite, several solutions are possible, with direct consequences for the service level that can offered. Packets that arrive at a full buffer are dropped, instantly precluding completion bounds such as latency and jitter guarantees. General computer networks typically use this approach. Alternatively, packets are not dropped and are left waiting in the network. Care then has to be taken to not introduce deadlock (e.g. if re-ordered packets that wait for free buffers do not overtake each other), or livelock (e.g. in deflection routing [29] packets keep moving, but may never arrive at their destination). In both cases, again, completion bounds may be in jeopardy. An approach to offer completion bounds (e.g. for a bounded communication latency) is to ensure packets never wait in the network by reserving buffer resources at the receiver, and by using end-to-end flow control to constrain the sender to never send more than the available space.

The second example concerns data transmission in unreliable media. This is a topic of importance in NOCs because wires suffer increasingly from interference, such as cross-talk and voltage drops. To ensure data is transported unchanged, it can be retransmitted when corrupted, or error correction can be used. The energy efficiency of both approaches is investigated by [30] for unreliable busses. What concerns us here, is that using retransmission takes a variable, possibly unbounded, amount of time, whereas error correction takes place in a constant time. Therefore, time-related guarantees, such as minimum throughput, can only be given by the latter.

We conclude that the strictest guarantees, namely those involving performance bounds, pervade the system (all protocol layers): they cannot be grafted on as an afterthought [17, 31]. In NOCs we should and can choose communication protocols and styles to offer the required services, but, as the examples show, this requires vigilance.

3.2 Commitment and resource usage

The effect of the level of commitment on resource usage is illustrated in Figure 4.3. Suppose that the resource plotted vertically is bandwidth. For

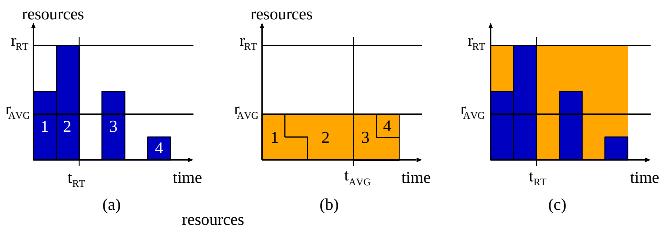

Figure 4.3. (a,b) Commitment to completion or bounded completion is reflected in resource usage. (c) Combining guaranteed services with best-effort services, discussed in Section 3.3.

real-time performance, the requested bandwidth must be offered in the same interval, see Figure 4.3(a). Thus, as many resources must be available as the largest request (r_{RT} at time t_{RT}). With resources dimensioned for the average resource usage (r_{AVG} in Figure 4.3(b)), completion times may shift to the future: the peak resource request at time t_{RT} is only completed at time t_{AVG}, in the example. Depending on the required completion bound and variability in resource requests more or fewer resources can be added. However, more resources than r_{RT} do not improve performance, while fewer resources than r_{AVG} mean that no completion commitment can be given, because the backlog of requests keeps growing.

Architectures offering only best-effort services do not reserve resources, and hence can have a *better average resource utilisation*, at the cost of *unpredictable or unbounded worst-case behaviour*.

3.3 Both best-effort and guaranteed services

A service is guaranteed if a commitment is given, and best effort otherwise. This holds for individual services, not their ensemble. For example, data transport may be uncorrupted (commitment to correctness), and lossless (commitment to delivery), and without throughput guarantees (best-effort throughput, i.e. no commitment to a completion bound). Moreover, a given service can be offered both with and without commitment, to flexibly use the available resources. For example, a data transport service can be offered both with and without completion bounds, to serve different users, in a single SOC. In Figure 4.3(c), the critical dark traffic, uses the guaranteed service, for real-time performance. The remaining $r_{\mathrm{RT}} - r_{\mathrm{AVG}}$ resources are, on average, available to other traffic, for example with less strict completion bounds, or with no completion bound (best-effort completion).

Figure 4.3(c) shows that when resources must be dimensioned for the worst-case for a given service commitment (e.g. guaranteed latency), the resources can also be used to give less stringent commitments (e.g. guaranteed delivery), or even a best-effort service. A combination of best-effort and guaranteed services gives the advantages of guaranteed services to only part of the system, but the available resources are used more efficiently. In the next section we show how this can be put in practice, in the ÆTHEREAL NOC.

4. The Æthereal network on chip

In this section we introduce the ÆTHEREAL NOC [17, 2]. It offers guaranteed services to obtain the advantages in composability, robustness, and so on of QOS-based design. These services are used for real-time and critical functions. The ÆTHEREAL NOC also provides best-effort services, to take advantage of their lower resource requirements and potentially better average performance. We describe the NOC services [24] in the next section.

In Section 4.2, we show how the services can be efficiently implemented, using a mix of time-division-multiplexed circuit switching, and packet switching [32, 31]. The programming model (connection creation and closing) and its advantages are also explained.

4.1 Æthereal Services

The ÆTHEREAL NOC offers differentiated services with *connections*. A connection describes communication between one master and one or more slaves, with an associated service level, such as fifo transaction ordering, and maximum latency. Connections must be *created* stating the requested service level; this is the negotiation phase. The NOC either accepts or rejects the request for the connection. Connection acceptance may lead to resource reservations in the NOC, e.g. buffers or a link bandwidth percentage. After usage, *closing* a connection frees the resources. Different connections are created and closed independently, possibly at different points in time. Configurations can be computed at compile time (i.e. off-line), or at run time. To ensure that the programming model scales as NOCs become larger, negotiations can be distributed.

4.1.1 Transactions.

Once a connection has been created, the master initiates *transactions* by means of *requests* which zero or more slaves *execute*, perhaps leading to a *response*. Examples of transactions are read, write, acknowledged write, test and set, and flush. By offering these transactions the ÆTHEREAL transaction model is similar to existing bus protocols, to ease migration of IP from current interconnects to NOCs. However, to be able to take full advantage of increased NOC performance, transactions can also be pipelined, split, and posted [24].

4.1.2 Connection Types.

The ÆTHEREAL NOC can offer three kinds of connections (Figure 4.4). A *simple* connection contains a master and

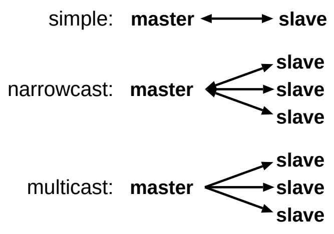

Figure 4.4. Connection types.

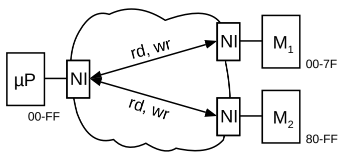

Figure 4.5. A narrowcast connection.

a single slave. The master initiates transaction which the slave executes, possibly resulting in a response (e.g. for a read). On a *narrowcast* connection, containing a master and one or more slaves, the request from the master is sent to and executed by exactly one slave. An example of the narrowcast connection is shown in Figure 4.5, where the master performs transactions on an address space that is mapped on two memory modules. Depending on the transaction address, a transaction is executed on one of the two memories. A *multicast* connection is a connection between one master and one or more slaves, in which requests are duplicated and each slave receives a copy of those requests. Currently, in a multicast connection no return messages are allowed, because of the volume of traffic they may generate (i.e., one response per destination), and the increased complexity in the master (because individual responses from slaves must be merged into a single response).

4.1.3 Connection Properties.

We distinguish the following services, or connection properties: 1) data integrity, 2) transaction ordering, 3) transaction completion, 4) connection flow control, and 5) connection throughput, latency, and jitter. A connection can request any combination of these properties (e.g. a throughput guarantee, flow control, but no transaction ordering). Recalling the levels of commitment, of Section 3.1, properties 1 and 2, above, commit to correctness (including order) of results. Completion is guaranteed by property 3 and 4 (4 implies 3). Connection latency and jitter give completion bounds. We discuss each property in turn.

1) *Data integrity* means that data is transported unchanged. We assume that data integrity is solved at the data link layer; every connection therefore offers data integrity. However, as noted in Section 3.1, this must be done in a way that supports higher-level commitments, in particular bounds on transport completion (latency).

2) *Transaction ordering.* Transaction orders are only defined on a single connection; transactions on different connections may be transported and executed in any order. In general, responses and requests may be reordered during their transport in the network. This means that requests (responses) may arrive in a different order at a single slave (master) than they were sent. (Note that we refer to requests and responses of different transactions. Within a single transaction, requests and responses are always ordered as follows: the master sends a request, a slave receives the request and executes it, the slave sends the response, the master receives the response.) We refer to [24] for a detailed analysis of different orderings. Here it suffices to state that three connections orderings are useful. *Unordered* connections, in which no order is assumed between any request and response. *Locally ordered connections*, where requests sent by the master for each slave are delivered to that slave in order they were sent. Requests for different slaves are unordered. Responses are delivered to the master in the order the requests were generated. For example, on a narrowcast connection with multiple single-port memory slaves, the read and write order to a single memory is important, but not between memories. *Globally ordered connections,* in which requests for (responses from) all slaves are delivered to the slaves (master) in the order the requests were sent. This last ordering is not offered as a service, because local ordering is usually sufficient, and because the user can emulate global ordering by means of acknowledged transactions. Global ordering may be used when the order in which different devices (slaves) are programmed is of importance.

3) *Transaction completion* stipulates that transactions requesting a response (e.g. acknowledged write) guarantee that the master always receives a response. The response contains a) a report that the request was not delivered to the slave, or b) the response of the slave, after it successfully executed the request, c) a notification that the slave successfully executed the request, but the response was dropped, d) a report that the slave failed to execute the request. For connections with flow control, no data will be dropped, and transaction completion is automatic. Without flow control fewer resources are required in

the network, but data may be dropped. For example, it may be fine to drop a transaction, as long as you know it has been lost, so that you can resend it.

4) *Connection flow control* guarantees that data that is sent will fit in the buffers at the receiving end. End-to-end flow control is one of main techniques to avoid network congestion, if data cannot be dropped (cf. Section 3.1). If a slave is slower than its master, and the slave buffers fill up, then the master will be blocked until there is sufficient free space for a transaction. Similarly, a slave may be blocked by a slower master on the return path.

5) *Bounds on connection throughput, latency, and jitter.* The ÆTHEREAL NOC provides connections that guarantee a bandwidth per fixed time interval. Thus, a combined throughput, latency, and jitter guarantee is given. Depending the time interval, the latency and jitter bounds may be rather large. This is acceptable in many applications, such as audio and video streaming, where throughput is more important than latency. In the next section give the reasons for giving the throughput guarantee in this form.

The ÆTHEREAL services include transactions of bus protocols, to ease migration from current bus-based systems to NOCs. But where in OCP [23] and VCI [22] connections are used only to relax transaction ordering, we offer a more general connection property model. Not only are more properties considered, but the request and response communications can be independently configured (e.g. for flow control, and throughput guarantees), allowing more fine-grained resource management. Connection-based service provision therefore allows better differentiation of services, and allows users to make full use of NOC performance.

4.2 Æthereal architecture

The challenge of designing a NOC lies in finding a balance between the NOC services and their implementation complexity and cost. Moreover, as has been mentioned in Section 3.1, different services (levels of commitment) are dependent on each other. Offering time-related guarantees (a completion bound) influences the NOC to the core. The ÆTHEREAL NOC has both guaranteed and best-effort services, and an architectural challenge is how to combine these efficiently. In other words, how can *guaranteed worst-case behaviour be joined with good average resource usage*. In the following sections we describe two conceptually disjoint networks that each solve one part. An efficient combination is then presented. Both networks contain two components: routers and network interfaces. Routers transport data and can be described at the OSI network layer. As the ÆTHEREAL services are offered to IP at the transport layer and are end to end (master to slave and vice versa), network interfaces are required to bridge the network layer and transport layer views on communication.

4.2.1 A guaranteed-throughput router. To give time-related guarantees on a connection, such as throughput guarantees (on a finite time scale) or latency bounds, the interference of other traffic in the NOC must be limited and characterised. Circuit switching gives strong guarantees but at a high cost: connections have to be dimensioned for the worst case traffic. Time-division multiplexed circuit switching reduces this cost, but creating and closing of circuits still takes much time, and grows with the size of the network. The life-time of circuits must grow to amortise this cost [33]. Rate- and deadline-based scheduling [34, 35] can characterise the worst-case contention and offer bounds on latency, by regulating the inflow of data, but at the a high buffer cost.

The ÆTHEREAL NOC uses *contention-free routing*, which is based on a time-division-multiplexed circuit-switching approach, where one or more circuits are set up for a connection, which is assume to be relatively long-lived. Guaranteed throughput (GT) packets never use the same link at the same time, i.e. all contention is avoided. This can be achieved by controlling both the time GT packets enter the network, and their speed in the network. All routers logically have a common notion of time, embodied in a slot counter (see Figure 4.6). GT packets propagate at the rate of one router per slot counter increment. By regulating the time a GT packet is injected in the network, it is in effect scheduled to use each successive link in its path in a successive slot, see Figure 4.6. This method avoids the emptying and filling of circuits between

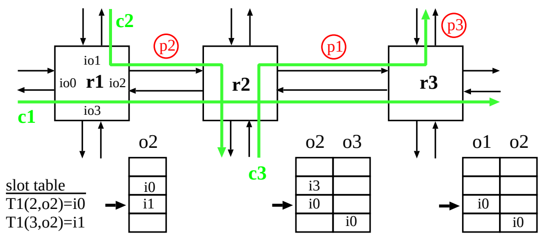

Figure 4.6. Three routers, all in slot $s = 3$, switch packets p_i of circuit c_i to outputs o_i, for $T(s, o_i) = c_i$. For each table, only the relevant columns (outputs) are shown. Input/output link pairs are labelled *io*$_i$.

switching of circuits. Conceptually it resembles input queuing with store-and-forward routing, which results in low buffering costs because GT packets are small. The end-to-end (network interface to network interface) latency is the number of hops multiplied by the size of the GT packet. This is minimal, once the packet has entered its slot. This model also allows multicast circuits. GT packets are headerless, and are routed by means of slot tables in every network interface and router. Section 4.2.3 explains how the slot tables are programmed.

4.2.2 A best-effort router. The best-effort router uses packet switching and has a more conventional structure. Our experiments [31] indicate that both input queueing with worm-hole routing or virtual-cut-through routing, and virtual output queueing with worm-hole routing are feasible, in terms of buffering costs. Input queueing uses fewer buffers, but suffers from head-of-line blocking. Virtual output queueing has a higher performance but at the cost of more buffers.

4.2.3 A combined GT-BE router. The logically separate guaranteed (GT) and best-effort (BE) routers are combined (Figure 4.7(a)) to share the router resources (e.g. switch and data path, Figure 4.7(b)), and to obtain the advantages of both. The GT router offers a fixed end-to-end latency for its traffic, which has the highest priority, enforced by the arbiter. The BE router

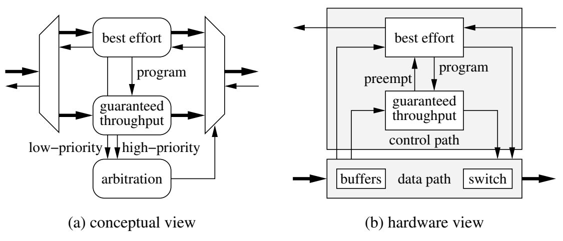

Figure 4.7. Two views of the combined GT-BE router.

uses all the bandwidth (slots) that has not been reserved or is not used by GT traffic. This allows the sharing of links and data path. Resources are therefore never left unused, when there is data, cf. Figure 4.3(b). They are either used for critical traffic with real-time requirements (for which a completion bound has been given), or for best-effort traffic (without a completion bound).

To allow distributed and scalable programming of connections, the GT router slot tables (cf. Figure 4.6), are programmed by means of BE packets, see the arrow "program" in Figure 4.7(a&b). This can be done in a pipelined and concurrent (multiple simultaneous negotiations, also from the same the source), and distributed (active in multiple routers) fashion. Negotiations, resulting in slot allocations, can be done at compile time, and be configured deterministically at run time. Negotiations can also be done at run time, centrally or distributed.

4.2.4 A network interface. ÆTHEREAL network interfaces convert the OSI network layer services of the routers to transport layer services for the user. All connection properties (cf. Section 4.1) that are end-to-end are implemented by the network interfaces. These are: reordering, transaction completion, and flow control. Unordered connections require that transaction identifiers are used, these are added at the network interfaces. For locally ordered connections, reordering buffers are required in the network interfaces. Since our routers do not reorder traffic in a connection, this is only required for narrowcast connections at the master for responses. Transaction completion also requires resource reservations in the network interfaces. Flow control serves to prevent overflow of buffers at the slave or master network interface. In a credit-based approach, this requires state (credits for the amount of space available), and additional communication (when data is consumed credits are returned to the producer). This is taken care of by the network interface.

IPS negotiate with network interfaces to obtain connections with certain properties. For this network interfaces may reserve resources, such as network interface buffers and credit counters, and slots in router tables.

5. Related work

Differentiated services have received attention in general computer networks in the context of ATM [36] and the Internet [25]. Real-time traffic schemes have been described using rate-based and deadline-based scheduling [34, 35].

The differences between single-hop on-chip communication such as busses and switches and NOCS are described in [24]. Bus protocols such as [22] often have time-division multiplexing and/or priorities added, such as MicroNetwork [37], to give throughput guarantees. To reduce the guaranteed but average high latency, statistical approaches can be used (e.g. Lotterybus [38]). The bound on completion time is then lost (cf. Section 3.1). Switches [39, 40], also single-hop interconnects, can also offer performance guarantees.

On-chip busses with bridges [41, 42, 43], but also [44], are potentially full-blown NOCS. By judiciously placing restrictions on topology, bridging, and transaction models, many problems that arise in general networks can be avoided. For example, an appropriate topology simplifies routing to a single path, and avoids transaction reordering. Limiting a master to a single outstanding transaction has the same effect. By not buffering in bridges, effectively an end-to-end circuit is set up per transaction, avoiding reordering and the need for end-to-end flow control. In both cases, the increasing latency of larger networks make these simplifying assumptions untenable in the future [33, 45].

The octagon interconnect [46] is an interesting combination of packet and circuit switching. An octagon corresponds to a single 8×8 router that uses circuit switching per transaction (or equivalently, packets of unlimited size), with a high performance for a connection in an octagon. An individual octagon can be made larger (the so-called core and edge node strategy), but this does not scale to larger networks for the reasons mentioned above. Packet switching can then be used instead, by using so-called bridge and member nodes strategy.

At this point all general NOC issues appear. This NOC seems a best-effort architecture, at least when multiple octagons are used.

The Spin [28, 29] NOC uses packet switching with worm-hole routing and input queuing in a fat tree topology. It is a scalable network for data transport, but uses a parallel network (bus) for control. It is a best-effort network, and is optimised for average performance (e.g. by the use of optimistic flow control coupled with deflection routing). Commitment is given for packet delivery, but latency bounds are only given statistically.

Dally et al [9] describes a preplaced mesh NOCs, with IPs that are synthesised in the tiles. It uses packet switching on lossless virtual channels with end-to-end flow control. No details are given on the programming model, but guaranteed throughput (and latency) services are supported.

In the context of FPGAs using bit-level circuit switching to implement inter-IP communication is expensive. A first optimisation is to use coarser circuits [47] to reduce the cost. A packet-switched interconnect with a predefined set of services allows sharing of communication resources, just like for ASICs, and enables dynamic reconfiguration [10, 11]. The NOC can be synthesised [10], but a preplaced hard-wired NOC is the next logical step.

New SOC architectures that rely on NOCs include chip multiprocessing [12, 13, 14, 7, 8, 48], to interconnect the homogeneous or heterogeneous tiles, and network processors [49].

6. Conclusions

We observe that, although future applications will be more dynamic, *users* expect embedded systems to behave predictably. A design style based on guaranteed quality of services (QoS) can solve this apparent contradiction. *Designers* of systems on chip (SOC) use networks on chip (NOC) to keep up with Moore's law. NOCs solve both deep-submicron problems (e.g. signal integrity), and narrow the design productivity gap (the efficiency with which we design SOCs) by dividing global problems into local, decoupled problems, e.g. GALS.

The combination of NOCs and QoS is natural, through network protocol stacks, and beneficial for several reasons. It enables IP re-use and platform-based design to decouple applications and architectures. QoS-based design also ensures that when global problems, such as clocking, are solved by combining local, decoupled solutions (e.g. GALS), the combination has a globally predictable behaviour, as required by the user.

There are several levels of QoS *commitment*, building on each other: correctness (e.g. uncorrupted transport), completion (e.g. packet delivery), and completion bounds (e.g. maximum latency). A service without commitment (a *best-effort* service, such as unbounded latency), can have a better average resource utilisation than a *guaranteed* service, but at the cost of unpredictable or unbounded worst-case behaviour. The *combination of best-effort and guaranteed services is advantageous*: critical parts of the system are predictable, while resources are used more efficiently.

The ÆTHEREAL NOC combines best-effort and guaranteed services at different levels. Its programming model and architecture are also described. The ÆTHEREAL NOC is at the basis of a QoS-based design style, as advocated in this chapter.

References

[1] Semiconductor Industry Association. *The International Technology Roadmap for Semiconductors*. 2001.

[2] Paul Wielage and Kees Goossens. Networks on silicon: Blessing or nightmare? In *Euromicro Symposium On Digital System Design*, Dortmund, Germany, September 2002. Keynote speech.

[3] T. N. Theis. The future of interconnection technology. *IBM journal of research development*, 44(3):379–390, May 2000.

[4] Marcel J. Pelgrom, Hans P. Tuinhout, and Maarten Vertregt. Transistor matching in analog CMOS applications. In *IEDM*, pages 915–918, 1998.

[5] K. Keutzer, S. Malik, A. Richard Newton, Jan M. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 19(12):1523–1543, 2000.

[6] Jens Muttersbach, Thomas Villiger, and Wolfgang Fichtner. Practical design of globally-asynchronous locally-synchronous systems. In *6th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, April 2000.

[7] Paul Stravers and Jan Hoogerbrugge. Homogeneous multiprocessing and the future of silicon design paradigms. In *VLSI-TSA*, 2001.

[8] Ken Mai, Tim Paaske, Nuwan Jayasena, Ron Ho, William J. Dally, and Mark Horowitz. Smart memories: A modular reconfigurable architecture. In *ISCA*, June 2000.

[9] William J. Dally and Brian Towles. Route packets, not wires: On-chip interconnection networks. In *Design Automation Conference*, pages 684–689, June 2001.

[10] Théodore Marescaux, Andrei Bartic, Dideriek Verkest, Serge Vernalde, and Rudy Lauwereins. Interconnection networks enable fine-grain dynamic multitasking on FPGAs. *FPL*, 2002. LNCS 2438.

[11] Edson L. Horta, John W. Lockwood, David E. Taylor, and David Parlour. Dynamic hardware plugins in an FPGA with partial run-time configuration. In *Design Automation Conference*, June 2002.

[12] Lance Hammond, Basam A. Hayfeh, and Kunle Olokotun. A single-chip multiprocessor. *IEEE Computer*, pages 79–85, September 1997.

[13] Jaehyuk Huh, Stephen W. Keckler, and Doug Burger. Exploring the design space of future CMPs. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2001.

[14] Eylon Caspi, André DeHon, and John Wawrzynek. A streaming multithreaded model. In *Third Workshop on Media and Stream Processors (MSP-3)*, December 2001.

[15] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vincentelli. Addressing the system-on-a-chip interconnect woes through communication-based design. In *Design Automation Conference*, pages 667–672, June 2001.

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

[17] K. Goossens, J. van Meerbergen, A. Peeters, and P. Wielage. Networks on silicon: Combining best-effort and guaranteed services. In *Proceedings of Design Automation and Test Conference in Europe*, pages 423–425, March 2002.

[18] A. Ferrari and A. Sangiovanni-Vincentelli. System design: traditional concepts and new paradigms. In *International Conference on Computer Design*, pages 2–12, 1999.

[19] J. D. Day and H. Zimmerman. The OSI reference model. In *Proceedings of the IEEE*, volume 71, pages 1334–1340, 1983.

[20] K. G. W. Goossens and O. P. Gangwal. The cost of communication protocols and coordination languages in embedded systems. In Farhad Arbab and Carolyn Talcott, editors, *Coordination languages and models*, number 2315 in Lecture notes in computer science, pages 174–190. Springer Verlag, April 2002.

[21] Steve Deering. Watching the waist of the protocol hourglass. In *6th IEEE International Conference on Network Protocols*, October 1998. Keynote speech.

[22] VSI Alliance. Virtual component interface standard, 2000.

[23] OCP International Partnership. Open core protocol specification, 2001.

[24] Andrei Rădulescu and Kees Goossens. Communication services for networks on silicon. In Shuvra Bhattacharyya, Ed Deprettere, and Juer-

gen Teich, editors, *Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation*. Marcel Dekker, December 2002.

[25] Vijay P. Kumar, T. V. Lashman, and Dimitrios Stiliadis. Beyond best effort: Router architectures for the differentiated services of tomorrow's internet. *IEEE Communications Magazine*, pages 152–164, May 1998.

[26] Girish Varatkar. Traffic analysis for on-chip networks design of multimedia applications. In *Design Automation Conference*, June 2002.

[27] Sungjoo Yoo, Kyoungseok Rha, Youngchul Cho, Jinyong Kung, and Kiyoung Choi. Performance estimation of multiple-cache IP-based systems: Case study of an interdependency problem and application of an extended shared memory model. In *International Workshop on Hardware/Software Codesign*, May 2002.

[28] Pierre Guerrier and Alain Greiner. A generic architecture for on-chip packet-switched interconnections. In *Proceedings of Design Automation and Test Conference in Europe*, pages 250–256, 2000.

[29] Pierre Guerrier. *Un Réseau D'Interconnexion pour Systémes Intégrés*. PhD thesis, Université Paris VI, March 2000.

[30] Davide Bertozzi, Luca Benini, and Giovanni De Micheli. Low power error resilient encoding for on-chip data buses. In *Proceedings of Design Automation and Test Conference in Europe*, March 2002.

[31] E. Rijpkema, K. Goossens, A. Rădulescu, 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 *Proceedings of Design Automation and Test Conference in Europe*, March 2003.

[32] Edwin Rijpkema, Kees Goossens, and Paul Wielage. A router architecture for networks on silicon. In *Proceedings of Progress 2001, 2nd Workshop on Embedded Systems*, Veldhoven, the Netherlands, October 2001.

[33] André DeHon. Robust, high-speed network design for large-scale multiprocessing. A.I. Technical report 1445, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, September 1993.

[34] Hui Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, 83(10):1374–96, October 1995.

[35] Jennifer Rexford, John Hall, and Kang G. Shin. A router architecture for real-time communication in multicomputer networks. *IEEE Transactions on Computers*, 47(10):1088–1101, October 1998.

[36] ATM Forum. *ATM User-Network Interface Specification*. Prentice Hall, July 1994. Version 3.1.

[37] Drew Wingard. MicroNetworks-based integration for SOCs. In *Design Automation Conference*, 2001.

[38] Kanishka Lahari, Anand Raghunathan, and Ganesh Laskhminarayana. Lotterybus: A new high-performance communication architecture for system-on-chip designs. In *Design Automation Conference*, June 2001.

[39] Jeroen A.J. Leijten, Jef L. van Meerbergen, Adwin H. Timmer, and Jochen A.G. Jess. Stream communication between real-time tasks in a high-performance multiprocessor. In *Proceedings of Design Automation and Test Conference in Europe*, pages 125–131, 1998.

[40] Paul J.M. Havinga. *Mobile Multimedia Systems*. PhD thesis, University of Twente, The Netherlands, February 2000.

[41] Kyeong Keol Ryu, Eung Shin, and Vincent J Mooney. A comparison of five different multiprocessor SoC bus architectures. In *Euromicro*, 2001.

[42] Tycho van Meeuwen, Arnout Vandecappelle, Allert van Zelst, Francky Catthoor, and Diederik Verkest. System-level interconnect architectures exploration for custum memory organizations. In *International Symposium on System Synthesis*, pages 13–18, October 2001.

[43] Milenko Drinic, Darko Kirovski, Seapahn Meguerdichian, and Miodrag Potknojak. Latency-guided on-chip bus network design. In *Proc. of IEEE/ACM International Conference on Computer Aided Design*, pages 420–423, November 2000.

[44] John Bainbridge and Steve Furber. CHAIN: A delay-insensitive chip area interconnect. *IEEE Micro*, 22(5), 2002.

[45] Frederic Chong, Henry Minsky, André deHon, Matthew Becker, Samuel Peretz, Eran Egozy, and Frank F. Knight, Jr. Metro: A router architecture for high-performance, short-haul routing networks. In *International Symposium on Computer Architecture*, April 1994.

[46] Faraydon Karim, Anh Nguyen, Sujit Dey, and Ramesh Rao. On-chip communication architecture for OC-768 network processors. In *Design Automation Conference*, June 2001.

[47] André DeHon. Rent's rule based switching requirements. In *SLIP*, April 2001. Extended abstract.

[48] Shashi Kumar, Axel Jantsch, Juha-Pekka Soininen, Martti Forsell, Mikael Millberg, Johny Öberg, Tiensyrjä, and Ahmed Hemani. A network on chip architecture and design methodology. In *ISVLSI*, 2002.

[49] David Whelihan and Herman Schmit. Memory optimization in single chip network switch fabrics. In *Design Automation Conference*, June 2002.