Chapter 1

WILL NETWORKS ON CHIP CLOSE THE PRODUCTIVITY GAP?

Axel Jantsch and Hannu Tenhunen
Royal Institute of Technology, Stockholm
axel@imit.kth.se, hannu@imit.kth.se

Abstract We introduce two properties of the design process called the *arbitrary composability* and the *linear effort properties*. We argue that a design paradigm, which has these two properties is scalable and has the potential to keep up with the pace of technology advances. Then we discuss some of the trends that will enforce significant changes on current design methodologies and techniques. Finally, we argue that the emerging Network-on-Chip (NoC) paradigm promises to address these trends and challenges and has all prerequisites to provide the arbitrary composability and the linear effort properties. Consequently we conclude that NoC is a likely basis for future System-on-Chip platforms and methodologies.

Keywords: Networks on chip, Productivity gap, System on chip design methodology

1. Introduction

To boost design productivity it is crucial that the effort to add new parts to a given design does not depend on the size of the existing design but only on the size of the new parts. In other words, the design effort must be a linear function of the size of the new parts. If this is the case, large parts and blocks of previous designs can be reused and the design effort can be invested into the new parts. This is also a necessary prerequisite to provide a solid methodology, architecture, and thus a platform, that are sustainable over several technology generations.

The central thesis of this chapter is that a Network-on-Chip (NoC) has the potential to provide such a sustainable platform and, if successful, will incur such a significant change on the system-on-chip architecture

and design process that it can be called a paradigm change. On the other hand, if it fails to do so, NoC will be just one of several architectures and platforms available to embedded system designers.

> **Arbitrary composability property:** *Given a set of components and a set of combinator operators which allow to connect and integrate the components into larger component assemblages. Components and combinators together are arbitrarily composable if a given component assemblage* **A** *can be extended with any component by using any of the combinators without changing the relevant behavior of* **A**.

Please note, that this and the following property are meant as engineering heuristics, not as mathametical properties. As such they are ideals and can be achieved at higher or lower degrees.

Note further, that this property is defined with respect to what is considered to be a *relevant behavior*. Thus depending on the given objectives and definition of behavior, the same components and combinators may or may not have the arbitrary composability property.

For instance, the standard logic gates `NAND`, `NOR`, `INV`, etc. have this property with respect to their logic level I/O behavior because adding new gates to a netlist of gates will not change the behavior of the original netlist, unless old connections are broken. A given network of gates can be used in any context and will exhibit identical behavior whatever the surrounding netlist may be. New gate netlists can be added to existing ones, using the outputs and results produced by any other part of the circuit without changing the older parts. This is the foundation of our ability to build designs with millions of gates and to reuse large blocks in arbitrary environments.

It should be noted that this nice property of gates is in part due to the implementation process which allows the scaling of transistor sizes, insertion of buffers, and sensible placement and routing by automatic tools.

It is enlightening to see the effects when the arbitrary composition property is violated. Two of the most severe problems in today's designs stem from violations of this property. Timing closure, i.e. the problem to get the timing of the circuit implementation right, is difficult because small changes or addition to the gate netlist may change the timing of the entire system by adding to the critical path or due to an unexpected effect of placement and routing on the timing of seemingly unrelated circuit parts. The system verification problem is so hard because at the system level behaviors are not easily composable and tiny changes in one part may have unexpected effects on seemingly unrelated other parts of

the system. In both cases the design effort grows more than linear with the system size.

If this property is guaranteed, the effort of adding new components to a working system only depends on the new components but not on the size of the reused system (figure 1.1). Thus, a corollary of the arbitrary

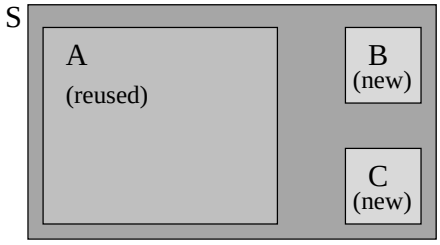

Figure 1.1. With the arbitrary composability property the design effort to add new components to an existing system depends on the integration effort and on the new components but not on the size of A because the design effort to build A has already been spent and is reused as well: $\texttt{Deffort}(S) = \texttt{Deffort}(B) + \texttt{Deffort}(C) + \texttt{Ieffort}(3)$

composability property is the following linear effort property.

> **Linear Effort Property:** *Given is a set of components and a set of combinator operators which allow to connect and integrate the components into larger component assemblages. A design process which builds a system from the components and combinators has the linear effort property if a given set of n assemblages $A_1, \ldots, A_n$ can be integrated into a system S by means of the combinators with an effort dependent on n but not on the size of the assemblages:* $\texttt{Ieffort}(n)$. *Thus the total design effort for S is*
>
> $$\texttt{Deffort}(S) = \texttt{Deffort}(A_1) + \cdots + \texttt{Deffort}(A_n) + \texttt{Ieffort}(n)$$

Note, that this property implies that the interface complexity of an assemblage does not depend on the size of the assemblage. Obviously, this is not true in practice but it is equally obvious that this is a necessary precondition to build arbitrary large systems. Thus, we must approach this ideal as close as possible to be able to build larger and larger systems. The fact, that we have not been sufficiently close to this ideal is the fundamental reason for the design productivity gap.

We believe that NoC based platforms have a good potential to provide both the arbitrary composition and the linear effort properties to a high degree but they do not automatically guarantee them. We will keep these properties in mind throughout this chapter, but first we review

some of the underlying trends and challenges that lead to NoC and similar architectures.

2. Trends and Challenges

IC manufacturing technology will provide us with a a few billion transistors on a single chip within a few years [1]. Assuming that these predictions hold and that the market will continue to absorb ever higher volumes of ICs, the key questions are: how will the future chips be organized and how will future systems, which include these chips, be designed? There are a few trends which, if continued, will bring about a significant change for architecture and design of integrated circuits.

Communication versus computation. Technology scaling works better for transistors than for interconnecting wires. This leads gradually to a domination of performance figures, power consumption and area by wires and make transistors of secondary importance. At the system level it has a profound effect by changing the focus from number crunching and computation to data transport and communication [2, 3, 4]. Communication becomes often the bottleneck because it seems much harder to design and get right.

Deep submicron effects. Cross-coupling, noise and transient errors are only some of the unpleasant side-effects of technology scaling [5, 6]. It requires significant skills, experience, knowledge and time to keep them under control while exploiting the limits of a technology. A digital or system designer with an expected design productivity of millions of transistors per day is not able to deal with these effects properly. Therefore, designers reuse blocks which are carefully designed by experts with the proper skills. However, it is of critical importance that the deep submicron effects don't pop up again when predesigned blocks are combined in arbitrary ways. Consequently, at the physical design level the property of arbitrary composability means that the electrical and physical properties of blocks are not affected when combining them.

Global synchrony. Physical effects of deep sub-micron technology make it increasingly difficult to maintain global synchrony among all parts of the chip [7]. The clock signal will soon need several clock cycles to traverse the chip, clock skew becomes unmanageable, and the clock distribution tree is already today a major source of power consumption and cost. The trends of scaling to smaller geometric dimension and higher clock frequency make these problems more significant every year.

Thus, it is unlikely that large chips will be synchronous designs with only one clock domain.

Design productivity gap. Synthesis and compiler technology development do not keep pace with IC manufacturing technology development [1]. As a consequence we need either exponentially growing design teams or design time to design and implement systems which fit onto a single IC. Since both alternatives are unrealistic we have in the past escaped from the problem by using ever more complex components as primitive design units. These primitive design units have evolved from individual transistors to logic gates to entire ALUs, multipliers, finite state machines and processor cores. In fact *reuse* of ever more complex design elements has been the main device to increase productivity and will likely remain so in the years to come. It has also kept us close enough to the ideal of the linear effort property to manage the increasing number of transistors.

Heterogeneity of functions. Obviously, systems that can be implemented on a single chip become increasingly more complex. As a result different functions and features with vastly different characteristics and history reside on the same chip. Signal processing algorithms, that recover and generate radio signals, will coexist with global control, maintenance and accounting functions as well as with natural language comprehension and generation functions. These functions are developed in different contexts, by different teams, with different design languages and tools. However, they need to be integrated into a single chip.

In order to lead a concrete discussion we describe next a typical NoC architecture and in section 4 we investigate how a NoC approach could address the listed problems. To paint a fair picture section 5 illuminates the price to pay when adopting a NoC based approach. Finally, in section 6 we speculate how the design process will change.

3. Network on Chip

The Network-on-Chip (NcC) architecture, as outlined in figure 1.2, provides the communication infrastructure for the resources. In this way it is possible to develop the hardware of resources independently as stand-alone blocks and create the NoC by connecting the blocks as elements in the network. Moreover, the scalable and configurable network is a flexible platform that can be adapted to the needs of different workloads, while maintaining the generality of application development methods and practices.

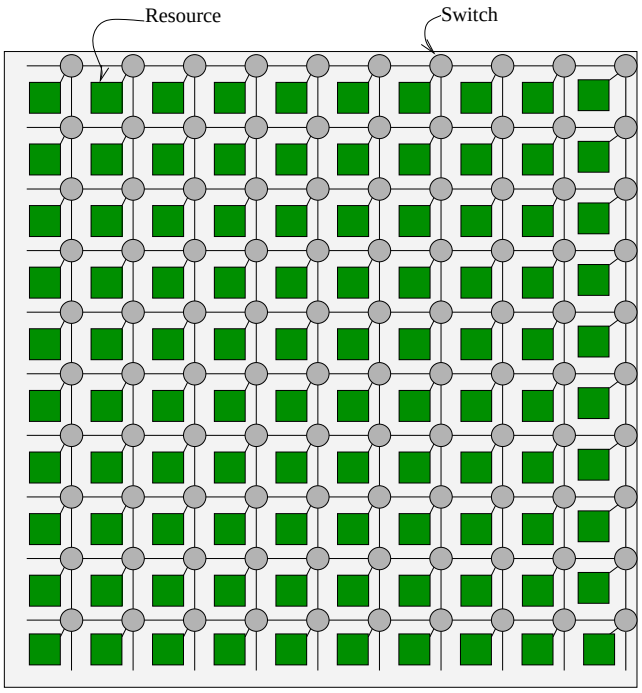

Figure 1.2. Each node in the mesh contains a switch and a resource.

A two dimensional mesh interconnection topology is simplest from a layout perspective and the local interconnections between resources and switches are independent of the size of the network. Moreover, routing in a two-dimensional mesh is easy resulting in potentially small switches, high bandwidth, short clock cycle, and overall scalability. A NoC consists of resources and switches that are directly connected such, that resources are able to communicate with each other by sending messages. A resource is a computation or storage unit. Switches route and buffer messages between resources. Each switch is connected to four other neighboring switches through input and output channels. A channel consists of two one-directional point-to-point buses between two switches or a resource and a switch. Switches may have internal queues to handle congestion. We expect that the area of a resource is either the maximal synchronous region in a given technology or a cluster of computing elements and memory connected via a bus. We expect the size of a resource to shrink with every new technology generation. Consequently the number of resources will grow, the switch-to-switch and the switch-to-resource bandwidth will grow, but the network wide communication protocols will be unaffected.

Note, that this is one, fairly simple NoC variant described here only to substantiate the following discussion. Many other more sophisticated architectures have been proposed [8, 9, 10, 11, 12, 13, 14].

4. How does NoC address the problems

Although there are numerous factors and facets to investigate, we focus on two mechanisms that work in favor of a NoC based approach: *Reuse* and *Predictability*. We believe these are by far the most important factors overshadowing other secondary effects.

4.1 Reuse

As mentioned above, reuse has always been the primary means to bridge the technology gap [15, 16, 17]. More and more complex components, from transistors to gates to functional blocks, such as ALUs and multipliers, to microprocessors and DSP cores, have become the primitive building blocks. In this way, the designers could move up the abstraction levels and describe the system's functionality in more and more abstract terms relying on more and more powerful "primitive" components. Curiously, synthesis technology has mostly been used to bridge one, relatively shallow, level of abstraction. This can be observed in both the hardware and the software domain. Technology mapping, logic and RTL synthesis are in principle straight forward steps with a well characterized optimization space. Synthesis techniques that attempted to bridge more abstraction levels and to make more profound design decisions have typically failed. Examples are high level and architectural synthesis. In software the mapping from C to microprocessor instruction sets is more or less direct and the corresponding compiler technology is tremendously successful. In contrast, compilation from functional, logic and other higher level languages lacks efficiency and has never become mainstream. Thus, we believe that reuse by providing more complex components will continue to be the main mechanism to exploit the potential of technology. Synthesis and compilation techniques will provide the surface mapping from more convenient descriptions onto the primitive components.

However, as a difference to the past, communication "components", or better, communication structures and services have also to become primitive design elements. And this is precisely what a NoC based approach is all about: The reuse of communication services.

Let us briefly review what can be reused in a NoC based approach.

Components and resources. Arbitrary computation elements can be connected to the communication network. In fact we expect that typical NoC based systems may contain processor cores, DSP cores, memory banks, specialized I/O blocks such as Ethernet or Bluetooth protocol stack implementations, graphics processors, FPGA blocks, etc. The size of a resource can range from a bare processor core to a local cluster of several processors and memory connected via a local bus. Resources have to comply to the interfaces of the communication network in order to connect to it and use its services.

The reuse of processor cores has been developed during the last ten years by defining bus interfaces. By defining network interfaces, NoC takes this concept further because it allows to integrate an arbitrary number of resources into a network. In a bus-based system, adding a new resource has a profound impact on the performance of the rest of the system because the same communication resource is now shared among more resources. In a NoC adding new resources also means to add new communication capacity by adding new switches and interconnects. This scalability property is a necessary precondition for the arbitrary composability property but it is not sufficient to guarantee it. The communication network must further be able to guarantee allocated bandwidth and to enforce a decent behaviour of the resources to avoid, for instance the monopolization of the entire communication bandwidth by a single resource.

However, it is important to acknowledge, that a NoC based approach has the potential to provide the arbitrary composability property with tremendous benefits for the design productivity.

Communication infrastructure. The main immediate benefit from a NoC based approach is clearly due to the possibility to reuse the communication network. The switches, the interconnects and the lower level communication protocols can be design, optimized, verified and implemented once and reused in a large number of products. If requirements on performance, reliability and cost differ too much in different application domains, domain specific communication networks can be reused at least for all products in the same domain. For instance, it is likely that mobile, hand-held devices have so different demands on power consumption and performance than infra-structure equipment that different NoC platforms for these two domains are well justified.

Apart from the hardwired communication infra-structure, higher level network services can as well be reused. There is in fact a long list of services that would benefit many applications but can impossibly be developed from scratch for each new product. Examples are

- the detection, monitoring and management of faults in the network;
- the allocation and management of network resources and possibly task migration for load balancing and power optimization;
- the management of global and shared memory;
- the provision of sophisticated communication services such as channels with guaranteed bandwidth and quality of service, multi-cast and broadcast communication, etc.

Most of these tasks are typically provided by the operating system in today's uniprocessor applications. Similarly, a NoC operating system will be very generic and can be reused for many products. Due to the increased complexity of future systems as compared to today's systems, the operating system will be much more sophisticated and complex, thus making the case for reuse even stronger.

Application parts and feature reuse. Reuse will not stop with components and generic services. New products can be composed of existing, complete features. Peeking into the future we can envision a traditional mobile phone which is enhanced by speech analysis subsystem, a speech synthesizer and a language analysis and processing sub-system to provide a spoken language interface to the phone. The same modules or features are apparently useful in a wide range of products and should therefore be reused as much as possible. Obviously, the main challenge will be to define and standardize the high level interfaces between these features to allow for an efficient communication and sharing of information. However, we can observe that a NoC provides an excellent ground for this kind of feature reuse. For optimized implementation a feature may come fully implemented either in software or partially as a dedicated hardware block. Either way, the feature can be plugged into one or several resource slots and the NoC provides at least the low level communication and network services for free for the interaction between the feature and the rest of the system.

Design, simulation and prototype environment. A significant part of system development costs are typically spent in setting up simulation environments and building prototypes. Since many products are based on the same NoC platform much of this investment can be shared by many products. Furthermore, even if different domain specific NoC platforms vary in their performance and power characteristics, they are

sufficiently similar to allow the reuse of much of the design and verification environment across very different application domains. In contrast, application specific platforms which are not derived from the same principle concepts but are developed in an ad-hoc way to suit a particular application domain, will not provide as much potential for cross-domain reuse.

Verification effort. The system verification effort is frequently as high as the design effort itself. Hence, by reusing predesigned and preverified parts and services, verification time can be as drastically reduced as design time. But reuse is even more important for the verification than for the design activity because the uncertainty about the required verification time is much higher and more difficult to plan. Moreover, the uncertainty about the resulting product quality is high and the potential cost of undetected errors in the final product can be enormous. Since risk and uncertainties around verification and verification effects are much higher, verification benefits more from reuse than the design activities. Reused components are typically much better verified, because they have already been used in other products. Moreover, system verification can be done much more effectively when correctness and reliability of the components can be assumed, because errors are identified faster.

In summary, the usage of a NoC based platform boosts the potential for reuse in many ways.

4.2 Predictability

The second main aspect of our focus is predictability and it is in fact closely related to reuse.

Communication performance. Due to its regular geometry and communication network, communication performance becomes potentially much more predictable. "Potentially" and not "necessarily" because the regular communication hardware will significantly help to analyze and assess performance but measures at higher protocol levels and network services have to realize this potential. Since the communication hardware is shared by many resources, the activity of one resource can delay the communication of other resources. One mis-behaving resource can monopolize part of the communication hardware and indefinitely block other resources from accessing it. Therefore, the network has not only to provide the raw bandwidth but also a policy for bandwidth allocation. There are many ways to do this with different advantages and disadvantages. One possibility is to allocate channels with a maxi-

mum bandwidth and fixed latency to two communicating resources. By properly allocating and managing these channels many communication activities can coexist peacefully and all of them know exactly the available bandwidth and what latency each data packet will exhibit. This makes systemwide performance analysis feasible and accurate.

As indicated above, a reliable communication resource allocation policy resulting in predictable communication performance for all tasks and applications in the NoC is a central part for establishing the property of arbitrary composability. If a task or a feature can request, obtain and use communication bandwidth independent of all other tasks and features in the NoC, then the cost of reusing, mixing and matching existing tasks and features is relatively small, since they have not to be modified internally.

Electrical properties. Due to the regularity of the layout the electrical and physical properties of the network are well known. The switch-to-switch wires, most likely the longest on the chip except for clock, power and ground wires, have all exactly the same well defined length. Potential uncertainties and irregularities are confined to the resource slots and have most likely no global impact. Thus, the regular and known geometry leads to the accurate analysis of electrical properties. Furthermore, since the NoC platform is reused in many products, elaborate electrical and power models can be developed e.g. to model the dynamic power consumption caused by the communication in the network. High level traffic pattern models can be combined with electrical models for the better assessment of noise and power consumption.

Design and verification time. As already elaborated above, reuse will decrease uncertainties and risk in particular for the verification tasks. This naturally makes design and verification time more predictable. Another reason for increased predictability of the design process is reduced design freedom. Since the network structure and basic services are fixed as well as the size of resource slots, the design space is significantly reduces and the remaining tasks are well separated resulting in more smaller and independent tasks. The remaining system level architectural decisions are the selection of the platform, the network size and the resource allocation. Then all the resources can be designed and implemented separately. Hence, the main challenge is the development of the overall system functionality, its partitioning into processes and their mapping onto resources. However, due to the relatively fixed architecture and the predictable performance and electrical properties (see above) the behavioural design is to a large extent independent from

the implementation phase. This increased modularization of the design tasks makes the overall process more predictable.

5. Disadvantages

Obviously, all these nice properties, outlined above, do not come for free. Essentially we pay for each of them by loosing optimality. Reusing components always means that we use something more general, thus less optimal, than necessary for a particular task. Adopting a fixed and relatively inflexible network topology means that we exclude all the other topologies that may be more suitable for the problem at hand. A particular protocol stack with associated services will not be ideal for all embedded systems and any NoC operating system will often be too general with unnecessary overhead and lacking important services at the same time. Only time will tell if it is possible to find a sensible balance between generality and efficiency which fits sufficiently many applications.

One way to alleviate the problem may be offered by a layered protocol stack and a layered set of services. An application may select a NoC platform only up to a particular level of the stack, thus avoiding the overhead of the upper layers. This flexibility however comes at a price. A strict layering of protocols and services may not be the optimal solution. An integrated implementation of several layers may result in a more efficient design at the expense of the possibility to have direct access to the lower layers.

6. Effect of NoC on the design process

Let us briefly review a design process based on a NoC platform. The main activities are:

Configuring the platform. Depending on the available configuration options this phase may be extensive or insignificant. A more general platform suitable for a wider range of applications will have more configuration options.

Every platform will allow to select the size and most likely offer a selection of communication services that can be included. Every design will need the core service for transporting raw data from sending to receiving resources. This core service may be packet oriented or based on virtual channels. More sophisticated communication means with varying levels of fault tolerance, bandwidth and latency control can be optional. Still higher level services could support transparent task migration from one resource to another or a virtual global shared memory.

Selecting resources. Since every resource slot can receive an arbitrary resource, all the resources have to be chosen by the designer. There may be some constraints because network services like a NoC operating system have to be implemented also. Part of the resource selection task is the design of the memory architecture and I/O architecture, which require specific resources distributed in a particular way.

Reuse of features. The selection of resources is of course intimately connected to the reuse of features. In our terminology a feature is a particular functionality together with its implementation. E.g. a speech analysis feature can be modeled as a task graph and implemented on one or several DSP processors or custom hardware blocks. Thus, a feature can occupy several resource slots but it can also share a resource, e.g. a DSP, with other features. Resource sharing of features has to be considered carefully because it will most likely compromise the property of arbitrary composability unless precautions are taken.

Evaluation and integration. The integration of all features and resources into the final system, the evaluation of performance and the verification of functionality is the final and perhaps most challenging task. An efficient and elaborate simulation and prototyping environment, that can be shared by many design projects, will significantly aid the successful system integration. But above all, the simplicity and predictability of the interfaces of resources, the communication network and features at all levels of the protocol stack will determine how well the property of linear effort can be approximated.

7. Conclusion

Developing a system with several dozens or hundreds of processor like resources is a formidable task. It can only be accomplished if features and components are aggressively reused and if they are *arbitrarily composable* at the physical level, at the architectural and structural level and at the application level (figure 1.3). Or, in other words, the protocol stack from the physical layer to the network and transport layer to the application layer must be well defined such that arbitrary components and features can be connected to the NoC backbone via the protocol stack without affecting the rest of the system.

We have focused on two issues which are particularly crucial for providing the properties of arbitrary composition and linear effort. *Reuse* from blocks to services and features allows designers to productively combine existing parts to new configurations and innovative products. *Predictability* makes reuse efficient because adding new components and

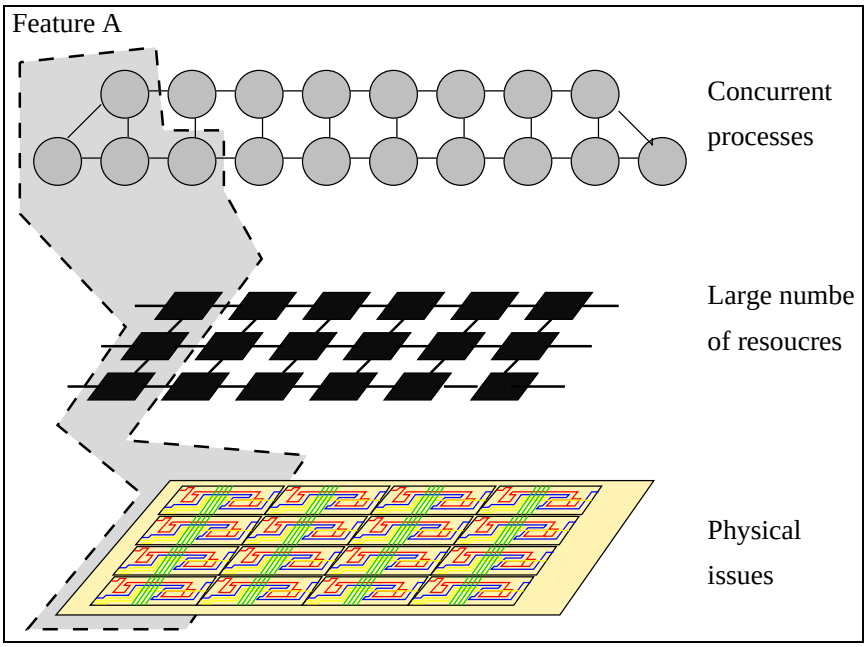

Figure 1.3. Features and components must be arbitrarily composable at the physical, the architectural and the application level to facilitate seamless feature integration.

features does not require redesigning and reverifying the existing parts, thus avoiding a more than linear growth of the design effort.

If NoC based platforms are able to provide the arbitrary composability and the linear effort properties while still allowing for sufficiently efficient product implementations, the design of systems-on-chip will be revolutionized, eventually leading to a "design by feature combination". The NoC paradigm is highly suited to provide SoC platforms scalable and adaptable over several technology generations because it opens the opportunity to define and standardize a communication service infrastructure and a protocol stack from the physical to the application layer. If these are well defined and facilitate the arbitrary composability and the linear effort properties, it will allow the efficient reuse of large resources, communication and network services, and application features. The TCP/IP protocol stack and the Internet, which also revolutionized the use of computers, can be an inspiring example. Indeed, due to their inherent scalability NoC platforms may allow the design productivity to grow as fast as technology capabilities and may eventually close the design productivity gap.

References

[1] Semiconductor Industry Association. Interbational technology roadmap for semiconductors. Technical report, World Semiconductor Council, 1999. Edition 1999.

[2] James A. Rowson and Alberto Sangiovanni-Vincentelli. Interface-based design. In *Proc. of the 34th Design Automation Conference*, 1997.

[3] Kurt Keutzer, Sharad Malik, Richard Newton, Jan Rabaey, and Alberto Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Trasnactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1523–1543, Decmber 2000.

[4] Marco 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 *Proceedings of the 38th Design Automation Conference*, June 2001.

[5] Dennis Sylvester and Kurt Keutzer. Getting to the bottom of deep submicron. In *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 203–211, 1998.

[6] Dennis Sylvester and Kurt Keutzer. Getting to the bottom of deep submicron ii: a global wiring paradigm. In *Proceedings of the 1999 International Symposium on Physical Design*, pages 193–200, 1999.

[7] Thomas Meincke, Ahmed Hemani, S. Kumar, P. Ellervee, J. Öberg, T. Olsson, P. Nilsson, D. Lindqvist, and H. Tenhunen. Globally asynchronous locally synchronous architecture for large high performance ASICs . In *Proc. of IEEE Int. Symp. on Circuits and Systems (ISCAS)*, volume II, pages 512–515, Orlando, USA, May 1999.

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

[9] Shashi Kumar, Axel Jantsch, Juha-Pekka Soininen, Martti Forsell, Mikael Millberg, Johnny Öberg, Kari Tiensyrjä, and Ahmed Hemani. A network on chip architecture and design methodology. In *Proceedings of IEEE Computer Society Annual Symposium on VLSI*, April 2002.

[10] William J. Dally and Brian Towles. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the 38th Design Automation Conference*, June 2001.

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

[12] Drew Wingard. MicroNetwork-based integration of SOCs. In *Proceedings of the 38th Design Automation Conference*, June 2001.

[13] K. Goossens, J. van Meerbergen, A. Peeters, and P. Wielage. Networks on silicon: Combining best-effort and guaranteed services. In *Proceedings of the Design Automation and Test Conference*, March 2002.

[14] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Paul Johnson Henry Hoffman, Jae-Wook Lee, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Matt Frank Volker Strumpen, Saman Amarasinghe, and Anant Agarwal. The Raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE Micro*, 22(2):25–35, March/April 2002.

[15] Michael Keating and Pierre Bricaud. *Reuse Methodology Manual for System-on-Chip Designs*. Kluwer Academaic Publishers, 1998.

[16] Terry Thomas. Technology for ip reuse and portability. *IEEE Design & Test of Computers*, 16(4):6–15, October 1999.

[17] Henry Chang, Larry Cooke, Merrill Hunt, Grant Martin, Andrew McNelly, and Lee Todd. *Surviving the SOC Revolution - A Guide to Platform-Based Design*. Kluwer Academic Publishers, 1999.