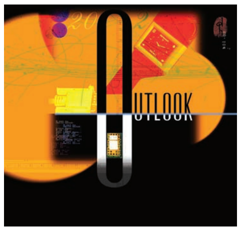
Networks on Chips: A New SoC Paradigm



On-chip micronetworks, designed with a layered methodology, will meet the distinctive challenges of providing functionally correct, reliable operation of interacting system-on-chip components.

Luca Benini
University of Bologna

Giovanni De Micheli
Stanford University

System-on-chip (SoC) designs provide integrated solutions to challenging design problems in the telecommunications, multimedia, and consumer electronics domains. Much of the progress in these fields hinges on the designers' ability to conceive complex electronic engines under strong time-to-market pressure. Success will rely on using appropriate design and process technologies, as well as on the ability to interconnect existing components—including processors, controllers, and memory arrays—reliably, in a plug-and-play fashion.

By the end of the decade, SoCs, using 50-nm transistors operating below one volt, will grow to 4 billion transistors running at 10 GHz, according to the International Technology Roadmap for Semiconductors. The major challenge designers of these systems must overcome will be to provide for functionally correct, reliable operation of the interacting components. On-chip physical interconnections will present a limiting factor for performance and, possibly, energy consumption.

Silicon technologies face other challenges. Synchronization of future chips with a single clock source and negligible skew will be extremely difficult, if not impossible. The most likely synchronization paradigm for future chips—globally asynchronous and locally synchronous—involves using many different clocks. In the absence of a single timing reference, SoC chips become distributed systems on a single silicon substrate. Global control of the information traffic is unlikely to succeed because the system needs to keep track of each component's states. Thus, components will initiate data transfers autonomously, according to their needs. The global communication pattern will be fully distributed, with little or no global coordination.

As SoC complexity scales, capturing the system's functionality with fully deterministic operation models will become increasingly difficult. As global wires span multiple clock domains, synchronization failures in communicating between different domains will be rare but unavoidable events.[1] Moreover, energy and device reliability concerns will impose small logic swings and power supplies, most likely less than one volt. Electrical noise due to crosstalk, electromagnetic interference, and radiation-induced charge injection will likely produce data errors, also called upsets. Thus, transmitting digital values on wires will be inherently unreliable and nondeterministic. Other causes of nondeterminism include design components with a high level of abstraction and coarse granularity and distributed communication control.

Focusing on using probabilistic metrics such as average values or variance to quantify design objectives such as performance and power will lead to a major change in design methodologies. Overall, SoC design will be based on both deterministic and stochastic models. Creating complex SoCs requires a modular, component-based approach to both hardware and software design.

Based on the premise that interconnect technology will be the limiting factor for achieving SoCs' operational goals, we postulate that the layered design of reconfigurable micronetworks, which exploits the methods and tools used for general networks, can best achieve efficient communication on SoCs.

0018-9162/02/$17.00 © 2002 IEEE

Wiring Delays

Projections for future silicon technologies show that chip size will scale up slightly while gate delays decrease compared to wiring delays. A simple computation shows that delays on wires that span the chip will extend longer than the clock period. This trend is a trivial consequence of the finite propagation speed of electromagnetic waves, which is $v = (0.3/\sqrt{\in})$ mm per second in a homogeneous medium with relative permittivity $\in$. In 50 nm technology, the projected chip die edge will be around 22 mm, with a clock frequency of 10 GHz.

Thus, the delay for a signal traversing the chip diagonally will be approximately 100 picoseconds, or one clock period, in the ideal case that $\in = 1$. A lower bound of two clock periods applies to general media with $\in > 1$.[1] Obviously, signal propagation on real-life interconnections is much slower than this lower bound, and optimistic predictions estimate propagation delays for highly optimized global wires—taking wire sizing and buffering into account—to be between six and 10 clock cycles for chips made using 50 nm technology.[2]

References

1. D. Sylvester and K. Keutzer, "A Global Wiring Paradigm for Deep Submicron Design," *IEEE Trans. CAD/ICAS*, Feb. 2000, pp. 242-252.
2. R. Ho, K. Mai, and M. Horowitz, "The Future of Wires," *Proc. the IEEE*, Apr. 2001, pp. 490-504.

NEW DESIGN APPROACH

Network engineers have already gained experience with using stochastic techniques and models for large-scale designs. We propose borrowing models, techniques, and tools from the network design field and applying them to SoC design.

We view a SoC as a micronetwork of components. The network is the abstraction of the communication among components and must satisfy quality-of-service requirements—such as reliability, performance, and energy bounds—under the limitation of intrinsically unreliable signal transmission and significant communication delays on wires. We propose using the micronetwork stack paradigm, an adaptation of the protocol stack shown in Figure 1,[2] to abstract the electrical, logic, and functional properties of the interconnection scheme.

SoCs differ from wide area networks in their local proximity and because they exhibit less nondeterminism. Local, high-performance networks—such as those developed for large-scale multiprocessors—have similar requirements and constraints. Some distinctive characteristics, such as energy constraints and design-time specialization, are unique to SoC networks, however.

Whereas computation and storage energy greatly benefit from device scaling, which provides smaller gates and memory cells, the energy for global communication does not scale down. On the contrary, as the "Wiring Delays" sidebar indicates, projections based on current delay optimization techniques for global wires[3] show that global on-chip communication will require increasingly higher energy consumption. Hence, minimizing the energy used for communications will be a growing concern in future technologies. Further, network traffic control and monitoring can help better manage the power that networked computational resources consume. For example, the clock speed and voltage of end nodes can vary according to available network bandwidth.

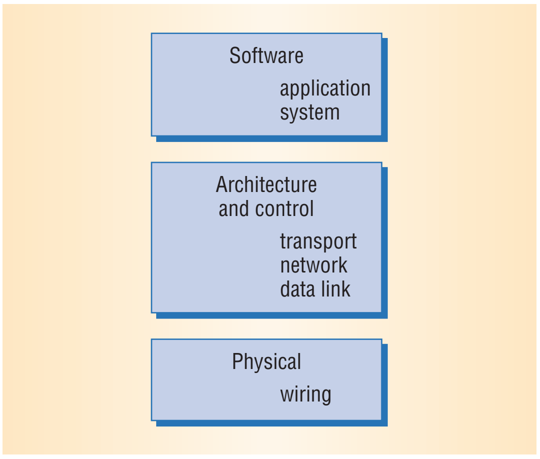


Figure 1. Protocol stack from which the micronetwork stack paradigm can be adapted. Bottom up, the layers span increasing design abstraction levels.

Another facet of the SoC network design problem, design-time specialization, raises many new challenges. Macroscopic networks emphasize general-purpose communication and modularity. Communication network design has traditionally been decoupled from specific end applications and is strongly influenced by standardization and compatibility constraints in legacy network infrastructures. In SoC networks, these constraints are less restrictive because developers design the communication network fabric on silicon from scratch. Thus, only the abstract network interface for the end nodes requires standardization. Developers can tailor the network architecture itself to the application, or class of applications, the SoC design targets.

We thus envision a vertical design flow in which every layer of the micronetwork stack is specialized and optimized for the target application domain. Such an application-specific on-chip network-synthesis paradigm represents an open and exciting research field. Specialization does not imply complete loss of flexibility, however. From a design standpoint, network reconfigurability will be key in providing plug-and-play component use because the components will interact with one another through reconfigurable protocols.

In addition to facilitating high-speed communication, reducing the voltage swing also has a beneficial effect on power dissipation.

ON-CHIP SIGNAL TRANSMISSION

Wires are the physical realization of communication channels in SoCs and, for our purposes, buses function as wire ensembles. Intensive research[3-5] into on-chip wiring has resulted in the commercial development of several physical design tools to support automated wiring. Nevertheless, coping with global wires that span significant distances, such as those beyond one millimeter, requires a paradigm shift.

Most likely, the reverse-scaled global wires will be routed on the top metal layers provided by the technology. Wiring pitch and width increase in higher wiring levels so that wires at top levels can be much wider and thicker than low-level wires.[5] Increased width reduces wire resistance, even considering the skin effect, while increased spacing around the wire prevents capacitance growth. At the same time, inductance effects increase relative to resistance and capacitance. As a result, future global wires will function as lossy transmission lines,[1] as opposed to today's lumped or distributed resistance-capacitance models.

In addition to facilitating high-speed communication, reducing the voltage swing also has a beneficial effect on power dissipation. Reduced-swing, current-mode transmission requires careful receiver design, with good adaptation to line impedance and high-sensitivity sensing, possibly with the help of sense amplifiers.

When using current technologies, most chip developers assume that electrical waveforms always carry correct on-chip information. Guaranteeing error-free information transfer at the physical level on global on-chip wires will become more difficult for several reasons.[6] Signal swings will be reduced and noise—due to crosstalk, electromagnetic interference, and other factors—will have increased impact. Thus, it will not be possible to abstract the physical layer of on-chip networks as a fully reliable, fixed-delay channel. At the micronetwork stack layers atop the physical layer, noise is a source of local transient malfunctions. An upset is the abstraction of such malfunctions. Upset probability can vary over different physical channels and over time.

In current designs, wiring-related effects are undesirable parasitics, and designers use specific, detailed physical techniques to reduce or cancel them. A well-balanced design should not try to achieve ideal wire behavior at the physical layer because the corresponding cost in performance, energy efficiency, and modularity may be too high. Physical-layer design should find a compromise between satisfying competing quality metrics and providing a clean and complete abstraction of channel characteristics for the micronetwork layers above.

MICRONETWORK ARCHITECTURE AND CONTROL

The architecture specifies the interconnection network's topology and physical organization, while the protocols specify how to use network resources during system operation. Whereas both micronetwork and general network design must meet performance requirements, the need to satisfy tight energy bounds differentiates on-chip network implementations.

Interconnection network architectures

On-chip networks relate closely to interconnection networks for high-performance parallel computers with multiple processors, in which each processor is an individual chip. Like multiprocessor interconnection networks, nodes are physically close to each other and have high link reliability. Further, developers have traditionally designed multiprocessor interconnections under stringent bandwidth and latency constraints to support effective parallelization.[7] Similar constraints will drive micronetwork design.

Shared-medium networks. Most current SoCs have a shared-medium architecture, which has the simplest interconnect structures. In this architecture, all communication devices share the transmission medium. Only one device can drive the network at a time. These networks support broadcast as well, an advantage for the highly asymmetric communication that occurs when information flows from few transmitters to many receivers. Within current technologies, the backplane bus is the most common example of an on-chip, shared-medium structure. This convenient, low-overhead interconnection handles a few active bus masters and many passive bus slaves that only respond to bus master requests.

We need bus arbitration mechanisms when several processors attempt to use the bus simultaneously. A bus arbiter module performs centralized arbitration in current on-chip buses. A processor seeking to communicate must first gain bus mastership from the arbiter. Because this process implies a control transaction and communication performance loss, arbitration should be as fast and 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 bandwidth consumption, developers have devised split transaction protocols for high-performance buses. In these protocols, the network releases bus mastership upon request completion, and the slave must gain access to the bus to respond, possibly several bus cycles later. Thus, the bus can support multiple outstanding transactions.

Obviously, bus masters and bus interfaces for split-transaction buses are more complex than those for simple atomic-transaction buses. For example, developers chose a 128-bit split-transaction bus for the Lucent Daytona chip,[8] a multiprocessor on a chip that contains four 64-bit processing elements that generate transactions of different sizes. To improve bus-bandwidth utilization and minimize the average latency caused by simultaneous requests, the bus partitions large transfers into smaller packets.

Although well understood and widely used, shared-medium architectures have seriously limited scalability. The bus-based organization remains convenient for current SoCs that integrate fewer than five processors and, rarely, more than 10 bus masters. Energy inefficiency is another critical limitation of shared-medium networks. In these architectures, every data transfer is broadcast, meaning the data must reach each possible receiver at great energy cost. Future integrated systems will contain tens to hundreds of units generating information that must be transferred. For such systems, a bus-based network would become a critical performance and power bottleneck.

Direct and indirect networks. The direct or point-to-point network overcomes the scalability problems of shared-medium networks. In this architecture, each node directly connects to a limited number of neighboring nodes. These on-chip computational units contain a network interface block, often called a router, that handles communication and directly connects to neighboring nodes' routers. Direct interconnect networks are popular for building large-scale systems because the total communication bandwidth also increases when the number of nodes in the system increases.

The Raw Architecture Workstation (RAW) architecture[9] is an example of a direct network implementation derived from 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 term RAW derives from the "raw" hardware's full exposure 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 determines exactly which set of wires connect at every cycle. Moreover, the compiler pipelines the long wires to support high clock frequency.

Indirect or switch-based networks offer an alternative to direct networks for scalable interconnection design. In these networks, a connection between nodes must go through a set of switches. The network adapter associated with each node connects to a switch's port. Switches themselves do not perform information processing—they only provide a programmable connection between their ports, setting up a communication path that can change over time.[7] Significantly, 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. As the "Virtex II FPGA" sidebar indicates, some field-programmable gate arrays are examples of indirect networks on chips.

Energy inefficiency is a critical limitation of shared-medium networks.

Hybrid networks. Introducing a controlled amount of nonuniformity in communication-network design provides several advantages. Multiple-backplane and hierarchical buses are two notable examples of the many heterogeneous or hybrid interconnection architectures that developers have proposed and implemented. These architectures cluster tightly coupled computational units with high communication bandwidth and provide lower bandwidth intercluster communication links. Because they use a fraction of the communication resources and energy to provide performance comparable with homogeneous, high-bandwidth architectures, energy efficiency is a strong driver toward using hybrid architectures.[10]

Micronetwork control

Using micronetwork architectures effectively requires relying on protocols—network control algorithms that are often distributed. Network control dynamically manages network resources during system operation, striving to provide the required quality of service. Following the micronetwork stack layout shown in Figure 1, we describe the three architecture-and-control layers—data link, network, and transport—from the bottom up.

Data link layer. The physical layer is an unreliable digital link in which the probability of bit upsets is non-null. Data-link protocols increase the reliability of the link, up to a minimum required level,

Virtex II FPGA

Most current field-programmable gate arrays 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 many interconnected computing elements. Current FPGA communication networks differ from future SoC micronetworks in granularity and homogeneity.

Processing elements in traditional FPGAs implement simple bit-level functional blocks. Thus, communication channels in FPGAs are functionally equivalent to wires that connect logic gates. Because future SoCs will house complex processing elements, interconnects will carry much coarser quantities of information. The different granularity of computational elements and communication requirements has far-reaching consequences for 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 greater degrees of freedom for optimizing communication as well.

The concept of dynamically reconfiguring FPGAs applies well to micronetwork design. SoCs benefit from programmability on the field to match, for example, environmental constraints. This programmability also lets runtime reconfiguration adapt, for example, to a varying workload. Reconfigurable micronetworks exploit programmable routers, switches, or both. Their embodiment may leverage multiplexers whose control signals are set—as with FPGAs—by configuration bits in local storage.

For example, Figure A shows the Xilinx Virtex II FPGA with various configurable elements to support reconfigurable digital-signal-processor design. The internal configurable rectangular array contains configurable logic blocks (CLBs), random access memories (RAMs), multipliers (MUL), switches (SWT), I/O buffers (IUB), and dynamic clock managers (DCM). Routing switches facilitate programmable interconnection. Each programmable element connects to a switch matrix, allowing multiple connections to the general routing matrix. Values stored in static memory cells control all programmable elements, including the routing resources. Thus, Virtex II exemplifies an indirect network over a heterogeneous fabric.

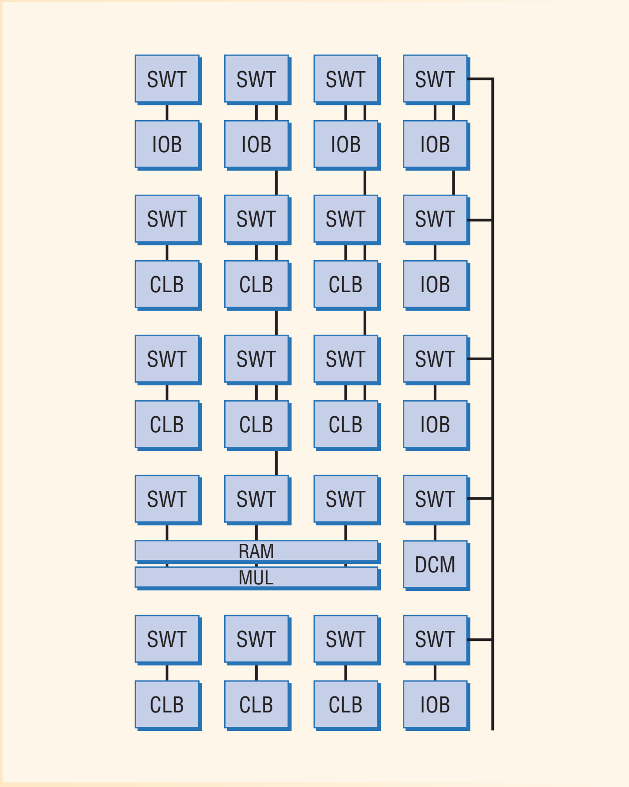


Figure A. Xilinx Virtex II, a field-programmable gate array architecture that exemplifies an indirect network over a heterogeneous fabric.

under the assumption that the physical layer by itself is not sufficiently reliable.

In a shared-medium network, contention creates an additional error source. Contention resolution, fundamentally a nondeterministic process, is an additional noise source because it requires synchronization of a distributed system. In general, synchronization can virtually eliminate nondeterminism at the price of some performance loss. For example, centralized bus arbitration eliminates contention-induced errors in a synchronous bus but the slow bus clock and bus request-and-release cycles impose a substantial performance penalty.

Packetizing data deals effectively with communication errors. Sending data on an unreliable channel in packets makes error containment and recovery easier because the packet boundaries contain the effect of errors and allow error recovery on a packet-by-packet basis. Using error-correcting codes that add redundancy to the transferred information can achieve error correction at the data link layer. Packet-based error-detection and -recovery protocols that have been developed for traditional networks, such as alternating-bit, go-back-*N*, and selective repeat, can complement error correction.[2] Several parameters in these protocols, such as packet size and number of outstanding packets, can be adjusted to achieve maximum performance at a specified residual error probability, within given energy consumption bounds, or both.

Network layer. This layer implements end-to-end delivery control in network architectures with many communication channels. In most current on-chip networks, all processing elements connect to the same channel: the on-chip bus, leaving the network layer empty. However, when a collection of links connects the processing elements, we must decide how to set up connections between successive links and route information from its source to the final destination. Developers have studied these *switching* and *routing* tasks extensively in the context of both multiprocessor interconnects[7] and general communication networks.[2]

Switching algorithms can be grouped into three classes: circuit, packet, and cut-through switching.[7] These approaches trade off better average delivery time and channel utilization for increased variance and decreased predictability. The low latency of cut-through switching schemes will likely make them preferable for on-chip micronetworks from a performance standpoint. However, aggressive forwarding of data through switches can increase traffic and contention, which may waste energy. Depending on the application domain, nondeterminism can be more or less tolerable.

Switching is tightly coupled to routing. Routing algorithms establish the path a message follows through the network to its final destination. Classifying, evaluating, and comparing on-chip routing schemes[7] requires analyzing several trade-offs, such as

- predictability versus average performance,
- router complexity and speed versus achievable channel utilization, and
- robustness versus aggressiveness.

We can make a coarse distinction between deterministic and adaptive routing algorithms. Deterministic approaches always supply the same path between a given source-destination pair and offer the best choice for uniform or regular traffic patterns. In contrast, adaptive approaches use information about network traffic and channel conditions to avoid congested network regions. An adaptive approach is preferable when dealing with irregular traffic or in networks with unreliable nodes and links.

We conjecture that future on-chip micronetwork designs 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 nondeterminism is undesirable, deterministic routing may be the best choice. The "SPIN Micronetwork" sidebar describes a micronetwork that uses deterministic routing.[11]

Transport layer. Atop the network layer, the transport layer decomposes messages into packets at the source. It also resequences and reassembles the messages at the destination. Packetization granularity presents a critical design decision because most network-control algorithms are highly sensitive to packet size. Most macroscopic networks standardize packets to facilitate internetworking, extensibility, and the compatibility of the networking hardware that different manufacturers produce. Packet standardization constraints can be relaxed in SoC micronetworks, which can be customized at design time.

In general, either deterministic or statistical procedures can provide the basis for flow control and negotiation. Deterministic approaches ensure that traffic meets specifications, and they provide hard bounds on delays or message losses. Deterministic techniques have the disadvantage of being based on worst cases, however, and they generally lead to significant underutilization of network resources. Statistical techniques offer more efficient resource utilization, but they cannot provide worst-case guarantees.

SPIN Micronetwork

The Scalable, Programmable, Integrated Network (SPIN) on-chip micronetwork defines packets as sequences of 32-bit words, with the packet header fitting in the first word. SPIN uses a byte in the header to identify the destination, allowing the network to scale up to 256 terminal nodes. Other bits carry packet tagging and routing information, and the packet payload can be of variable size. A trailer—which does not contain data, but a checksum for error detection—terminates every packet. SPIN has a packetization overhead of two words. The payload should thus be significantly larger than two words to amortize the overhead.

The SPIN micronetwork adopts cut-through switching to minimize message latency and storage requirements in the design of network switches. However, it provides some extra buffering space on output links to store data from blocked packets. Figure B shows SPIN's fat-tree network architecture, which derives its name from the progressively increasing communication bandwidth toward the root. The architecture is nonblocking when packet size is limited to a single word. Because packets can span more than one switch, SPIN's blocking is a side effect of cut-through switching alone.

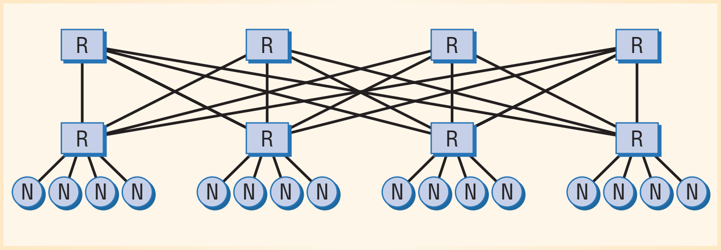


Figure B. SPIN architecture. R blocks are switches, N blocks are nodes.

SPIN uses deterministic routing, with routing decisions set by the network architecture. In fat-tree networks, tree routing is the algorithm of choice. The network routes packets from a node, or tree leaf, toward the tree root until they reach a switch that is a common ancestor with the destination node. At that point, the network routes the packet toward the destination by following the unique path between the ancestor and destination nodes.

Network design and control decisions will become critical as communication energy consumption scales up in SoC architectures.

The Silicon Backplane Micronetwork (http://www.sonicsinc.com), a shared-medium bus based on time-division multiplexing, offers an example of transport layer issues in micronetwork design. When a node wants to communicate, it must issue a request to the arbiter during a time slot. If arbitration is favorable, it may be granted access in the following time slot. Hence, arbitration introduces a nondeterministic waiting time in transmission. To reduce nondeterminism, the micronetwork protocol provides a form of slot reservation: Nodes can reserve a fraction of the available time slots, thereby allocating bus bandwidth deterministically.

Future development. The theoretical framework developed for large-scale networks provides a convenient environment for reasoning about on-chip micronetworks as well. Currently very scarcely explored, the micronetwork design requires further work to predict the tradeoff curves in this space. We also believe that this area offers significant room for innovation: On-chip micronetwork architectures and protocols can be tailored to specific system configurations and application classes. Further, the impact of network design and control decisions on communication energy presents an important research theme that will become critical as communication energy consumption scales up in SoC architectures.

SOFTWARE LAYERS

Network architectures and control algorithms constitute the infrastructure and provide communication services to the end nodes, which are programmable in most cases. The software layers for SoCs include system and application programs.

System software

The operating system captures the system programs that support SoC operation. System support software in current SoCs usually consists of ad hoc routines designed for a specific integrated core processor under the assumption that a processor provides global, centralized system control. In future SoCs, the prevailing paradigm will be peer-to-peer interaction among several possibly heterogeneous processing elements. Thus, we think that system software will be designed as a modular distributed system. Each programmable component will be provided with system software to support its own operation, manage its communication with the micronetwork, and interact effectively with neighboring components' system software.

Seamless composition of micronetwork components will require system software that is configurable according to the network's requirements. System software configuration may be achieved in various ways, ranging from manual adaptation to automatic configuration. One end of the spectrum favors software optimization and compactness while the other end favors ease of design and fast turnaround time. With this vision, on-chip communication protocols should be programmable at the system software level to adapt the underlying layers to the components' characteristics.

Most SoCs are dedicated to a specific application, and system software seeks to provide the required quality of service within the physical constraints of that application. Consider, for example, a SoC for a wireless mobile video terminal. Quality of service relates to the video quality, which implies specific computation, storage element, and micronetwork performance levels. Constraints relate to the strength and signal-to-noise ratio of the radio-frequency signal and to the energy available in the battery. Thus, the system software must provide high performance by orchestrating the information processing within the service stations and optimizing information flow. Moreover, the software should achieve this task while minimizing energy consumption.

The system software provides an abstraction of the underlying hardware platform. We can view the system as a queuing network of service stations. Each service station models a computational or storage unit, while the queuing network abstracts the micronetwork. Moreover, we can assume the following:

- Each service station can operate at various service levels, providing corresponding performance and energy consumption levels. This approach abstracts the physical implementation of components with adjustable voltage or frequency levels, or both, along with the ability to disable their functions in full or in part.
- The system software can control the information flow between the various units to provide the appropriate quality of service. This function entails controlling information routing, local buffering into storage arrays, and the information flow rate.

Thus, the system software must support both dynamic power management (DPM) of its components and dynamic information-flow management.

DPM entails selecting the appropriate component state to service a workload with the minimum energy consumption. For example, an ARM processor core can be in a sleep, idle, or run state. Moreover, the processor run state can be subdivided into substates corresponding to the operating frequencies and voltages. DPM policies provide the control algorithms for state transitions.[12] Transitions among states have a finite delay penalty, even when changing the operation frequency. Thus, identifying the policies that maximize performance under energy constraints or that solve the dual problem may be computationally complex.[12]

We envision at least two operating modes for SoCs with many power-manageable components. In one mode, the system software running on a power-manageable component determines its state transitions, based on the system state and its workload. Thus, the component's system software has a local DPM policy and controls the underlying hardware through appropriate system calls. In the second mode, components send their neighbors messages requesting state changes. Such requests originate and are serviced at the system software level. For example, an image processor can be required to raise its service levels before receiving a stream of data. In this case, the system software embodies policies that accept requests from other components and perform transitions according to such requests. At the same time, the policies can originate requests for other components.

Dynamic information-flow management configures the network and the local interconnection's bandwidth to satisfy information-flow requirements. This problem relates tightly to DPM, and we can view it as an application of DPM to the micronetwork instead of to a component. Again, policies implemented at the system software layer request either specific protocols or parameters at the lower layers to achieve the appropriate information flow, using the least amount of resources and energy.

The Maia processor[10] provides an example of information-flow management. It combines an ARM8 processor core with 21 satellite units, including processing and storage units. The ARM8 processor configures the memory-mapped satellites using a 32-bit configuration bus, and it uses two pairs of I/O interface ports and direct memory read/writes to communicate data with satellites. A two-level hierarchical mesh-structured reconfigurable network provides connections between satellites. Applying dynamic voltage scaling to the ARM8 core increases energy efficiency. This approach spatially distributes application programs to achieve energy savings an order of magnitude greater than for DSP processors with the same performance levels. Such savings derive from Maia's ability to reconfigure itself to best match the applications, activate satellites only when data is present, and operate at dynamically varying rates.

We believe that a layered-micronetwork design methodology will likely be the only path to mastering the complexity of SoC designs in the years to come.

Application software

Because SoCs will leverage many legacy applications, the system software must provide appropriate libraries and facilities to support standard programming languages. At the same time, dynamic binary code conversion may be a useful paradigm for providing alternative execution units for application programs.

SoC application software development should achieve two major goals:

- preserve portability and generality of the applications across different platforms, and
- provide some intelligence to leverage the distributed nature of the underlying platform.

One strategy for satisfying these apparently conflicting goals is to provide programming interfaces for applications and system software. Applications could then acquire information about the specific platform from the system software while providing adequate quality of service by asking the system software to set the hardware to specific states that guarantee the desired quality level. System software can serve or deny requests from applications according to the state of other running processes. Similarly, the software can neglect requests if it recognizes that they are not pertinent to the platform the system is using.

Despite numerous challenges, we believe that developers will find adequate solutions to the problems of designing SoC networks. At the same time, we believe that a layered-micronetwork design methodology will likely be the only path to mastering the complexity of SoC designs in the years to come. ■

References

1. B. Ackland et al., "A Single Chip, 1.6-Billion, 16-b MAC/s Multiprocessor DSP," *IEEE J. Solid-State Circuits*, Mar. 2000, pp. 412-424.
2. A. Agrawal, "Raw Computation," *Scientific Am.*, Aug. 1999, pp. 60-63.

3. L. Benini and G. De Micheli, “System-Level Power Optimization: Techniques and Tools,” *ACM Trans. Design Automation of Electronic Systems*, Apr. 2000, pp. 115-192.
4. R. Hegde and N. Shanbhag, “Toward Achieving Energy Efficiency in Presence of Deep Submicron Noise,” *IEEE Trans. VLSI Systems*, Aug. 2000, pp. 379-391.
5. W. Dally and J. Poulton, *Digital Systems Engineering*, Cambridge Univ. Press, New York, 1998.
6. J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach,* IEEE CS Press, Los Alamitos, Calif., 1997.
7. P. Guerrier and A. Grenier, “A Generic Architecture for On-Chip Packet-Switched Interconnections,” *Proc. IEEE Design Automation and Test in Europe* (DATE 2000), IEEE Press, Piscataway, N.J., 2000, pp. 250-256.
8. R. Ho, K. Mai, and M. Horowitz, “The Future of Wires,” *Proc. IEEE*, Apr. 2001, pp. 490-504.
9. D. Sylvester and K. Keutzer, “A Global Wiring Paradigm for Deep Submicron Design,” *IEEE Trans. CAD/ICAS*, Feb. 2000, pp. 242-252.
10. T. Theis, “The Future of Interconnection Technology,” *IBM J. Research and Development*, May 2000, pp. 379-390.
11. J. Walrand and P. Varaiya, *High-Performance Communication Networks*, Morgan Kaufmann, San Francisco, 2000.
12. H. Zhang et al., “A 1-V Heterogeneous Reconfigurable DSP IC for Wireless Baseband Digital Signal Processing,” *IEEE J. Solid-State Circuits*, Nov. 2000, pp. 1697-1704.

Luca Benini *is an associate professor in the Department of Electronics and Computer Science at the University of Bologna. His research interests include the design of portable systems and all aspects of computer-aided digital-circuit design, with special emphasis on low-power applications. Benini received a PhD in electrical engineering from Stanford University. Contact him at lbenini@deis.unibo.it.*

Giovanni De Micheli *is a professor of electrical engineering and, by courtesy, of computer science, at Stanford University. His research interests include design technologies for integrated circuits and systems, with particular emphasis on synthesis, system-level design, hardware and software codesign, and low-power design. De Micheli received a PhD in electrical engineering and computer science from the University of California at Berkeley. Contact him at nanni@stanford.edu.*