Networks on chips: A new paradigm for component-based MPSoC design

L. Benini G. De Micheli

DEIS Università di Bologna CSL Stanford University
Bologna, Italy 40136 Stanford, CA 94305

1 Introduction

We consider *multi-processor systems on chips* (MPSoCs) that will be designed and produced in *deep submicron technologies* (DSM) with minimal features in the $[100 - 25]nm$ range. Such systems will be designed using pre-existing components, such as processors, controllers and memory arrays. The major design challenge will be to provide a functionally-correct, reliable operation of the interacting components. The physical interconnections on chip will be a limiting factor for performance and possibly for energy consumption.

In this chapter, we address a methodology for the design of the interconnection among components, that satisfies the needs for modular and robust design, under the constraints provided by the physical interconnect.

1.1 Technology trends

In the projections of future silicon technologies, the operating frequency and transistor density will continue to grow, making energy dissipation and heat extraction a major concern. At the same time, supply voltages will continue to decrease, with adverse impact on signal integrity. The voltage reduction, even though beneficial, will not suffice to mitigate the energy consumption problem, where a major contribution will be due to leakage. Thus, SoCs will incorporate *Dynamic Power Management* (DPM) techniques in various forms to satisfy energy consumption bounds [9].

Global wires, connecting different functional units, are likely to have propagation delays largely exceeding the clock period [33]. Whereas signal pipelining on interconnections will become common practice, correct design will require knowing the signal delay with reasonable accuracy. Indeed, a negative side effect of technology downsizing will be the spreading of physical parameters (e.g., variance of wire delay per unit length) and its relative importance as compared to the timing reference signals (e.g., clock period).

Synchronization of future chips with a single clock source and negligible skew will be extremely hard or impossible. The most likely synchronization paradigm for future chips is *globally-asynchronous locally-synchronous* (GALS), with many different clocks. Global wires will span multiple clock domains, and synchronization failures in communicating between different clock domains will be rare but unavoidable events [20].

1.2 Non-determinism in SoC abstraction models

As SoCs complexity scales, it will be more difficult, if not impossible, to capture their functionality with fully deterministic models of operation. For example, a transaction request may not be satisfied within a fixed worst-case time window, unless this window is so large to make the constraint useless. On the other hand, average

response time will be a more realistic objective to pursue. Similarly, high-level abstraction of components will lead to non-deterministic models. For example, the time for context restoration in a processor waking up from sleep mode cannot always be assessed deterministically at design time, because it often depends on the amount of state information that needs to be restored to resume operation.

SoC designs will rely on tens, if not hundreds of information processing components that can dialogue among each other. Global control on the information traffic will be unlikely to succeed, because of the need of keeping track of the states of each component. Thus components will initiate autonomously data transfer according to their needs. The global communication pattern will be fully distributed, with little or no global coordination. Overall, SoC design will be based on both deterministic and stochastic models. There will be a definite trend in quantifying design objectives (e.g., performance, power) by probabilistic metrics (e.g., average values, variance). This trend will lead to a major change in design methodologies. Engineers are not new to large-scale design under stochastic models. Networks are examples of systems that are well understood and successfully designed with stochastic techniques and models [15, 56].

1.3 A new design approach to SoCs

To tackle the above-mentioned issues we can borrow models, techniques and tools from the network design field and apply them to SoCs design. In other words, we view a SoC as a *micro-network* of components. The network is the abstraction of the communication among the components and has to satisfy *quality of service* (QoS) requirements (e.g., reliability, performance and energy bounds) under the limitation of intrinsic unreliable signal transmission and non-negligible communication delays on wires. We postulate that SoC interconnect design can be done using the *micro-network stack* paradigm, which is an adaptation of the protocol stack [56] (Figure 1). Thus the electrical, logic, and functional properties of the interconnection scheme can be abstracted.

SoCs differ from wide-area networks because of local proximity and because they are much more predictable at design time. Local, high-performance networks (such as those developed for large-scale multiprocessors) have closer characteristics. Nevertheless, SoCs place computational and storage elements very close to each other. The use of the same substrate for computing, storage and communication, the lack of bottlenecks (such as I/O pads) and the freedom in designing wiring channels yield different design constraints and opportunities.

Overall, the most distinctive characteristics of micro-networks are energy constraints and design-time specialization. Whereas computation and storage energy greatly benefits from device scaling (smaller gates, smaller memory cells), the energy for global communication does not scale down. On the contrary, projections based on current delay optimization techniques for global wires [53] show that global communication on chip will require increasingly higher energy consumption. Hence, communication-energy minimization will be a growing concern in future technologies. Furthermore, network traffic control and monitoring can help in better managing the power consumed by networked computational resources. For instance, clock speed and voltage of end nodes can be varied according to available network bandwidth.

Design-time specialization is another facet of the SoC network design problem that raises many new challenges. Macroscopic networks emphasize general-purpose communication and modularity. Communication network design has traditionally been decoupled from specific end applications, and strongly influenced by standardization and compatibility constraints with legacy network infrastructures. In SoCs networks, these constraints are less restrictive. The communication network fabric is designed on silicon from scratch. Standardization is needed only for specifying an abstract network interface for the end nodes, but the network architecture itself can be tailored to the application, or class of applications, targeted by the SoC design. Hence, we envision a vertical design flow where every layer of the micro-network stack is specialized and optimized for the target application domain. Such *application-specific on-chip network synthesis* paradigm represents, in our view, an open and exciting research field. Needless to say, specialization does not imply complete loss of flexibility. From a design stand-point, network reconfigurability will be key in providing plug-and-play use of components, since they will interact with the others through (reconfigurable) protocols.

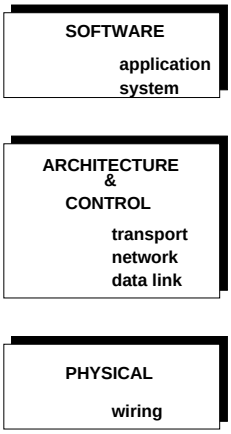

Figure 1: Micro-network stack

2 Signal transmission on chip

We consider now on-chip communication and its abstraction as a micro-network. We analyze the physical layer of the micro-network stack in this section. Wires are the physical realization of communication channels in SoCs. (For our purposes, we view busses as ensembles of wires.) Even though components may be hierarchically designed, we shall focus on components and their interconnection with *global wires*. We neglect the internals of components and their wiring by *local wires*, because the properties of such wires can be made to scale with the technology, and it is likely that present design styles and methods will still apply [53].

2.1 Global wiring and signalling

Global wires are the physical implementation of the communication channels. The technologies under consideration will exceed 10 wiring levels. Most likely, global wires will be routed on the top metal layers provided by the technology. The pitch (i.e., width plus separation) of such wires is *reverse scaled*, i.e., wiring pitch increases (with respect to device pitch) with higher wiring levels. Wire widths also grow with wiring levels. Wires at top levels can be much wider and thicker than low-level wires [54]. There are several good reasons for routing global communication on top wires. Increased pitch reduces cross-coupling and offers more opportunities for wire shielding, thereby improving noise immunity. Increased width reduces wire resistance (even considering the skin effect), while at the same time increased spacing around the wire prevents capacitance growth. At the same time, inductive effects will grow relatively to resistance and capacitance [23]. As a result, future global wires are likely to be modeled as lossy transmission lines, as opposed to today's lumped or distributed RC models.

Physical layer signalling techniques for lossy transmission lines have been studied for a long time by high-speed board designers and microwave engineers [7, 20]. Non-negligible line resistance causes signal attenuation and dispersion in frequency of fast signals, thereby increasing intersymbol interference, and ultimately limiting communication bandwidth. The impact of attenuation and frequency dispersion can be reduced by splitting wires in several sections with signal regeneration buffers in between. Buffering significantly improves bandwidth, and it is also beneficial for latency, especially when the RC component is dominant [33]. Line inductance requires careful impedance matching at the transmitting or receiving end of the line, in order to avoid signal reflections that largely increase intersymbol interference and noise [20]. Traditional rail-to-rail voltage signalling with capacitive termination, as used today for on-chip communication, is definitely not well-suited for high-speed, low-energy communication on future global interconnect [20].

In light of these trends, we believe that physical-layer signalling techniques for on-chip global communication will evolve in the near future. High-speed communication is achieved by reducing the voltage swing, which has also a beneficial effect on power dissipation. A reduced swing, current-mode transmitter requires careful receiver design,

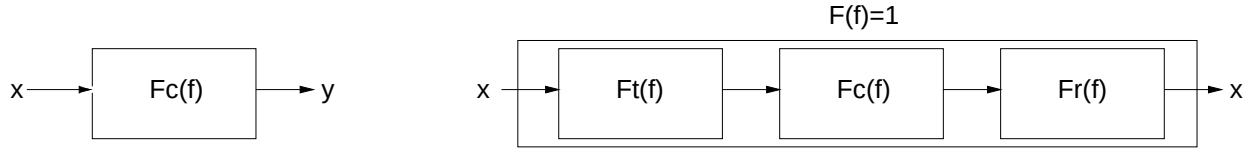

Figure 2: Channel transfer function and equalization

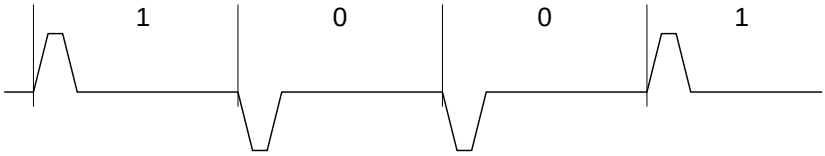

Figure 3: Current-pulse signalling: a simple AM scheme.

with good adaptation to line impedance, and high-sensitivity sensing, possibly with the help of sense amplifiers.

Example 1 *Low swing drivers and receivers are already common in current integrated circuits. The bit-lines of fast SRAMs use a differential, pre-charged, low-swing signalling scheme to improve speed and energy efficiency. The transmitter is made of the precharge transistors (one for each bit line), and the selected cell. The receiver is the sense amplifier at the end of bit-line pairs. During SRAM readout, the addressed cell is connected to the bit line, and it draws a small, almost constant discharge current (the cell's transistors are in saturation). A small differential deviation (e.g., 100 mV) from the precharged value on the bit lines is sufficient for the sense amplifier to detect the information stored in the addressed cell, and to restore it to full-swing logic values. A well-designed SRAM read circuit based on this signalling scheme is more than ten times faster and approximatively two orders of magnitude more energy-efficient than single ended, rail-to-rail signaling [20].*

Example 2 *Dynamic voltage scaling can be applied to interconnect wires, with the objectives of lowering the voltage swing as much as possible, while preserving correctness of the transmitted information. Worm et al. [61] devised a feedback scheme by which voltage swing on wires (busses) for a communication link is determined on-line by monitoring the error rate of the information. In particular, this scheme supports packetized information transmission, with retransmission of the corrupted packets. As retransmission rate increases over (decreases below) a threshold, the voltage swing is increased (decreased).*

This scheme is useful in setting the voltage swing according to the operating conditions of the SoC (environmental noise, temperature), as well as for self-calibrating the SoC with respect to technological parameters, which may have a wide spread in future technologies.

In general, transmitter and receiver design for on-chip communication should follow the well-known channel equalization paradigm employed in digital communication, as shown in Figure 2. In this approach, the channel is abstracted by a transfer function $F_C(f)$ in the frequency domain. The transmitter and the receiver should *equalize* the transfer function by jointly implementing its inverse: $F_T(f) \cdot F_R(f) = F_C^{-1}(f)$. Notice that, in general, effective channel equalization may require non-linear transformations of the signal by transmitter and receiver.

Signal *modulation* is probably the simplest non-linear transformation that could be applied to optimize channel efficiency. Currently, no modulation is employed for standard on-chip communication, but several modulation schemes are being investigated, e.g., [64]. Modulation can greatly enhance the effective communication bandwidth: (i) by shifting the information content of a signal out of frequency regions where the channel does not transmit reliably; (ii) by enabling multiple channels to be carried over the same wire.

Example 3 *Current-pulse signalling [20], as shown in Figure 3, is a simple example of amplitude modulation, where the carrier is a square current waveform with duty cycle much smaller than 50%. The amplitude of the*

waveform is modulated by multiplying it by -1 (a negative current pulse) when transmitting a zero, by $+1$ (a positive current pulse) when transmitting a one.

A more complex example of on-chip modulation has been presented by Yoshimura et al. [64], who introduced a new multi-master bus architecture where several transmitters share the same bus, by modulating their output data using a code-division multiple access (CDMA) scheme. The frequency spreading and de-spreading is achieved by multiplying the information by a pseudorandom sequence generated by linear feedback shift registers (LFSRs) at both the transmitting and receiving ends. Thus, superimposed signals can be discriminated by just knowing the characteristics of the LFSRs. In this approach, multiple simultaneous bus driving is achieved by using charge pumps in the bus drivers.

Non-linear transformations can also help in increasing communication energy efficiency. In most current on-chip signalling schemes power is consumed during signal transitions. Thus, signal encoding (a non-linear transformation) for reduced transition activity can help optimize communication energy. Numerous encoding techniques have been proposed for this purpose [11].

2.2 Signal integrity

With present technologies, most chips are designed under the assumption that electrical waveforms can always carry correct information on chip. We believe that guaranteeing error-free information transfer (at the physical level) on global on-chip wires will become harder, if not impossible, in future technologies because of several reasons:

- Signal swings are going to be reduced, with a corresponding reduction of voltage noise margins.
- *Cross-talk* is bound to increase, and the complexity of avoiding cross-talk by identifying all potential on-chip noise sources will make it unlikely to succeed fully.
- *Electromagnetic interference* (EMI) by external sources will become more of a threat because of the smaller voltage swings and smaller dynamic storage capacitances.
- The probability of occasional *synchronization failures* and/or metastability will rise. These erroneous conditions are possible during system operation, because of transmission speed changes, local clock frequency changes, timing noise (jitter), etc.
- Alpha particles (from the package) and collisions of thermal neutrons (from cosmic ray showers) can create spurious pulses, which can affect signals on chip and/or discharge dynamic storage capacitances. This problem, known as *soft errors*, used to be relevant only for large DRAMs. It is now conjectured to be a potential hazard for large SoCs as well [50]

Example 4 *A soft error occurs when a particle induces the creation of electron-hole pairs, corresponding to an electric charge that is larger than a critical charge. The critical charge is a characteristic of the cell node (e.g., memory or register node), technology and operating voltage. Typical values range from 10 to 100 pC in current technologies. The shrinking of layout geometries and corresponding reduction of node capacitances, as well as the reduction in supply voltages, contribute to a reduction of the critical charge. Recent experiments [17] have shown that the soft error rate can grow by two order of magnitudes in the current decade.*

Soft errors may be permanent or transient. Transient soft errors can be seen as voltage pulses, that propagate through a circuit when the pulse width is larger than the transition times of logic gates. The arrival of a pulse at a latch boundary when a clock transition occurs may cause the wrong information to be stored. Nicolaidis et al. [46] developed and commercialized error correcting techniques based on timing redundancy. Latches are replaced by latch pairs, with identical inputs and whose outputs feed comparators. One latch has a delay line on its input. Thus, the comparator detects spurious pulses that arrive at the latch boundary. Standard error recovery techniques can then be used when a soft error is detected.

From the aforementioned considerations, we conclude that it will not be possible to abstract the physical layer of on-chip networks as a fully reliable, fixed-delay channel. At the micro-network stack layers above the physical one, we can view the effects of synchronization losses, crosstalk, noise, ionizing radiation, etc., as a source of local transient malfunctions. We can abstract malfunctions as *upsets*. This is a simple and yet powerful model that parallels the *stuck-at fault* model in the testing field. In particular, an upset is the abstraction of a malfunction that causes the wrong binary information to be read or stored. Upset modeling consists of deriving analytical models of crosstalk and radiation-induced disturbances, and derive the upset probability. Upset probability can vary over different physical channels and over time. It is important to understand that upset probability is negligible (but not null) in current technologies, but it will not be so in future technologies.

A number of new design challenges are emerging at the micro-network physical layer, and a paradigm shift is needed to effectively address them. Indeed, current design styles consider wiring-related effects as undesirable parasitics, and try to reduce or cancel them by specific and detailed physical design techniques. It is important to realize that a well-balanced design should not over-design wires so that their behavior approaches an ideal one, because the corresponding cost in performance, energy-efficiency and modularity may be too high. Physical layer design should find a compromise between competing quality metrics and provide a clean and complete abstraction of channel characteristics to micro-network layers above.

3 Micro-network architecture and control

Network design entails the specification of *network architectures* and *control protocols*. The architecture specifies the topology and physical organization of the interconnection network, while the protocols specify how to use network resources during system operation. Even though architecture and protocol design are tightly intertwined, we shall first focus on different network architectures and then will address protocol design issues.

Network design is influenced by many design metrics. Whereas both micro-network and general network design must meet performance requirements, on-chip network implementations will be differentiated by the need of satisfying tight energy bounds. Design choices are strongly impacted by physical channel characteristics as well as by expected workloads. In many cases it is possible to extract some information on usual communication patterns and workload. This information can be used to optimize some design parameters. For example, for SoC executing embedded software programs, the characteristics of the embedded instruction stream can be modeled and used for the memory-processor interconnection design [10]. When no expected workload information is available, network design should emphasize robustness.

3.1 Interconnection network architectures

On-chip networks are closely related to interconnection networks for high-performance parallel computers with multiple processors, where processors are individual chips. Similarly to multiprocessor interconnection networks, nodes are physically close and link reliability is high. Furthermore, multiprocessor interconnections have traditionally been designed under stringent bandwidth and latency constraints to support effective parallelization. Similar constrains will drive the design of micro-networks. For these reasons we survey different on-chip network architectures using the same taxonomy proposed by Duato et al. [25] for multiprocessor interconnections. We show that current on-chip interconnections can be classified in a similar way and that their evolution follows a path similar to multiprocessor interconnection architectures.

3.1.1 Shared-medium networks

Most current SoC interconnection architectures fall within the *shared medium* class [3, 18, 51, 57, 60]. These are the simplest interconnect structures, in which the transmission medium is shared by all communication devices. Only one device can drive the network at a time. Every device connected to the network has a network interface, with requester, driver, and receiver circuits. The network is usually passive and it does not generate control or

data messages. A critical issue in the design of shared-medium networks is the *arbitration strategy* that assigns the mastership of the medium and resolves access conflicts. A distinctive property of these networks is the support for broadcast of information, which is very advantageous when communication is highly asymmetric, i.e., the flow of information originates from few transmitters to many receivers.

Within current technologies, the most common embodiment of the on-chip, shared-medium structure is the *backplane bus*. This is a very convenient, low-overhead interconnection for a small number of active processors (i.e., bus masters) and a large number of passive modules (i.e., bus slaves) that only respond to requests from bus masters. The information units on a bus belong to three classes, namely: *data, address* and *control*. Data, address, and control information can either be time-multiplexed on the bus, or they can travel over dedicated busses/wires, spanning the tradeoff between performance and hardware cost (area). Most on-chip busses value performance and use a large number of dedicated wires.

A critical design choice for on-chip busses is synchronous versus asynchronous operation. In synchronous operation, all bus interfaces are synchronized with a common clock, while in asynchronous busses all devices operate with their own clock and use a handshaking protocol to synchronize with each other. The tradeoffs involved in the choice of synchronization approach are complex and depend on a number of auxiliary constraints, such as testability, ease of debugging and simulation, and the presence of legacy components. Currently, all commercial on-chip busses are synchronous, but the bus clock is slower than the clock of fast masters. Hence, simplicity and ease of testing/debugging is prioritized over sheer performance.

Bus arbitration mechanisms are required when several processors attempt to use the bus simultaneously. Arbitration in current on-chip busses is performed in a centralized fashion by a bus arbiter module. A processor wishing to communicate must first gain bus mastership from the arbiter. This process implies a control transaction and communication performance loss, hence arbitration should be as fast as possible and as rare as possible. Together with arbitration, the response time of slow bus slaves may cause serious performance losses, because the bus remains idle while the master waits for the slave to respond. To minimize the waste of bandwidth, *split transaction protocols* have been devised for high performance busses. In these protocols, bus mastership is released just after a request has completed, and the slave has to gain access to the bus to respond, possibly several bus cycles later. Thus the bus can support multiple outstanding transactions. Needless to say, bus masters and bus interfaces for split transaction busses are much more complex than those of simple atomic-transaction busses. Even though the majority of current on-chip busses support only atomic transactions, some split transaction organizations have begun to appear [1].

Example 5 *The AMBA 2.0 bus standard has been designed for the ARM processor family [3]. It is fairly simple and widely used today. Within AMBA, the high-performance bus protocol (AHB) connects high-performance modules, such as ARM cores and on-chip RAM. The bus supports separate address and data transfers. A bus transaction is initiated by a bus master, which requests access from a central arbiter. The arbiter decides priorities when there are conflicting requests. The arbiter is implementation specific, but it must adhere the ASB protocol. Namely, the master issues a request to the arbiter. When the bus is available, the arbiter issues a grant to the master. Arbitration address and data transfer are pipelined in AM*BA AHB* to increase the bus effective bandwidth, and burst transactions are allowed to amortize the performance penalty of arbitration. However, multiple outstanding transactions are supported only to a very limited degree: the bus protocol allows a split transaction, where a burst can be temporarily suspended (by a slave request). New bus transactions can be initiated during a split burst.*

The Lucent Daytona chip [1] is a multi-processor on a chip that contains four 64-bit processing elements. These elements generate transactions of different sizes. Thus, a special 128-bit split-transaction bus was chosen. The bus was designed to minimize the average latency when simultaneous requests are made. Large transfers are partitioned into smaller packets enabling bus bandwidth to be better utilized. Each transaction is associated with an ID. Thus, multiple outstanding transactions can be serviced and be prioritized under program control. Thus feature is critical for system performance. In fact, the recently announced most advanced version of the AMBA bus protocol (AMBA AXI) fully supports multiple outstanding transactions.

Shared-medium network organizations are well-understood and widely used, but their scalability is seriously limited. The bus-based organization is still convenient for current SoCs that integrate less than five processors

and rarely more than ten bus masters. Multiprocessor architects have realized long ago that bus-based systems are not scalable, as the bus invariantly becomes the bottleneck when more processors are added. Another critical limitation of shared-medium network is their energy inefficiency. In these architectures, every data transfer is broadcast. Broadcasts must reach each possible receiver at a large energy cost. Future integrated systems will contain tens to hundreds of units that generate information to be transferred. For such systems, a bus-based network would become the critical performance and power bottleneck.

3.1.2 Direct and indirect networks

The *direct* or *point-to-point* network is a network architecture that overcomes the scalability problems of shared-medium networks. In this architecture, each node is directly connected with a subset of other nodes in the network, called *neighboring* nodes. Nodes are on-chip computational units, but they contain a network interface block, often called a *router*, which handles communication-related tasks. Each router is directly connected with the routers of the neighboring nodes. Different from shared-medium architectures, as the number of nodes in the system increases, the total communication bandwidth also increases. Direct interconnect networks are therefore very popular for building large-scale systems.

Indirect or *switch-based* networks are an alternative to direct networks for scalable interconnection design. In these networks, a connection between nodes has to go through a set of *switches*. The network adapter associated with each node connects to a port of a switch. Switches do not perform information processing. Their only purpose is to provide a programmable connection between their ports, or, in other words, to set up a communication path that can be changed over time [25]. Interestingly enough, the distinction between direct and indirect networks is blurring, as routers (in direct networks) and switches (in indirect networks) become more complex and absorb each other's functionality.

Most current *field-programmable gate arrays* (FPGAs) consist of a homogeneous fabric of programmable elements connected by a switch-based network. FPGAs can be seen as the archetype of future programmable SoCs: they contain a large number of interconnected computing elements. Current FPGAs communication networks differ from future SoC micro-networks in *granularity* and *homogeneity*.

The concept of dynamic reconfigurability of FPGAs also applies to the design of micro-networks. SoCs benefit from programmability in the field (e.g., to match environmental constraints) and to run-time reconfiguration (e.g., to adapt to a varying workload). Reconfigurable micro-networks exploit programmable routers and/or switches. Their embodiment may leverage multiplexers whose control signals are set by configuration bits in local storage, as in the case of FPGAs.

Processing elements in FPGAs implement simple bit-level functional blocks. Thus communication channels in FPGAs are functionally equivalent to wires that connect logic gates. Since future SoCs will house complex processing elements, interconnect will carry much coarser quanta of information. The different granularity of computational elements and communication requirements has far-reaching consequences on the complexity of the network interface circuitry associated with each communication channel. Interface circuitry and network control policies must be kept extremely simple for FPGAs, while they can be much more complex when supporting coarser-grain information transfers. The increased complexity will introduce also larger degrees of freedom for optimizing communication.

Example 6 *The Xilinx Spartan-II FPGA chips are rectangular arrays of configurable logic blocks (CLBs). Each block can be programmed to perform a specific logic function. CLBs are connected via a hierarchy of routing channels. Thus each chip has an indirect network over a homogeneous fabric.*

The RAW architecture [2] is a fully programmable SoC, consisting of an array of identical computational tiles with local storage. Full programmability means that the compiler can program both the function of each tile and the interconnections among them. The name RAW stems from the fact that the "raw" hardware is fully exposed to the compiler. To accomplish programmable communication, each tile has a router. The compiler programs the routers on all tiles to issue a sequence of commands that determine exactly which set of wires connect at every cycle. Moreover, the compiler pipelines the long wires to support high clock frequency. Thus, RAW can be viewed as a direct network.

The Xilinx Virtex-II are FPGAs with various configurable elements to support reconfigurable digital signal processor (DSP) design. The internal configurable rectangular array contains CLBs, RAMs, multipliers and clock managers. Programmable interconnection is achieved by routing switches. Each programmable element is connected to a switch matrix, allowing multiple connections to the general routing matrix. All programmable elements, including the routing resources, are controlled by values stored in static memory cells. Thus, Virtex-II can be also seen as an indirect network over a heterogeneous fabric.

From an energy viewpoint, direct networks have the potential for being more energy-efficient than shared medium networks, because energy per transfer on a point-to-point communication channel is smaller than that on a large shared-medium architecture. Clearly, communication between two end-nodes may go through several point-to-point links, and the comparison should take this effect into account [49]. When local communication between neighboring nodes dominates, the energy advantage of point-to-point networks is hardly disputable and some recent experimental work has quantified such advantage [41]. It is also interesting to observe that point-to-point networks are generally much more demanding in terms of area than shared-medium networks. In this case the positive correlation between area and power dissipation breaks down.

3.1.3 Hybrid networks

Direct and indirect networks yield the scalability required to support communication in future SoCs and they can provide, with the support of effective control techniques (discussed in the next subsection) the performance levels required to avoid communication bottlenecks. Current FPGAs show convincing evidence that multi-stage, scalable networks can be implemented on-chip, and current high-performance multiprocessors demonstrate that these advanced network topologies can sustain extremely high performance levels. However, the communication infrastructure of both FPGAs and multiprocessors are strongly oriented toward homogeneity in information transfer.

Homogeneous interconnection architectures have important advantages: they facilitate modular design, and they are easily scaled up by replication. Nevertheless, homogeneity can be an obstacle to flexibility and fine-tuning of architectures to application characteristics. While homogeneous network architectures may be the best choice for general-purpose computing, systems developed for a particular applications (or a class of applications) can benefit from a more heterogeneous communication infrastructure, that provides high bandwidth in a localized fashion only where it is needed to eliminate bottlenecks.

The advantages of introducing a controlled amount of non-uniformity in communication network design are widely recognized. Many heterogeneous, or *hybrid* interconnection architectures, have been proposed and implemented. Notable examples are *multiple-backplane* and *hierarchical* (or *bridged*) busses. These organizations enable clustering of tightly coupled computational units with high communication bandwidth, and provide lower bandwidth (and/or higher latency) inter-cluster communication links. Compared to a homogeneous, high-bandwidth architecture, they can provide comparable performance using a fraction of the communication resources, and at a fraction of the energy. Thus, energy efficiency, is a strong driver toward hybrid architectures.

Example 7 *The AMBA 2.0 standard [3] specifies three protocols: the high performance bus (AHB), the advanced system bus (ABS) and the advanced peripheral bus (APB). The latter is designed for lower-performance transactions related to peripherals. An AMBA-based chip can instantiate multiple bus regions which cluster components with different communication bandwidth requirements. Connections between two clusters operating with different protocols (and/or different clock speed) is supported through bus bridges. Bridges perform protocol matching and provide the buffering and synchronization required to communicate across different clock domains. Note that bridges are also beneficial from an energy viewpoint, because the high-transition rate portion of the bus is decoupled from the low-speed, low transition-rate, peripheral section.*

3.2 Micro-network control

Effective utilization of the physical realization of micro-network architectures depends on protocols, i.e., on network control algorithms which are often distributed. We shall describe control algorithms following the micro-

network stack of Figure 1 from the bottom up. Network protocols can be implemented either in software or in hardware. In our analysis, this distinction is immaterial, because we focus only on the various control functions, and not on how to implement them. In synthesis, network control is responsible for dynamically managing network resources during system operation, striving to provide the required quality of service.

3.2.1 Data Link layer

As seen in Section 2 and in Figure 1, on-chip global wires are the physical support for communication. The data-link layer abstracts the physical layer as an unreliable digital link, where the probability of bit upsets is non null. Such upset probabilities are increasing as technology scales down. Furthermore, we have seen that reliability can be traded off for energy. The main purpose of data-link protocols is to increase the reliability of the link up to a minimum required level, under the assumption that the physical layer by itself is not sufficiently reliable. Another important function of the data-link layer is to regulate the access to a shared-medium network, where contention for a communication channel is possible. Thus we can see this layer as the superposition of two sub-layers. The lower sublayer (i.e., closest to the physical channel) is called *media access control* (MAC), while the higher sublayer is the *data-link control* (DLC). The MAC regulates access to the medium, while the DLC increases channel reliability, e.g., by providing error detection and/or correction capabilities.

Error detecting and *correcting codes* (ECCs) can be used in different ways. When only error detection is used, error recovery involves the retransmission of the faulty bit or word. When using error correction, some (or all) errors can be corrected at the receiving end. Error detection and/or correction requires an encoder/decoder pair at the channel's end, whose complexity depends on the encoding being used. Obviously, error detection is less hardware intensive than error detection and correction. In both cases, a small delay has to be accounted for in the encoder and decoder. Data re-transmission has a price in terms of latency. Moreover, both error detection and correction requires additional (redundant) signal lines. When ECC techniques are used in conjunction with packetized information, the redundant lines can be avoided by adding the redundant information at the tail of the packet, thus trading off space for delay.

There are several error detecting and correcting codes. For example, *Berger* codes can detect all unidirectional codes. Other codes, like *Hamming* codes, address the issues of bidirectional errors, such as those found in noisy channels. Error detection may reveal single or multiple errors in a data fragment (e.g., word). Similarly, error correction can restore the correct information caused by single or multiple errors. When errors go undetected, their effect can be catastrophic. When errors are detected and cannot be corrected (e.g., multiple errors with a single-error correction scheme), data retransmission has to be used. In the general case, one has often to admit that some errors may go undetected, and the designer's goal is to make this event very unlikely. In practice, one can choose a *mean time to failure* (MTTF) as a design specification. For example, one can chose an encoding scheme that has a MTTF of ten years, much above the SoC projected lifespan.

Latency and energy consumption for data transmission using ECC are very important design parameters. While the former has been studied for long time, the latter has been object of research only recently, especially in view of the fact that energy consumption in micro-networks has to be curtailed. Overall, energy depends on the switching activity in the communication link (including the redundant lines), on the encoder and decoder, on the retransmission scheme, and of course on the noisiness of the channel.

Example 8 *Bertozzi et al. [14] applied error correcting and detecting codes to an AMBA 2.0 AHB bus and compared the energy consumption in five cases: (1) original unencoded data; 2) single-error correction (SEC), (3) single-error correction and double-error detection (SECDEC), (4) multiple-error detection (ED), (5) parity. Hamming codes were used. Note that in case 3, a detected double error requires retransmission. In case 4, using* (n, k) *linear codes,* $2^n - 2^k$ *errors patterns of length* n *can be detected. In all cases, some errors may go undetected and be catastrophic. Using the property of the codes, it is possible to map the MTTF requirement into bit upset probabilities, and thus comparing the effectiveness of the encoding scheme in a given noisy channel (characterized by the upset probability) in meeting the MTTF target.*

Bertozzi investigated the energy efficiency of various encoding schemes. We summarize here one interesting case, where three assumptions apply. First, wires are long enough so that the corresponding energy dissipation

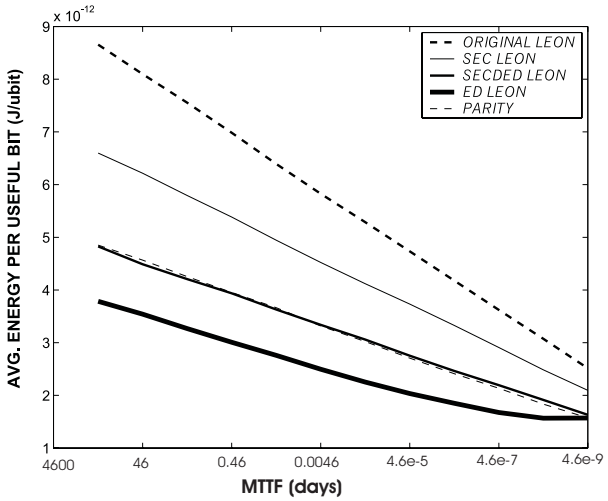

Figure 4: Energy efficiency for various encoding schemes on an AMBA bus connecting a memory controller to an I-cache of a LEON processor.

dominates encoding/decoding energy. Second, voltage swing can be lowered until the MTTF target is met. Third, upset probabilities are computed using a white Gaussian noise model [32].

Figure 4 shows the average energy per useful bit as a function of the MTTF (which is the inverse of the residual word error probability). In particular, for reliable SoCs (i.e., for MTTF = 1 year), multiple-error detection with retransmission is shown to be more efficient than error-correcting schemes. We refer the reader to [14] for results under different assumptions.

An additional source of errors, not considered in Section 2, is contention in shared-medium networks. Contention resolution is fundamentally a non-deterministic process, because it requires synchronization of a distributed system, and for this reason it can be seen as an additional noise source. In general, non-determinism can be virtually eliminated at the price of some performance penalty. For instance, centralized bus arbitration in a synchronous bus eliminates contention-induced errors, at the price of a substantial performance penalty caused by the slow bus clock and by bus request/release cycles.

Future high-performance shared-medium on-chip micro-networks may evolve in the same direction as high-speed local area networks, where contention for a shared communication channel can cause errors, because two or more transmitters are allowed to concurrently send data on a shared medium. In this case, provisions must be made for dealing with contention-induced errors.

An effective way to deal with errors in communication is to packetize data. If data is sent on an unreliable channel in packets, error containment and recovery is easier, because the effect of errors is contained by packet boundaries, and error recovery can be carried out on a packet-by-packet basis. At the data link layer, error correction can be achieved by using standard error correcting codes that add redundancy to the transferred information. Error correction can be complemented by several packet-based error detection and recovery protocols, such as *alternating-bit*, *go-back-N*, *selective repeat*, which have been developed for macroscopic networks [56, 15]. Several parameters in these protocols (e.g., packet size, number of outstanding packets, etc.) can be adjusted depending on the goal to achieve maximum performance at a specified residual error probability and/or within given energy consumption bounds. The choice of protocol and optimization of its parameters for micro-networks is an open problem.

We stress that contention for communication resources is unavoidable in all scalable network architectures. Contention-free communication is achievable either at the price of a large performance penalty (through global synchronization and centralized arbitration), or at the price of a large and poorly scalable contention-free interconnection architecture (e.g., a complete crossbar). Hence, almost every practical on-chip network will have to

account for the effects of contention caused by shared channels. Packetization of information flow is needed to effectively deal with contention: it limits the effects of contention-related errors, but it is also instrumental in implementing fair sharing of communication resources. For these reasons, we believe that future on-chip micro-networks will be packet-based, and designers will need guidelines and tools to decide packetization granularity and to trade off packet data and control content.

Example 9 *The scalable programmable integrated network (SPIN) is an on-chip micro-network [5, 31] that defines packets as sequences of 36-bits words. The packet header fits in the first word. A byte in the header identifies the destination (hence, the network can be scaled up to 256 terminal nodes), and other bits are used for packet tagging and routing information. The packet payload can be of variable size. Every packet is terminated by a trailer, which does not contain data, but a checksum for error detection. Packetization overhead in SPIN is 2 words. The payload should be significantly larger than 2 words to amortize the overhead. A SPIN on-chip network consists of two macro-cell circuits, a router and two wrappers respecting the virtual component interface (VCI) standard. A prototype implementation is described in [5].*

3.2.2 Network layer

The data link layer implements a reliable (packet-based) link between processing elements connected to a common link. The network layer implements the end-to-end delivery control in advanced network architectures with many communication channels. In most current on-chip networks, all processing elements are connected to the same channel, namely, the on-chip bus. Under these conditions, the network layer is empty. However, when processing elements are connected through a collection of links, we must decide how to set up connection between successive links, and how to route information from its source to the final destination, through a series of nodes in between. These key tasks, called *switching* and *routing*, are specific to the network layer, and they have been extensively studied both in the context of multiprocessor interconnects [25] and in the context of general communication networks [56, 15].

Switching algorithms can be grouped in four classes: *circuit switching*, *packet switching*, *cut-through switching* and *wormhole switching*. With circuit switching, a path from the source to the destination is reserved prior to the transmission of data, and the network links on the paths are released only after the data transfer has been completed. Circuit switching is advantageous when traffic is characterized by infrequent and long messages, because communication latency and throughput on a fixed path are generally very predictable. With circuit switching, network resources are kept busy for the duration of the communication, and the time for setting up a path can produce a sizable initial latency penalty. Hence, circuit switching is not widespread in packet networks where atomic messages are data packets of relatively small size: communication path setup and reset would cause unacceptable overhead and degrade channel utilization.

Packet switching addresses the limitations of circuit switching for packetized traffic. The data stream at the source is divided into packets, and the time interval between successive packets may change. A packet is completely buffered at each intermediate switch (or router) before it is forwarded to the next. This approach is also called *store-and-forward*, and it introduces non-deterministic delays due to message queueing at each intermediate node. Packet switching leads to significantly higher utilization of communication resources with respect to circuit switching, at the price of an increase in non-determinism. Furthermore, many packets belonging to a message can be in the network simultaneously and out-of-order delivery to the destination is possible.

Packet switching is based on the assumption that an entire packet has to be completely received before it can be routed to an output channel. This assumption implies that the switch, or the router, must provide significant storage resources. Furthermore, each intermediate node adds latency to the message, because it needs to fully receive a packet before forwarding it toward the next node. Cut-through switching strategies have been developed to reduce buffering overhead and latency. In cut-through switching schemes, packets are routed to the output channel and they can start leaving the switch before they are completely received in the input buffer. Improved performance and memory usage are counterbalanced by increased message blocking probability under heavy traffic, because a single packet may occupy input and output ports in several switches. As a result, non-determinism in message delivery increases.

Wormhole switching was originally designed for parallel computer clusters [25] because it achieves small network delay and requires little buffer usage. In wormhole switching, each packet is further segmented into *flits* (flow control units1). The header flit reserves the routing channel of each switch, the body flits will then follow the reserved channel, the tail flit will later release the channel reservation. One major advantage of wormhole routing is that it does not require the complete packet to be stored in the switch while waiting for the header flit to route to the next stages. Wormhole switching not only reduces the store-and-forward delay at each switch, but it also requires much less buffer spaces. One packet may occupy several intermediate switches at the same time. Because of its low latency and low storage requirement advantages, wormhole is an ideal switching technique for on-chip multiprocessor interconnect networks. A number of recently proposed network-on-chip prototype implementations are indeed based on wormhole packet switching [5, 21, 22, 29].

Different switching approaches trade off better average delivery time and channel utilization for increased variance, and decreased predictability. The impact of switching on energy efficiency has not been explored in detail. Depending on the application domain, non-determinism can be more or less tolerable. In many cases, hybrid or adaptive switching approaches can achieve an optimal balance between average performance measures and predictability and energy.

Switching is tightly coupled to routing. Routing algorithms establish the path followed by a message through the network to its final destination. The classification, evaluation and comparison of on-chip routing schemes [25] involves the analysis of several trade-offs, such as predictability versus average performance, router complexity and speed versus achievable channel utilization, and robustness versus aggressiveness. A coarse distinction can be made between *deterministic* and *adaptive* routing algorithms. Deterministic approaches always supply the same path between a given source-destination pair, and they are the best choice for uniform or regular traffic patterns. In contrast, adaptive approaches use information on network traffic and channel conditions to avoid congested regions of the network. They are preferable in presence of irregular traffic or in networks with unreliable nodes and links. Among other routing schemes, probabilistic broadcast algorithms [26] have been proposed for *Networks on Chip* (NoCs).

Optimal routing for on-chip micro-networks is another open and important research theme. We predict that future approaches will emphasize speed and decentralization of routing decisions. Robustness and fault tolerance will also be highly desirable. These factors, and the observation that traffic patterns for special-purpose SoCs tend to be irregular, seem to favor adaptive routing. However, when traffic predictability is high and non-determinism is undesirable, deterministic routing may be the best choice.

Example 10 *The SPIN micro-network adopts wormhole routing [5]. Routing decisions are set by the network architecture, the fat-tree. Packets from a node (a tree leaf) are routed toward the tree root until they reach a switch that is a common ancestor with the destination node. At that point, the packet is routed toward the destination following the unique path between ancestor and destination node.*

Reconfigurable networks, such as today's FPGAs, or the heterogeneous interconnects of many reconfigurable computing platforms, adopt a deterministic and static routing approach. Routing (i.e., switch programming) is performed at reconfiguration time, and routing decisions are taken on a much coarser time scale than in packet-switched networks. From a networking viewpoint, switch programming can be viewed as setting up a number circuit-switched connections that last a very long time.

Routing signals in the RAW [2] machine is under software control. The compiler routes the signals along the optimal pathways by precisely scheduling the signals to meet the demand of the applications. Thus routing is adaptive, and connections in RAW are referred to as "soft wires".

The routing scheme in the Aethereal micro-network has two modes of operation: it supports both best effort services and guaranteed throughput. Thus links can be configured according to the type of application that is running and its requirements between two end-nodes. Best-effort packets use wormhole source routing, while the guaranteed throughput mode uses time-division multiplexed circuit switching [29].

Some other routing schemes have found direct applications in NoCs. *Hot potato* or *deflection* routing is based on the idea of delivering a packet to an output channel of a switch at each cycle. It requires a switch with an equal

number of input and output channels. Therefore, input packets can always find at least one output exit and no *deadlock* occurs. However, *livelock* is a potential problem in hot potato routing. Proper deflection rules need to be defined to avoid livelock problems.

Example 11 *A contention-aware hot-potato routing scheme was proposed by Nilsson. [47]. It is based on a two-dimensional mesh NoCs and on the switch architecture described by Kumar [35]. Each switch node also serves as network interface to a node processor (also called resource). Therefore, it has five inputs and five outputs. Each input has a buffer that can contain one packet. One input and one output are used for connecting the node processor. An internal FIFO is used to store the packets when output channels are all occupied. The routing decision at every node is based on the "stress values", which indicate the traffic loads of the neighbors. The stress value can be calculated based on the number of packets coming into the neighboring nodes at a unit time, or based on the running average of the number of packets coming to the neighbors over a period of time. The stress values are propagated between neighboring nodes. This scheme is effective in avoiding "hot spots" in the network. The routing decision steers the packets to less congested nodes.*

Ye [63] perfected this scheme by using a wormhole-based contention-look-ahead routing algorithm that can "foresee" the contention and delays in the coming stages using a direct connection from the neighboring nodes. It is also based on a mesh network topology. The major difference from [47] is that information is handled in flits, and thus large and/or variable size packets can be handled with limited input buffers. Therefore, this scheme combines the advantages of wormhole switching and hot potato routing.

The choice of routing algorithms has an impact on the overall system energy efficiency [26, 63], and that there is a trade off between energy efficiency and performance. Consider, for instance, packet flooding schemes, sometimes adopted for adaptively routing information though the fastest route between two nodes. This approach can be beneficial for performance, but it is very energy-inefficient.

3.2.3 Transport layer

On top of the network layer, the transport layer decomposes messages into packets at the source. It also resequences and reassembles them at the destination. Packetization granularity is a critical design decision, because the behavior of most network control algorithms is very sensitive to packet size. In most macroscopic networks, packets are standardized to facilitate internetworking, extensibility and compatibility of networking hardware produced by different manufacturers. Packet standardization constraints can be relaxed in SoC micro-networks, which can be customized at design time.

Example 12 *The size of the packets has a direct impact on both performance and energy consumption. Thus, an interesting problem is the search for optimal packet size. To some extent, optimal packetization depends on the network architecture and on the system function (including application software). Ye [62] performed a set of experiments on mesh networks of computing nodes, each node including processing and local cache memory. The experiments consisted of simulating the execution of benchmark programs using the RSIM simulator. The qualitative meaning of these experiments can be summarized as follows.*

Larger packets correlate to lower cache miss rates but to higher miss penalties, due to the increase of packetization, memory access and contention delays. The cache miss rate decreases and then stabilizes with the increase of packet size, while the miss penalty increases overlinearly. Therefore, there exists an optimal packet size to achieve best performance.

The energy spent on the interconnect network increases as the packet size increases, because the packet energy per hop increases as well as the number of hops. On the other hand, storage energy decreases as packets get larger, because of several factors including lower cache miss rates. Overall, the energy consumption has a minimum value for some packet length, such a minimum depending on the relative importance of storage and interconnect energy consumption. Note that the optimal packet length for performance may differ from the optimal packet length from an energy standpoint. (See Figure 5.)

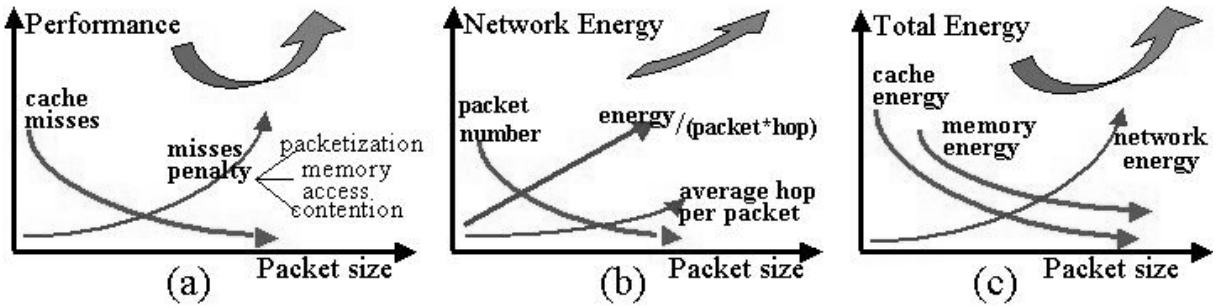

Figure 5: Latency and energy trends as packet size varies

Other functions of the transport layer are to control the flow of data into the network, to allocate network resources and to negotiate quality of service. Data flow can be controlled through *admission control*, commonly used in circuit-switched networks, that negate access to the network when there is no available path from source to destination. In packet-switched networks, access can be more finely tuned through *congestion control* that slows down forwarding of data packets to prevent congestion downstream. Alternatively, *traffic shaping* regulates the rate of entrance of data packets directly at the source. In packet-switched networks with predictable packet flow (for instance, in *virtual-circuit* networks where all packets from a source to a destination follow the same route), it is possible to negotiate guarantees on network performance (throughput, latency) upon connection setup.

In general, flow control and negotiation can be based on either deterministic or statistical procedures. Deterministic approaches ensure that traffic meets specifications, and provide hard bounds on delays or message losses. The main disadvantage of deterministic techniques is that they are based on worst cases, and they generally lead to significant under-utilization of network resources. Statistical techniques are more efficient in terms of utilization, but they cannot provide worst case guarantees. Similarly, from an energy viewpoint, we expect deterministic schemes to be more expensive than statistical schemes.

Example 13 *The SONICS Silicon Backplane Micro-network [57, 58] supports a time-division multiple access (TDMA) protocol. When a node wants to communicate, it needs to issue a request to the arbiter during a time slot and it may be granted access in the following time slot, if arbitration is favorable. Hence, arbitration introduces a non-deterministic waiting time in transmission. To reduce non-determinism, the micro-network protocol provides a form of slot reservation: nodes can reserve a fraction of the available time slots, and therefore they allocate bus bandwidth in a deterministic fashion. Reserved slots are assigned on a periodic pattern (a timing wheel), hence a master is guaranteed to have access to all its reserved slots within the revolution period of the timing wheel.*

The main shortcoming of the deterministic scheme is that the revolution period of the timing wheel is a large multiple of the bus clock period, hence long latencies can be experienced for bus access if the bus master requests are misaligned in time w.r.t. its reserved slots. A randomized TDMA scheme, called "Lotterybus" [36] has been proposed to achieve lower average bus access latency, at the price of the loss of worst-case latency guarantees.

Summarizing the overview of network architectures and control issues we can conclude that the theoretical framework developed for large-scale (local, wide area) and on-chip interconnection networks provides a convenient environment for reasoning about on-chip micro-networks as well. It is important to notice that the design space of micro-networks is currently very scarcely explored, and a significant amount of work is needed to predict the tradeoff curves in this space. We also believe that there is significant room for innovation: on-chip micro-networks architectures and protocols can be specialized for a specific system configurations and classes of applications. Furthermore, as pointed out several times above, the impact of network design and control decisions on communication energy is an important research theme that will become critical as communication energy scales up in SoC architectures.

4 Software layers

The hardware infrastructure described in the previous sections provides a basic communication service to the network's end nodes. Most of the end nodes in future on-chip networks will be programmable, ranging from general-purpose microprocessors, to application-specific processors, to reconfigurable logic. Other nodes, such as I/O blocks and memories will serve as slaves for the processor nodes, but they will also support some degree of reconfigurability. At the application level, programmers need a programming model and software services to effectively exploit the computational power and the flexibility of these highly parallel heterogeneous architectures. This section focuses on the system software and programming environment providing the hardware abstraction layer at the interface between the NoC architecture and the application programmer. First, we will discuss the programming models, then we will overview middleware architecture, and finally we will describe the development support infrastructure.

4.1 Programming Model

Programming models provide an abstract view of the hardware to application programmers. Abstraction is key for two main reasons: (i) it hides hardware complexity, (ii) it enhances code longevity and portability. Complex application programming on a large scale would be impossible without good programming models. On the other hand, the abstract view of the hardware enforced by programming model implies an efficiency loss in exploiting advanced hardware features. Hence, striking the best balance between abstraction and direct hardware control is an extremely critical issue. In traditional parallel computing, the two most common and successful programming models are *shared memory* and *message passing* [19, 6]. In the shared memory model communication is implicitly performed when parallel tasks access shared locations in a shared address space. In the message passing models, tasks have separate address spaces and inter-task communication is carried out via explicit messages (*send and receive* primitives).

Writing parallel code, or parallelizing existing sequential programs in the shared memory model, is generally easier than writing message passing code. However, shared memory requires significant hardware support to achieve high performance (e.g., snoopy caches), and message passing code, albeit harder to write, can achieve high performance even in architectures that implement very simple communication channels. From the scalability viewpoint, the shared memory models becomes less viable when communication latency is high or highly non-uniform (depending, for instance from the source and destination), thus, message passing is the best solution for large-scale and highly distributed parallel computers [19, 6].

In our view, message passing is the programming model of choice for NoC application software. Our position is motivated by several reasons. First, software for application-specific systems on chip is developed starting from specification languages that emphasize explicit communication between parallel executing tasks. For instance, complex signal processing applications are often developed using data-flow models, where data flows from one processing kernel to the other [40]. Second, embedded code development is traditionally focused on achieving high performance on limited hardware resources, and message passing can achieve higher performance at the price of increased development effort. Third, by making communication explicit, message passing pinpoints the main sources of unpredictability in program execution, namely communication latency and throughput. Hence, message passing programs are generally more predictable [43], a very desirable characteristic for embedded applications. Finally, since the main motivation for moving to NoC architectures is to provide better architectural scalability even in face of increasing communication latencies, it is natural to adopt a programming style that fully supports large-scale parallelism and makes it easier to focus on communication latency starting from the top abstraction layer in the design hierarchy.

The message passing model is supported by many formal frameworks (e.g., *communicating sequential processes*, CSP), languages (e.g., Occam) and application programming interfaces (e.g., the *message passing interface*, MPI) [6, 43]. Even though the theoretical importance of formal models and languages cannot be overemphasized, in the design practice we believe that traditional languages (such as C, C++ or Java) will maintain dominance thanks to the adoption of standardized and portable communication APIs. In the domain of multi-processor embedded signal processing systems, MPI has been proposed as message passing support library [38]. A natural evolution

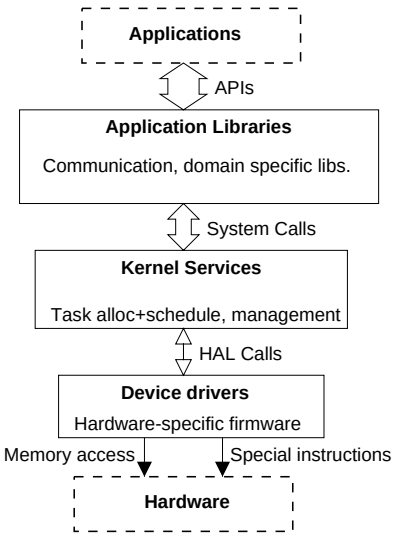

Figure 6: Middleware architecture

of traditional MPI for embedded applications is real-time MPI [34], an enhanced MPI standard which emphasizes run-time predictability. MPI appears to be an interesting candidate for NoC application level programming, however significant challenges are still open. First, MPI provides a number of messaging primitives, and many of them require quite complex system and software support, many MPI implementations are built on top of lower-level hardware-assisted messaging primitives, hence their performance may not be sufficient for the NoC environment. In our view, only a restricted subset of MPI functions are strictly required for NoC software development. In some cases, when performance constraints are extremely tight, lower-level messaging interfaces, such as *active messages* can be used [19]

4.2 Middleware Architecture

Programming models and their implementations (languages and APIs) are the front-end interface between application programmers and the target hardware platform. However, they are only the top layer of the complex software infrastructure that lies between hardware and applications. The main purpose of middleware is to provide safe and balanced access to hardware resources. First, tasks should be allocated and scheduled to the available NoC end nodes (processors). Similar allocation and scheduling decisions should be taken for transferring data from external memory to on-chip embedded memory blocks (and vice versa). Multiprocessor scheduling is a well developed discipline [27], but memory allocation and memory transfer scheduling for embedded memories is still under very active investigation [48]. An ancillary function in task management is task creation and destruction when task sets are dynamic.

Synchronization is another key middleware function. In single-processor systems, task synchronization is simplified because tasks are serialized on the processor, and therefore no true task parallelism is manifest in the system. In multiprocessor systems, there is obviously true task parallelism. Synchronization becomes much harder, and requires some form of hardware support (e.g. atomic transactions). Fortunately, this problem has been extensively studied for traditional multiprocessors and parallel computers, and a number of synchronization primitives, such as semaphores, critical sections, monitors have been proposed and thoroughly understood [6]. For NoC, synchronization primitives should be as lightweight as possible both from the performance and from the code size perspective.

Middleware is also in charge of managing (shared) peripherals, such as on-chip and off chip memories, I/O controllers, slave coprocessors, etc. Guaranteeing mutual exclusive access to peripherals is another facet of the synchronization function, but resource management also encompasses other tasks, such as power management

and reconfiguration management. We can expect that most peripherals in future NoCs will be reconfigurable and/or power manageable. Furthermore, even the communication network can be expected to be re-configurable at run time (for example through dynamic update of routing tables). Management is relatively straightforward in a single-processor environment, but it becomes a challenging task when control is distributed. In many parallel computing platform, peripheral management is performed by a centralized controller. Even though this choice simplifies the implementation of management policies, it creates a bottleneck in the system for scalability and robustness. For this reason distributed management policies are desirable, and they should be actively investigated in the future, possibly leveraging results from large-scale distributed environment (e.g. wireless networks).

Finally, the NoC middleware includes low level hardware control functions that abstract as much as possible platform-dependent details and build a hardware abstraction layer, with a small number of standard access functions. This should be the only part that must be modified when porting the middleware to a new NoC architecture. The middleware architecture and its layers are shown in Figure 6. Notice that the number of interface functions between layers grows as we get closer to application code.

In principle, most modern *operating systems* (OSs), such as Linux or Unix, support all above mentioned functions. Unfortunately, these OSs are too complex, time and memory consuming to be acceptable in an embedded software environment. As a consequence, we should employ lightweight, customizable middleware services instead of a full-blown OS. Several embedded OSs have have been designed in a modular fashion, and they can be tailored to the needs of applications. Most commercial embedded OSs are single-processor even though some natively multi-processor OSs are available both commercially and in the open source domain [52, 55]. However, the performance of their communication primitives has not been validated yet for NoC platforms. In this area, an intersting research topic is the customization of OS features and automatic or semi-automatic retargeting of middleware to different platforms [28].

In our view, there are two main challenges in the development of the new generation of NoC middleware. First, finding the sweet spot balancing hardware and software support for communication services. Second, providing a flexible mix of guaranteed and best-effort services. These two challenges are tightly interrelated, but not completely overlapping. Latency and throughput requirements for NoCs are extremely tight (remember that an advanced NoC must successfully compete against high-bandwidth on-chip busses), thus, high-performance hardware-assisted schemes, minimizing slow software-based network transaction management, seem to be preferable. On the other hand, most general hardware-assisted communication schemes require dedicated communication processors, which imply a non-negligible cost overhead [19]. Alternatively, user-level communication access schemes can eliminate part of the overhead associated with middleware-assisted network transactions, at the price of significant losses in software robustness and error containment capability. One possible compromise solution is to design communication-oriented, lightweight kernels that provide limited services, but high performance on critical communication functions. This approach has been taken in the Virtuoso real-time Kernel [59], which represents the state of the art in NoC oriented operating systems. Virtuoso has been specifically designed for performance-constrained multiprocessor signal-processing systems and it supports a fully distributed application development style, where tasks live in separate address spaces and communicate via simple message-passing (CSP-like) through explicitly instantiated channels. As opposed to traditional operating systems, communication is handled with the same priority and reduced overhead used for handling interrupts.

Even though a communication-centric approach in middleware design can be effective in reducing the overhead in supporting communication services, additional care should be taken in ensuring guaranteed-quality services, where "good-on-average" performance (i.e., *best effort* services) is not enough. Typical examples of applications requiring performance guarantees are streaming audio and video processing, and many signal processing tasks. Guaranteed quality of service (QoS) support requires careful consideration across all level of NoC design hierarchy, including application and middleware. Assuming that the NoC hardware backbone can support some form of QoS (e.g. upper bounds on message delivery times), the middleware framework is required to provide similar guarantees. One interesting way to tackle this challenge is to support quality of service negotiation followed by prioritized messaging starting from the application level [59]. In this paradigm a hard-real-time task initially negotiates a QoS with the NoS interface, then it is assigned a level of priority for both CPU and network access. Lower-priority packets waiting for network interface access can be preempted by high priority packets (coming from a high-priority task), thereby preventing priority inversion on the NoC interface.

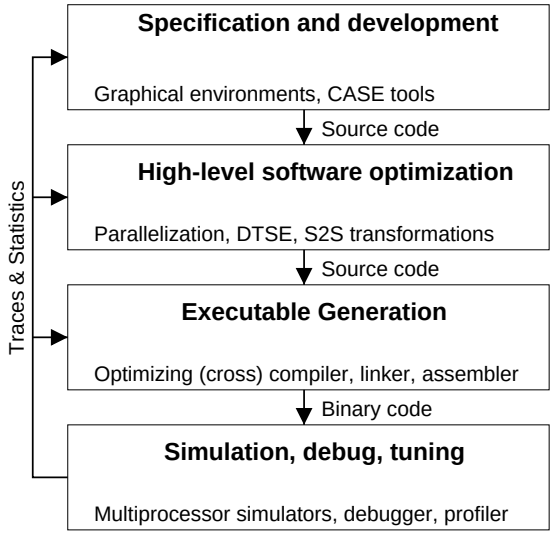

Figure 7: Software development infrastructure

4.3 Software Development Tools

The tools employed by programmers to develop and tune application software and middleware are a critical part of the NoC design infrastructure. Even though a detailed survey of this area is beyond the scope of the chapter, we will briefly overview the main building blocks of the software development infrastructure, as depicted in Figure 7. Specification and development tools have the main purpose of facilitating code development and increasing the programmer's productivity. In this area, we can group interactive, graphical programming environments, code versioning systems, code documentation systems and general computer-aided software engineering frameworks. Existing CASE systems can be adapted to support NoC programming. An opportunity for innovation in this area is the integration of software development and target architecture models to guide the mapping of software to available computation and communication resources [39].

High-level software optimization tools take as input the source code generated by CASE systems and produce optimized source code. In this area we can group numerous tools for embedded software optimization, such as parallelizing compilers [45] and interactive code transformation methodologies, like, for instance, the *data-transfer* and *storage exploration* (DTSE) proposed by IMEC [16]. In a NoC environment these tools should focus on removing or reducing communication bottlenecks, either by eliminating redundant data transfers, or by hiding latency and maximizing available bandwidth. Most of the transformations performed at this stage should be weakly dependent on specific hardware targets.

Target-specific optimizations are confined to the third software development stage, namely, executable generation. The back-end of the optimizing compiler should also link application object files with middleware libraries [37]. At this stage, inter-task communication and synchronization points cannot be removed or transformed, and optimization opportunities are confined within the task code. Instruction-level parallelism can be discovered and exploited at this stage. High level optimization and low-level optimizations during code generation could be seen as two separate stages in a complex software optimizer with architecture neutral frontend and architecture specific backend. Concerning NoC-specific optimizations, high-level communication optimization is performed in the frontend, while fine tuning of communication primitives and function calls to architectural features can be performed in the backend.

The last class of tools includes system simulation, profiling and code analysis/debugging. These tools are critical for functional debugging, as well as performance tuning. Several challenges must be addressed at this level. First, system simulation for NoCs is extremely challenging from the computational viewpoint. Hence, several simulation engines should be available, spanning the accuracy versus speed tradeoff. Modern hardware-software

specification languages and simulators, such as SystemC [30], can support hardware descriptions at multiple abstraction levels. NoC simulation platforms have recently been presented [8], and this area of research and development is currently very active. Profilers play a strategic role in functional debugging, code tuning and feedback directed code optimization, where profiling information is fed back as inputs to the optimization tools [4].

5 Conclusions

This chapter considers the challenges of designing SoCs with tens or hundreds of processing elements in $[100 - 25]nm$ silicon technologies. Challenges include dealing with design complexity and with providing reliable, high-performance operation with small energy consumption.

We claim that modular, component-based design of both hardware and software is needed to design complex SoCs. Starting from the observation that interconnect technology will be the limiting factor for achieving the operational goals, we develop a communication-centric view of design. We postulate that efficient communication on SoCs can be achieved by reconfigurable micro-networks, whose layered design can exploit methods and tools used for general networks. At the same time, micro-networks can be specialized and optimized for the specific SoC being designed.

We examined the different layers in micro-network design, and we outlined the corresponding research problems. Despite the numerous challenges, we remain optimistic that such problems can find adequate solutions. At the same time, we believe that a layered micro-network design methodology is likely to be the only path to master the complexity of SoC designs in the years to come.

6 Acknowledgments

This chapter was written with support from the MARCO/ARPA GSRC.

7 Glossary

- **CDMA** Code division multiple access. Spread spectrum communication technique achieved by multiplying data by a pseudo-random sequence before transmission on a channel and after reception.
- **Data Link** Abstraction of communication channel control. It provides a reliable link over an unreliable physical channel.
- **DPM** Dynamic power management. Method to reduce energy consumption by on-line control of frequency and/or supply voltage of components, including shut off.
- **DVS** Dynamic voltage scaling. Method to reduce energy consumption by on-line tuning the power supply voltage.
- **EMI** Electromagnetic interference.
- **FPGA** Field programmable gate array. Circuit whose function can be programmed on the field, by controlling transistor (or antifuse) switches with a bit map.
- **GALS** Globally asynchronous locally synchronous circuits.
- **LFSR** Linear feedback shift register. Shift register with feedback through EXOR gates that produces a pseudorandom sequence of vector. Used for built-in self-test.

- **MAC** Medium access control. Abstraction of communication channel control that deals with the interfaces to physical channel. Part of data link layer.
- **MTTF** Mean time to failure. Expected failure time for a given reliability distribution.
- **Network Routing** Establishes the path followed by messages. delivery control.
- **Network Switching** Abstraction of communication channel control. Implements the end-to-end delivery control.
- **Protocol Stack** Abstraction of communication control in seven layers.
- **QoS** Quality of service.
- **Reliability** Probability that a system (or component) fails as function of time.

References

[1] B. Ackland et al., "A Single Chip, 1.6-Billion, 16-b MAC/s Multiprocessor DSP," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 3, March 2000.

[2] A. Agrawal, "Raw Computation," *Scientific American*, August 1999.

[3] P. Aldworth, "System-on-a-Chip Bus Architecture for Embedded Applications," *IEEE International Conference on Computer Design*, pp. 297-298, 1999.

[4] E. Altman, K. Ebcioglu, M. Gachwind, S. Sathaye, "Advances and Future Challenges in Binary Translation and Optimization," *Proceedings of the IEEE*, vol. 89, no. 11, pp. 1710-1722, Nov. 2001.

[5] A.Adriahantenana and A Greiner, "Micro-network for SoC: Implementation of a 32-bit SPIN Network," *Design Automation and Test in Europe Conference*, pp. 1129-1129, 2003.

[6] G. Andrews, *Foundations of Multithreaded, Parallel and Distributed Programming*, Addison-Wesley, 2000.

[7] H. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, Addison-Wesley, 1990.

[8] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi, M. Poncino, "SystemC Co-Simulation of Multi-Processor Systems-on-Chip," *IEEE International Conference on Computer Design*, pp. 494-499, 2002.

[9] L. Benini, A. Bogliolo, G. De Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," *IEEE Transactions on Very Large-Scale Integration Systems*, vol. 8, no. 3, pp. 299-316, June 2000.

[10] L. Benini, G. De Micheli, E. Macii, M. Poncino, S. Quer, "Power Optimization of Core-based Systems by Address Bus Encoding," *IEEE Transactions on Very Large-Scale Integration Systems*, vol. 6, no. 4, pp. 578-58 Dec. 1998.

[11] L. Benini, G. De Micheli, "System-Level Power Optimization: Techniques and Tools," *ACM Transactions on Design Automation of Electronic Systems*, vol. 5, no. 2, pp. 115-192, April 2000.

[12] L. Benini and G. De Micheli, "Networks on Chips: A New SoC Paradigm," *IEEE Computers*, January 2002, pp. 70-78.

[13] L. Benini and G. De Micheli, "Powering Networks on Chips: Energy-efficient and Reliable Interconnect Design for SoCs," *ISSS, Proceedings of the International Symposium on System Synthesis*, Montreal, October 2001, pp. 33-38.

[14] D. Bertozzi, L. Benini and G. De Micheli, "Low-Power Error-Resilient Encoding for On-chip Data Busses," *DATE, International Conference on Design and Test Europe* Paris, 2002, pp. 102-109.

[15] D. Bertsekas, R. Gallager, *Data Networks*. Prentice Hall, 1991.

[16] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, A. Vandecappelle, *Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design*, Kluwer, 1998.

[17] N. Cohen, T. Sriram, N. Leland, D. Moyer, S. Butler and R. Flatley, "Soft Error Considerations for Deep-Submicron CMOS Circuit Applications," *IEDM, Proceedings of IEEE International Electron Device Meeting*, pp. 315-318, 1999.

[18] B. Cordan, "An Efficient Bus Architecture for System-on-chip Design," *IEEE Custom Integrated Circuits Conference*, pp. 623-626, 1999.

[19] D. Culler, J. Pal Singh, A. Gupta, *Parallel Computer Architecture: a Hardware/Software Approach*. Morgan-Kaufman, 1999.

[20] W. Dally and J. Poulton, *Digital Systems Engineering*, Cambridge University Press, 1998.

[21] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, L. Benini, "Xpipes: a Latency Insensitive Parameterized Network-on-Chip Architecture for Multi-Processor SoCs", *International Conference on Computer Design* pp.536-539, 2003.

[22] W. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks" *DAC, Proceedings of the 38th Design Automation Conference* pp.684-689.

[23] A. Deutsch, "Electrical Characteristics of Interconnections for High-Performance Systems," *Proceedings of the IEEE*, vol. 86, no. 2, pp. 315-355, February 1998.

[24] D. Ditzel, "Transmeta's Crusoe: Cool Chips for Mobile Computing", *Hot Chips Symposium*, Stanford, 2000.

[25] J. Duato, S. Yalamanchili, L. Ni, *Interconnection Networks: an Engineering Approach*. Morgan Kaufmann, 2003.

[26] T. Dumitra, S. Kerner and R. Marculescu, "Towards On-chip Fault-Tolerant Communication," *ASPDAC - Proceedings of the Asian-South Pacific Design Automation Conference*, pp. 225-232, 2003.

[27] H. El-Rewini, H. Ali, T. Lewis, *Task Scheduling in Parallel and Distributed Systems* Prentice-Hall, 1994.

[28] L. Gauthier, S. Yoo, A. Jerraya, "Automatic Generation and Targeting of Application-specific Operating Systems and Embedded Systems Software," *IEEE Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 11, pp. 1293-1301, Nov. 2001.

[29] K. Goossens, J. van Meerbergen, A. Peeters and P. Wielage, "Networks on Silicon: Combining Best Efforts and Guranteed Services," *Design Automation and Test in Europe Conference*, pp. 423-427, 2002.

[30] T. Groetker, S. Liao, G. Martin, S. Swan, *System Design with SystemC*, Kluwer, 2002.

[31] P. Guerrier, A. Grenier, "A Generic Architecture for On-chip Packet-switched Interconnections," *Design Automation and Test in Europe Conference*, pp. 250-256, 2000.

[32] R. Hegde, N. Shanbhag, "Toward Achieving Energy Efficiency in Presence of Deep Submicron Noise," *IEEE Transactions on VLSI Systems*, pp. 379-391, vol. 8, no. 4, August 2000.

[33] R. Ho, K. Mai, M. Horowitz, "The Future of Wires," *Proceedings of the IEEE*, January 2001.

[34] A. Kanevsky, A. Skjellum, A. Rounbehler, "MPI/RT - an Emerging Standard for High-performance Real-time Systems," *IEEE International Conference on System Sciences*, vol. 3, pp. 157-166, 1998.

[35] S. Kumar, A. Jantsch, J. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrj and A. Hemani, "A network on chip architecture and design methodology", *Proceedings of IEEE Computer Society Annual Symposium on VLSI*, April 2002, pp. 105-112.

[36] K.Lahiri, A.Raghunathan, G.Lakshminarayana, "LOTTERYBUS: A New High-Performance Communication Architecture for System-on-Chip Designs", *Design Automation Conference*, pp.15-20, 2001.

[37] R. Janka, R. Judd, J. Lebak, M. Richards, D. Campbell, "VSIPL: an Object-based Open Standard API for Vector, Signal and Image Processing," *IEEE Conference on Acoustic, Speech and Signal Processing*, pp. 949-952, vol. 2, 2001.

[38] R. Janka, L. Willis, L. Baumstark, "Virtual Benchmarking and Model Continuity in Prototyping Embedded Multiprocessor Signal Processing Systems," *IEEE Transaction onf Software Engineering*, vol. 28, no. 9, pp. 836-846, Sept. 2002.

[39] L. Lavagno, S. Dey, R. Gupta, "Specification, Modeling and Design Tools for System-on-chip," *Asia-Pacific Design Automation Conference*, pp. 21-23, Jan. 2002.

[40] E. Lee, A. Sangiovanni-Vincentelli, "A Framework for Comparing Models of Computation," *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, vol. 17, no. 12, pp. 1217-1229, Dec. 1998.

[41] S.-Y. Lee, S.-J. Song, K. Lee, J.-H. Woo, S.-E. Kim, B.-G. Nam, H.-J. Yoo, "An 800MHz star-connected on-chip network for application to systems on a chip," *IEEE Solid-State Circuits Conference*, pp. 468-469, 2003.

[42] C. Leiserson, "Fat-trees: Universal Networks for Hardware-efficient Supercomputing," *IEEE Transactions on Computers*, vol. 34, no. 10, pp. 892-901, October 1985.

[43] C. Leopold, *Parallel and Distributed Computing: a Survey of Models, Paradigms and Approaches*, Wiley-Interscience, 2001.

[44] J. Montanaro et al., "A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 11, pp. 1703-1714, Nov. 1996.

[45] S. Muchnick, *Advanced Compiler Design and Implementation*. Morgan & Kaufman, 1997.

[46] M. Nicolaidis, "Time Redundancy Based Soft-Error Tolerance to Rescue Nanometer Technologies," *Proceedings VTS*, 1999.

[47] E. Nilsson; M. Millberg, J. Oberg, A. Jantsch, "Load Distribution with the Proximity Congestion Awareness in a Networks on Chip", *Proceedings of Design Automation and Test in Europe*, March 2003, pp. 1126-1127.

[48] R. Panda, N. Dutt, A. Nicolau, F. Catthoor, A. Vandercappelle, E. Brockmeyer, C. Kulkarni, E. De Greef, "Data Memory Organization and Optimization in Application-specific Systems," *IEEE Design & Test of Computers*, vol. 18, no. 3, pp. 56-58, May-June 2002.

[49] C. Patel, S. Chai, S. Yalamanchili, D. Shimmel, "Power Constrained Design of Multiprocessor Interconnection Networks," *IEEE International Conference on Computer Design*, pp. 408-416, 1997.

[50] B. Prince *Report on Cosmic Radiation Induced SER in SRAMs and DRAMs in Electronic Systems*, May 2000.

[51] W. Remaklus, "On-chip Bus Structure for Custom Core Logic Design," *IEEE Wescon*, pp. 7-14, 1998.

[52] RTEMS Embedded Real-Time Open-source Operating System. *http://www.rtems.com.*

[53] D.Sylvester and K.Keutzer, "A Global Wiring Paradigm for Deep Submicron Design," *IEEE Transactions on CAD/ICAS*, Vol.19, No. 2, pp. 242-252, February 2000.

[54] T. Theis, "The future of Interconnection Technology," *IBM Journal of Research and Development*, Vol. 44, No. 3, May 2000, pp. 379-390.

[55] VSPWorks. Small Footprint Kernel Optimized for DSP. *http://www.windriver.com.*

[56] J. Walrand, P. Varaiya, *High-Performance Communication Networks.* Morgan Kaufman, 2000.

[57] W. Weber, "CPU Performance Comparison: Standard Computer Bus Versus SiliconBacplane," *www.sonics.com*, 2000.

[58] D. Wingard, "MicroNetwork-based integration for SOCs," *Design Automation Conference*, pp. 673-677, 2001.

[59] E. Verhulst, "The Rationale for Distributed Semantics as a Topology Independent System Design Methodology and its Implementation in the Virtuoso RTOS," *Design Automation for Embedded Systems*, vol. 6, pp. 277-294, 2002.

[60] S. Winegarden, "A Bus Architecture Centric Configurable Processor System," *IEEE Custom Integrated Circuits Conference*, pp. 627-630, 1999.

[61] F. Worm, P. Ienne, P. Thiran, and G. De Micheli, "An Adaptive Low-power Transmission Scheme for On-chip Networks," *ISSS, Proceedings of IEEE Integrated System Synthesis Symposium*, 2002.

[62] T. Ye, L. Benini and G. De Micheli, " Packetized On-Chip Interconnect Communication Analysis," *DATE, International Conference on Design and Test Europe*, 2003, pp. 344-349.

[63] T. Ye, L. Benini and G. De Micheli, " Packetization and Routing Analysis of On-chip Multiprocessor Networks," *Journal of System Architecture*, 2004.

[64] R. Yoshimura, T. Koat, S. Hatanaka, T. Matsuoka, K. Taniguchi, "DS-CDMA Wired Bus with Simple Interconnection Topology for Parallel Processing System LSIs," *IEEE Solid-State Circuits Conference*, pp. 371-371, Jan. 2000.

[65] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, J. Rabaey, "A 1-V Heterogeneous Reconfigurable DSP IC for Wireless Baseband Digital Signal Processing," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 11, pp. 1697-1704, Nov. 2000.