Networks on Silicon: Combining Best-Effort and Guaranteed Services

K. Goossens, J. van Meerbergen, A. Peeters, and P. Wielage
Philips Research Laboratories, Eindhoven, The Netherlands
{kees.goossens,jef.van.meerbergen,ad.peeters,paul.wielage}@philips.com

Abstract

*We advocate a network on silicon (*NOS*) as a hardware architecture to implement communication between* IP *cores in future technologies, and as a software model in the form of a protocol stack to structure the programming of* NOSs*. We claim guaranteed services are essential. In the* ÆTHEREAL NOS *they pervade the* NOS *as a requirement for hardware design, and as foundation for software programming.*

1. Introduction

It will soon be possible to manufacture systems on silicon (SOSs) containing 10M gates and megabytes of embedded software. The complexity of these systems will be overwhelming, and two fundamental questions arise. First, from a hardware view point, how can these SOSs be designed? What are the required hardware architecture concepts? Second, from a software (application) perspective, how can the SOS hardware be programmed? What software concepts are needed? Our tenet is that only a consistent approach to both questions will enable us to harness the opportunities offered by semiconductor technology.

We believe in the intellectual-property block (IP) re-use model. This means that a SOS contains hundreds of IPs that are bought or premade. Assuming their correctness, their communication is then the problem. We propose a *network on silicon* (NOS) to implement the communication between these IPs. A NOS is a hardware architecture together with a programming model.

Guaranteed services are the focus of this paper: they serve as the basis for the programming model, and as a requirement for the NOS hardware architecture. A *protocol stack* is a layered approach to offering different services to different clients, based on a common network. It structures communication complexity from the physical implementation up to the application in a number of layers. The OSI reference model [4], used in this paper, comprises seven layers. In Section 2 we motivate why a packet-switched router network is a suitable hardware architecture, using the lower three layers as a backdrop. The top four layers are independent of the hardware architecture and are discussed in Section 3. In that context we show why guaranteed services offer many advantages over *best-effort* services when programming SOSs.

2. Packet-switched Router Networks

In this section we consider how a SOS containing hundreds of IPs can be designed, especially regarding layout and timing closure. Consider that the communication patterns of the IPs are dynamic at two levels. First, multiple configurations exist, each corresponding to a required user functionality. Second, communication data rates are variable within a configuration (e.g. variable-bit-rate MPEG streams). Structuring the wires used for communication between IPs is the main problem. If dedicated point-to-point wires are used, IPs are connected to all possible communication partners. This leads to many global wires, with three problems. First, they must be dimensioned for worst-case conditions in terms of a) electrical parameters such as cross talk and wire spacing, and b) volume of traffic. Second, it causes wire congestion around each IP. Thin closely-spaced high-power wires solve congestion, but this conflicts with the requirements for global wires, which are fat and widely spaced. Third, layout and timing dependencies between all IPs make timing closure global, making it non-scalable. Worse, it is layout specific: it cannot be re-used over design iterations or different designs [2].

A router network based on packet switching helps on these accounts. Fundamentally, wires are shared, or made programmable, by introducing switches (routers). This is possible because dedicated wires are normally underutilised (as little as 10% [3]). The first benefit is that there are fewer programmable wires, which reduces wire congestion around IP blocks. Second, traffic of multiple IPs is combined over a single wire, which can be dimensioned for the average traffic over all blocks using that wire instead of worst-case traffic per IP. Third, inter-IP wires are now structured: either they are short to get on the network, or they are router to router. The network topology and router layout can both be fixed (e.g. a torus and mesh [3] or a fat tree and Maltese cross [5]), or only the network topology. This allows dedicated optimisations that are re-usable over SOSs. Finally, and most importantly, decoupling of computation (IPs) and communication (NOS) enables compositional layout and timing closure. IPs communicate only using the network so that their layout and timing closure are independent of other IPs.

Programmable wires also have drawbacks. First, different IPs will contend for the use of a wire, requiring arbitration or scheduling. The properties of a router network depend much on the kind of scheduling it uses. A schedule can be fixed for a configuration (static) or dynamic (within a configuration). Static scheduling must be worst case, while dynamic scheduling leads to dimensioning for average traffic volume. Scheduling can also be independent for each router

(local) or consider all routers (global). Global scheduling can avoid *contention* (several packets want to use the same resource at the same time) and *congestion* (packets waiting for a resource) but is not scalable. Global scheduling is usually static (circuit switching) and local scheduling dynamic (packet switching). Performance guarantees (i.e. absolute certainty instead of probabilistic behaviour) are more easily given for global static scheduling than for local dynamic scheduling. The way packets are buffered and scheduled in routers, and the effects on performance guarantees has been the subject of intense research. Fundamentally, sharing and guarantees are conflicting, and efficiently combining guaranteed traffic with best-effort traffic is hard [7].

Second, programmable wires may be slower than using dedicated IP-to-IP wires, which may contain some buffers. In a NOS a packet must traverse a number of routers. When scheduling statically, a packet need not incur more than a pipeline delay per hop. But for dynamic scheduling the header of each packet must be inspected before it can be sent to the appropriate output port. Contention and congestion may further delay a packet. Finally, a packet header uses some of the communication bandwidth.

Although a NOS based on packet-switching has its costs (packetisation, routers, buffering, harder to give guarantees) it is essential for an IP re-use strategy because it enables compositional and scalable integration of the IPs. Moreover, a NOS infrastructure (such as routers and protocol stack) can be re-used and its design cost amortised over multiple designs.

The issues raised in this section can be mapped to the OSI protocol stack [4]. Below we discuss the three lower layers that are specific to the hardware architecture of the network.

The *physical layer* defines how a single bit is sent over a physical medium (such as a wire, bus, optical link, radio waves). For a NOS it deals with signal voltages and slopes, wire sizing, etc. of inter-router wires. The preceding discussions motivate why a router-based NOS eases the implementation of the physical on-chip wires.

The *data-link layer* ensures reliable (lossless, fault-free) communication over a single data link. It considers cross-talk and fault-tolerance (both transient soft errors and hard errors). For shared and especially broad-cast media (such as busses), the medium access sublayer deals with contention for the link. Examples are bus arbitration, and router-to-router flow control.

The *network layer* has two main functions for a given network. First, delivering packets, more specifically, routing, congestion control, and scheduling over multiple links. Second, network management, including accounting, monitoring, management or steering, and internetworking.

3. Guaranteed Services are essential

In this section we discuss the remainder of the OSI protocol stack, and then show why the services offered by this stack must be predictable.

In a re-use model IPs that are bought or re-used cannot be adapted to each SOS they are used in. They will interact in many different ways (event-driven, data streaming, message passing, shared memory, etc.) [8]. Thus the interconnecting network has to be flexible, in terms of the services that it offers. This is the task of the top four OSI layers. They are independent of the network hardware architecture, *end to end* (the network is transparent), and *peer to peer* (between entities at the same level of abstraction, such as task to task). We now list these layers, starting with the lowest.

The aim of the *transport layer* is to provide reliable *end-to-end services* over *connections*, through packetisation, end-to-end connection management (perhaps over multiple networks), (de)multiplexing multiple transport connections over network connections, and so on. Examples of services per connection are: uncorrupted transmission, lossless transmission, in-order delivery, throughput, jitter (delay variation), and latency. Flow control avoids data loss by aligning IP production and consumption rates.

In the *session layer* multiple connections are combined into services such as multicast, half duplex and full duplex connections. Synchronisation of multiple connections (of audio and video streams, for example), and token management for mutual exclusion and atomic (chains of) transactions (e.g. test and set, swap) may also be offered.

The *presentation layer* converts the application view on data to a view more appropriate for transport and vice versa. Examples are encryption and endian-ness conversions.

In the *application layer* communication is natural to applications, running on IPs. This could be bit streams obeying the MP3 standard, video frames, or Java byte code.

We now return to the question of programming. A SOS consists of a set of IPs and a NOS that must be (dynamically) *configured*. For a given user function, computation is mapped to functional IPs (e.g. tasks to processors) and communication to the NOS. The number of configurations can be large and they may be generated at run-time by an application manager. To avoid unnecessary design iterations, we need to be sure of correctness of the combination of static hardware (the IPs and their interconnect) and dynamic software (the configurations) as early as possible in the design flow. Programming must be predictable; this can be ensured by basing it on a NOS that offers guaranteed services.

A NOS with guaranteed services simplifies programming individual IPs in three ways. First, some IPs have inherent performance requirements, such as a minimum throughput (for real-time streaming data), or bounded latency (for interrupts). A NOS with guaranteed services eases their integration. Second, an IP communicates using the NOS. Requests for services make its assumptions and dependencies on the NOS explicit, structuring the design and programming of the IP. Third, an IP can also be simpler when using guaranteed

services because it has fewer possible interactions (a stricter contract) with its environment. A service request may be granted or denied by the network. Communication failures are therefore restricted to the configuration phase of the IP, instead of every communication action, so simplifying the IP's programming model.

Guaranteed services offer the following advantages for the composition of IPs. 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 SOSs. When IPs, assumed to be correct, are combined in a system it must be clear that the system will function as a whole. This holds both at the time of design and at run time. Current systems often fail this requirement. Consider a bus; adding an IP influences the electrical parameters (like load) of the bus, as well as the existing traffic patterns. The latter can seriously affect cache behaviour and hence embedded processor performance. Second, a NOS that offers guaranteed services must accurately model its resources to offer those services. This makes costs of the NOS explicit early in the design of a SOS. At run-time an application or quality-of-service manager reconfigures the SOS. This can only be effective when the NOS is observable and controllable, but above all, predictable in terms of performance and cost. Third, a NOS based on guaranteed services concentrates communication failures to (re)configuration points. When the NOS declines a service request from an IP, only that IP is affected. Fallback strategies are much easier for local configuration-time failure than for global failure at any point caused by any of the communicating parties.

4. The Æthereal Network-on-Silicon

The ÆTHEREAL NOS intends to support both guaranteed and best-effort services. We can only list some salient features here, to indicate our current direction.

We use a combination of circuit and packet switching, in the spirit of ATM [1]. A channel implements a simplex point-to-point communication between two network interfaces (possibly with multiple paths). A channel can have a combined throughput and latency guarantee (GT), or be best effort (BE), with an unspecified finite latency upper bound. Delivery order is not guaranteed over different paths or channels. A connection combines multiple channels and can include flow control and reordering for lossless in-order (multicast) transmission. The programming model is concurrent and distributed: IPs can set up and remove connections in a pipelined fashion, and at the same time. Configurations that have been computed off-line can be efficiently loaded and extended or changed at run time. GT channels are created and removed with BE packets that reserve a path through the network, subject to resource availability.

BE packets use worm-hole source routing, and GT streams use time-division-multiplexed circuit switching. Routers use virtual output queuing for BE traffic. GT traffic is tightly scheduled to minimise buffering. Routers do not drop or reorder packets, the former due to fine-grain link-level flow control. A variant of the iSLIP switch-scheduling algorithm [6] is used for BE traffic. BE traffic use all spare capacity in the network, i.e. bandwidth that is not reserved or unused by GT traffic. Network interfaces implement multi-path and multi-channel ordering, and channel flow control.

5. Conclusions

In the IP re-use model communication between the many IP cores is key. We have shown that a network on silicon (NOS) eases the design of systems on silicon (SOSs) at the levels of hardware design and software programming.

A NOS hardware architecture based on a packet-switched router network allows average-case wire dimensioning, and reduces wire congestion around IPs. It breaks the fatal global timing closure loop by separating inter-IP from intra-IP communication, and can so reduce global design iterations. The OSI protocol stack structures the communication complexity in the router network (the physical to network layers) and the programming model (the transport to application layers).

While a protocol stack offers diverse services on a single hardware architecture, we show that predictability, in the form of guaranteed services, greatly benefits programming a SOS at two levels. Individual IPs with real-time requirements are more easily integrated, and the NOS's actions are restricted by the contract, simplifying the IP's IO. At the NOS level, programming of IPs is compositional because their services are independent of each other. Quality-of-service managers can effectively observe and steer a NOS because it behaves predictably in terms of performance and cost.

We believe that a NOS must be a synthesis of hardware architecture and software programming model with guaranteed services at its core. In the ÆTHEREAL NOS we aim to do so efficiently.

References

[1] ATM Forum. *ATM User-Network Interface Specification*. 1994

[2] R. E. Bryant, et al. Limitations and challenges of computer-aided design technology for CMOS VLSI. *Proc. of the IEEE*, 89(3):341–365, March 2001.

[3] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *DAC*, pages 684–689, June 2001.

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

[5] P. Guerrier. *Un Réseau D'Interconnexion pour Systémes Intégrés*. PhD thesis, Université Paris VI, Mar. 2000.

[6] N. McKeown. iSLIP: A scheduling algorithm for input-queued switches. *IEEE Transactions on Networking*, 7(2), Apr. 1999.

[7] J. Rexford and K. G. Shin. Support for multiple classes of traffic in multicomputer routers. In *LNCS* 853, 1994.

[8] M. Sgroi, et al. Addressing the system-on-a-chip interconnect woes through communication-based design. In *DAC'2001*