Addressing the System-on-a-Chip Interconnect Woes Through Communication-Based Design[1]

M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, A. Sangiovanni-Vincentelli
University of California at Berkeley, Princeton University
{sgroi, msheets, mihal, keutzer, jan, alberto}@eecs.berkeley.edu; malik@princeton.edu

ABSTRACT
Communication-based design represents a formal approach to system-on-a-chip design that considers communication between components as important as the computations they perform. Our "network-on-chip" approach partitions the communication into layers to maximize reuse and provide a programmer with an abstraction of the underlying communication framework. This layered approach is cast in the structure advocated by the OSI Reference Model and is demonstrated with a reconfigurable DSP example. The Metropolis methodology of deriving layers through a sequence of adaptation steps between incompatible behaviors is illustrated through the Intercom design example. In another approach, MESCAL provides a designer with tools for a correct-by-construction protocol stack.

GENERAL TERMS—Design

KEYWORDS—Network-on-chip; platform-based design; communication-based design; protocol stack.

1. INTRODUCTION
It is now not only possible, but also economical, to integrate complex systems on a single silicon die. Designing such systems on a chip (SOC) is a complex process, and is currently approached with little organizing principles. The Gigascale Silicon Research Center (GSRC) aims to provide the essential tools and methodologies to allow integrated circuit designers to make the transition from *ad hoc* SOC design to a disciplined platform-based design.

Essential elements of platform-based design are the design of the computation, *i.e.* the functional behavior of each core, and communication, *i.e.* its interaction between the cores. This *orthogonalization of concerns* is essential to the success of a re-use strategy as has been realized in recent years. The platform-based design methodology [9] addresses this concern by placing the computational cores and their interconnect strategy on the same footing.

As was learned by the telecommunications community a while ago, reliable communication between components requires the definition of a protocol that provides a set of rules dictating how the interaction among components takes place, so that the overall system communication and performance requirements are met, while physical resources such as area and energy are minimized. Traditionally, on-chip communication design has been done using rather *ad-hoc* and informal approaches that fail to meet the challenges posed by next-generation SOC designs, namely:

- *Predictability* - The capability of making early decisions based on the expected performance of the final implementation is very important in communication design to avoid time-consuming design iterations. The shift towards deep-sub-micron integration makes this aspect even more critical. The increasing ratio of the delay of long wires with respect to gate delay and the dependence of the propagation delay on the chip topology makes it increasingly hard to have the system functionality rely on physical parameters only.
- *Wiring delay* - SOCs that occupy a large area and require long wires to connect communicating components face relevant delay and synchronization problems, especially if multiple (potentially asynchronous) clock domains are used.
- *Power dissipation* - The power consumed by the interconnect structures, including clocks, is rapidly becoming a dominant component of the overall power-budget.
- *Diverse interconnect architectures* – In the past the choice of the interconnect architecture was limited to a few choices, given the small number of blocks that had to be interconnected and the relative simplicity in dominating the performance and delay trade-offs. For SOCs, a richer set of interconnect schema should be examined: for example, shared communication resources such as busses, crossbars, and meshes to minimize resource needs. Solving the latency *vs.* throughput tradeoff now requires to take in consideration a large number of design parameters, like pipeline stages, arbitration, synchronization, routing and repeating schemes.

To address these challenges it is critical to take a global view of the communication problem, and decompose it along lines that make it more tractable while not restricting the design space at the same time. Communication design has to begin at higher levels of abstraction than the architecture and RTL level. We believe that *a layered approach similar to that defined by the communication networks community* (and standardized as the ISO-OSI Reference Model (RM) [18]) to address the problem of connecting a large number of computers on wide-area networks should also be used for on-chip communication design. The layered approach is well suited to describe protocol functions that operate on data units at different levels of abstraction (in the form of streams, packets, bits or analog waveforms) and that are subject to various time granularity constraints. Each layer may include one or more closely related protocol functions, such as data fragmentation, encoding and synchronization.

Separating the communication protocol functions into layers that interact only via well-defined interfaces allows for a decomposition of the design problem into a set of simpler, tractable problems, and

[1] This research is sponsored in part by the Marco GSRC center, DARPA-ITO and CNR.

DAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.

simplifies the synthesis and validation tasks. As amply demonstrated in the communication-network domain, the approach also maximizes re-use. An excellent example in case is the 802.11 wireless local-area network standard, where a single media-access layer supports different physical implementations through a unified interface.

We call the layered-stack approach to the design of the on-chip inter-core communications the *Network-on-Chip* (NOC) methodology. Designing NOCs is not an easy task, and may result in protocol implementations that are incorrect (e.g. due to deadlocks and race conditions), or sub-optimal (e.g. are power hungry or introduce unacceptable latency). To avoid such problems, it is essential to develop a new generation of methodologies and tools that:

- By applying a discipline to on-chip communication design transition from ad-hoc SOCs to disciplined IC platforms.
- Are based on formal Models of Computation and support a correct-by-construction synthesis design flow and a set of analysis tools for broad design exploration.
- Maximize re-use with the definition of a set of interfaces between layers
- Provide an application programmer with a set of APIs abstracting architecture details.

While NOC design so far has not been addressed in a structured and rigorous manner, several approaches have been developed that are based on a similar point of view.

- The Virtual Socket Interface (VSI) Alliance [16] has developed a standard interface to connect virtual components (VCs) to on-chip buses. VCs and buses are adapted to this interface wrapping them with appropriate glue logic.
- The Cosy [2] approach, based on the infrastructure and concepts developed in the VCC framework [3] defines interfaces at multiple levels of abstraction. The top level, called application, is purely functional. Application-level transactions are then refined into system transaction when a hardware or software implementation of the upper layer functions is selected. This layer deals with selecting communication parameters and solving latency-throughput tradeoffs. Below, the Virtual Component Interface is adopted to interface with specific Physical bus protocols.
- The Sonics [15] μ-network approach de-couples the design of the communication among IPs. Each IP core communicates with an agent in the Silicon Backplane using a protocol called OCP (Open Core Protocol) and agents communicate with each other using network protocols. Both protocols can be customized by the SOC designer who can configure parameters such data width and buffer depth.
- One important aspect of the NOC problem— the reservation of network resources such as buffers and bandwidth—is addressed by B. Dally, who proposes the flit-reservation flow control ([12]). This approach makes use of packets sent over fast control wires to reserve resources in the data connection layer and allows to optimize the use of buffers without penalties in latency.

In this paper, we, first, describe the OSI Reference Model (RM) and discuss its use for NOCs with an example of application, the Pleiades platform. Then, we present the Metropolis methodology and framework with an application, Intercom. Then we present MESCAL, another approach to the design of NOCs, and offer some concluding remarks.

2. OSI REFERENCE MODEL APPLIED TO NOCs

The OSI RM is a framework that allows us to classify and describe network protocols. Since its standardization, it has been used as a reference for wired and wireless computer-network design. However, the same layering concept and many of the protocol functions can be used also to realize NOCs. Below, we briefly describe the seven OSI layers and for each layer we give examples of on-chip application.

- **Physical**—The physical layer is concerned with the lowest-level details of transmitting data (bits) on a medium. The NOC physical layer protocols define such things as signal voltages, timing, bus widths, and pulse shape. At this level delay and power consumption may be difficult to predict. The floorplan of the chip can have a dramatic effect on both of these metrics, as well as the actual routes chosen for the wires. Also of particular concern at this layer is the synchronization of signals, since IPs may be in different clocking domains or require asynchronous communication.
- **Data Link**—The data-link layer is responsible for reliable transfer of data over the physical link, and may include error detection and correction functions. It must also arbitrate the access to a shared physical medium, like a bus. Examples of Medium Access Control (MAC) protocols are token ring and time division multiplex access (TDMA). Delay predictability, throughput and power consumption may vary significantly depending on which arbitration scheme is adopted.
- **Network**—The network layer provides a topology-independent view of the end-to-end communication to the upper level protocol layers. The connections established in the network could be static, or dynamic, such as offered by the reconfigurable interconnect of FPGAs. Similarly, data routes can be persistent over multiple transactions, or each transaction can be dynamically routed. In the latter case, congestion control may be required to reduce traffic through overburdened links.
- **Transport**—Transport layer protocols establish and maintain end-to-end connections. Among other things, they manage flow control, perform packet segmentation and reassembly, and ensure message ordering. This abstraction hides the topology of the network, and the implementation of the links that make up the network. Therefore, it is used by the layers above the transport layer to provide components with more formal methods of communication.
- **Session**—Session layer protocols add state to the end-to-end connections provided by the transport layer. A common session protocol is synchronous messaging, which requires that the sending and receiving components rendezvous as the message is passed. The state maintained by the protocol is a semaphore that indicates when both the sender and the receiver have entered the rendezvous. Many embedded system applications utilize this sort of functionality to synchronize system components that are running in parallel. This is especially true when the system components are CPU-like processing elements that execute software programs.
- **Presentation**—The presentation layer is concerned with the representation of data within messages. Protocols at this level convert data into compatible formats. For example, two system components may exchange messages with different byte orderings, so this layer converts them to a common format.
- **Application**—This layer exports to the system components the highest level of abstraction of the underlying communication architecture. For example, in an embedded system that performs

video processing the basic communication function may be to transfer a video frame from one system component to another. The application layer would define a function that does exactly this by utilizing the functions defined at lower stack layers. The system components can use these abstract communication functions without concern for the details, thus simplifying the component design.

Communication-based design uses the stack model as a tool to guide the decomposition of the design problem. The separation of computation and communication reveals the communication requirements of the system. The application layer provides the set of communication functions that implement those requirements. It does so by building upon the functionality defined at lower levels in the stack model. In most cases, it is not necessary to implement protocols at all of the OSI stack layers to provide this high-level functionality. One of the benefits of the OSI stack model is that it scales to match the needs of the system components. If the system components do not require connections with state, data format conversion or other features, the corresponding stack layers can be omitted. However, as embedded systems scale in complexity their communication architectures will have to scale in functionality as well.

The layered approach of OSI Model is a useful method for structuring and organizing a protocol at an early stage of the design process. However, on the way to implementation, designers may consider whether the original layering structure should be maintained, or whether performance is optimized by combining adjacent layers.

An NOC Example: The Pleiades Platform

The Pleiades platform (in its instantiation, the Maia processor) [17] presents a reconfigurable integrated circuit for DSP applications that demonstrates how the NOC layers of abstraction are applicable to existing designs. The basic Pleiades architecture is a heterogeneous collection of satellites such as arithmetic logic units (ALUs), memories, processors, FPGAs, and multiply-accumulators. This collection of interconnected satellites is analogous to a set of IPs present in a SOC design. From a communication perspective, the computation at each satellite is arbitrary because each is wrapped in an inter-satellite communication interface.

This interface is actually the physical layer in the NOC framework, because it specifies the signal definitions, timing, and synchronization between two satellites. In the Pleiades case, this means that data links are 18-bits wide and have 2 control bits. Additionally, each satellite operates on a local clock, which is not necessarily coincident with the other satellite clocks. For this reason, the interface is self-timed through a two-phase asynchronous handshaking scheme. Lastly, the communication reduces power consumption through reduced-swing signaling. Individual links can be well characterized with predictable delay and energy consumption. Since each link in the Pleiades architecture is dedicated and error-free, no data-link layer is required.

It is in the network layer that the Pleiades architecture is especially novel. The interconnect consists of a two-tiered hierarchical mesh to provide energy efficiency as well as the required flexibility. At the local level, universal switchboxes provide a method for programmatically connecting wires with a cluster. The global level provides switchboxes connected in a larger-granularity mesh for inter-cluster communication. The switchboxes enable persistent paths while allowing satellites reconnection to implement a different algorithm. A network connection is set up at (re)configuration time, and is rewired every time a new task is over-laid on the configurable fabric Because of this flexibility, the energy consumption and delay are dependent upon the actual path through the switchboxes. From a refinement perspective, higher levels of abstraction cannot know these values a priori so upper bounds or statistical averages can be used to provide early estimations of these metrics. Additionally, constraints can be used in the refinement process to influence the programming of the actual routes.

3. METROPOLIS APPROACH

The Metropolis project [13] is developing a formal methodology for SOC design, based on the principles advocated in this paper.

3.1 The Metropolis Methodology

In the Metropolis methodology, the SOC designer first describes (or selects from IP libraries) the blocks that perform computations and, then, designs the communication among them using a rigorous successive refinement process. To maximize reusability, the layers of the protocols should only encapsulate the original computation cores without any change in their internal structure.

The design of communication begins with the declaration of a set of constraints that the protocol must satisfy and moves towards a final implementation through a sequence of *successive refinement* steps. Constraints usually consist of a set of formulae including variables such as power, number of errors, delay. They are propagated (e.g. through budgeting) at each design step, while the current specification is either refined with the addition of new details or is (partially) mapped onto architectural elements, such as physical channels and protocol layers that are selected from libraries. Note that the architecture/function orthogonalization principle applies to all levels of the design hierarchy as well as the constraint mapping mechanism. We refer to this aspect as the *fractal nature* of design since the same pattern repeats itself on all scales.

Metropolis is based on a formal representation of the system specification throughout all levels of abstraction, but is not biased towards any specific Model of Computation (MoC) or communication semantics. At the highest level of abstraction, the system specification is purely *denotational*, i.e. the system is described as a set of concurrent components, called processes, each defined as an input/output relation. In contrast to the traditional design practice, where the specification of the processes also includes details of the communication interface and hence is not easily reusable, the communication among processes is separated and dealt with explicitly. For example, at this level one can view an MPEG encoder as composed of components like DCT Transform, Huffmann encoding and motion compensation, without any information of how they interact (e.g. block by block or frame by frame communication). Communication is defined separately, when a set of protocols and a medium that physically connects the communicating components are selected.

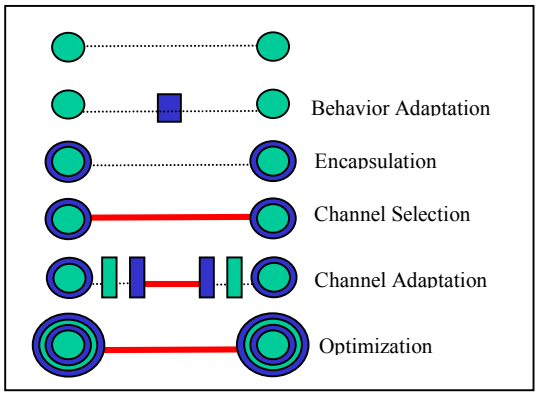
To explain our approach to communication design, we introduce the *adapter* concept [13]. For two behavioral objects to communicate with each other, they must agree on the semantics and syntax. If a mismatch in either is found, an adapter must be introduced.

Consider two processes that are connected, i.e., the output of a process, called sender, is connected to the input of the other process, called receiver. If the sender's output domain, defined as the set of its output signals, is different from the receiver's input domain—the set of input signals for which its behavior is defined—, an object that maps (at the semantic or syntactic level or both) signals from one domain to the other must be interposed between sender and receiver. For example, consider an IP block that sends 128-bit packets to a receiving IP that

accepts only 24-bit packets: an object breaking the large packets into smaller ones is needed. These functions, which we call *behavior adapters,* are the key to enable reusability of pre-designed blocks. Behavior adapters are also used every time a bridge between two networks using different protocols is needed. Behavior adapters can be synthesized automatically, as demonstrated in [11] for interfaces between incompatible hardware protocols.

So far, we assumed that communication between blocks occurs over a logical connection. To derive a physical implementation, the designer must select a channel, such as a wire, a bus or a network that can *physically transport* signals. In Metropolis each channel is defined by a set of properties, such as FIFO ordering, error rate and bandwidth, and a set of interfaces (e.g., read and write) that the processes connected to the channel can access. Once a physical channel is chosen, the designer must select also a model of computation that defines the firing rules that the processes must follow when they access the channel. The firing rules are part of a *MoC wrapper* that encapsulates each process. If the selected physical channel does not immediately meet the communication constraints on parameters like delay, throughput, reliability, it is necessary to introduce adapters (called *channel adapters*) between sender and receiver, and the channel. Consider the problem of implementing a reliable error-free connection. If an unreliable channel with a non-negligible error rate is selected, it is necessary to introduce adapters, e.g. encoding or retransmission functions, between the unreliable channel and the sender and receiver.

When no more adaptation steps are needed, all protocol layers are clearly identified. At this point, opportunities for optimizations should be explored, for example merging adapters and communicating processes. In general, behavior of processes may be changed due to these optimizations, and they may not be easily reusable as a result. One can always return to a higher level of abstraction at which modification can be kept local without affecting the rest of the system.



Behavior and channel adapters are the building blocks of our protocol-design methodology. Their presence and the order in which they are defined may vary from case to case. They roughly correspond to classical protocol layers defined in the OSI model, even though an OSI layer often includes multiple protocol functions, hence multiple adapters. However, we believe that taking a view of the protocol design process as a sequence of adaptation steps between incompatible behaviors and interfaces allows us to define a methodology for selecting and ordering protocol functions as well as to develop techniques for automatic protocol synthesis.

3.2 A Metropolis Example: Intercom

A case study for this successive refinement approach is the implementation of an Intercom mobile terminal that supports voice communication over a wireless LAN [14]. The final architecture of the Intercom includes an embedded microprocessor and custom logic connected through a chip-wide shared bus provided by Sonics, Inc [15]. Here, we describe the systematic approach used to design the NOC.

We used the Cadence VCC design environment [3] to capture the behavior of the system and evaluate the cost and performance of different implementations. This environment is based on the orthogonalization of function and architecture and is a pre-cursor of the Metropolis paradigm of separation between communication and computation realized via the concept of architectural services. A library of different communication architectures (e.g. a variety of busses and crossbars) is available to a designer, enabling her to evaluate the effects of the choice of communication scheme on the performance of the design. VCC models the function of the system components as a network of asynchronously communicating finite state machines (CFSMs). Clearly, this model of communication does not accurately reflect any realizable implementation of a communication network so refinement is required. Once each CFSM is mapped onto a physical resource, such as the embedded processor or custom logic, the communication refinement and implementation process begins.

The goal is to design a NOC that allows the embedded processor, custom logic, and memory to communicate on the chip. The first step is the selection of a physical channel and corresponds to the definition of the physical layer in the OSI RM. Similar to the Pleiades example, this refinement defines a standard interface that is implemented for each component in the system. In this case, this interface consists of a 32-bit data bus, a 32-bit address bus, and a set of control wires.

The first adaptation step is required because data transfer must ultimately occur in discrete words over wires. Hence, a block performing segmentation and reassembling of data exceeding the width of the bus is introduced to adapt the sender and receiver interfaces. Next, a medium access controller (MAC) adapter is introduced to arbitrate the access to the shared medium. For this purpose a time division multiple access (TDMA) scheme is used, where the arbitration policy for time-slots is round-robin token passing. This arbitration policy precludes starvation but makes accurate characterization of delay difficult to predict since it is dependent upon the communication profiles of the other components sharing the medium. The delay can be easily given an upper bound, but this often results in a pessimistic estimate.

A refinement at each receiver determines whether it is the intended destination for each communication by examining the address of the data. Similar memory-mapped addressing schemes are common in contemporary SOCs, especially in those containing embedded microprocessors. In addition, since the sender may have to wait to gain access to the medium, a buffer is introduced to queue the transaction. Since all the components in the system use the same clock, no explicit refinement is required for synchronization. The bit width, MAC, addressing, and buffer adaptors comprise the data link layer of the Intercom system. Since all interfaces between the components, adaptors, and medium are compatible, the refinement process is complete.

We used the Open Core Protocol specification and FastForward toolset of Sonics, Inc., whom we thank for the support, to generate an

interconnect implementing the network, data link, and physical layers of the Intercom project.

4. MESCAL

MESCAL stands for Modern Embedded Systems, Compilers, Architectures, and Languages [9]. The goal of the MESCAL project is to provide a programmer's model and software development environment that allows for the efficient implementation of an interesting set of applications onto a family of fully programmable architectures. MESCAL is based on the assumption that domain-specific programmable solutions are required to deliver the benefits of programmability while still delivering acceptable performance. Design reuse within an application domain is accomplished through software programmability of the system components in the architecture.

As we have seen in previous sections, any successful solution to the design reuse problem must fully take into account the communication requirements of the embedded application. These communication requirements are revealed in the high-level application models that are used to describe the applications. The task of the designer is to implement these communication requirements through the *communication architecture* of the target platform. The communication architecture covers everything from the physical interconnects to the software that the application processes use to perform communication. We use the OSI stack model as the foundation of a divide-and-conquer approach to the design of communication architectures.

In the MESCAL project, we seek to provide a set of tools that support the stack methodology for communication architecture design. These tools provide a method of formally specifying protocol stacks for on-chip networks. In this section we show that by providing the stack model with a set of formal semantics we can perform correct-by-construction synthesis from stack diagrams to communication architecture implementations. Also, we show how the MESCAL programmable architecture allows for the flexibility of communication architectures after fabrication.

4.1 MESCAL Model-of-Communication

Just as the application model is a high-level view of the application's computational requirements, the stack model is a high-level view of the system's communication requirements. Formal models of computation give application models meaning by providing them with functional semantics. We can provide similar functional semantics to the stack model. Anyone who is familiar with the OSI stack and network protocols can look at a diagram of a protocol stack and understand the communication system that it is describing. The functional semantics are a formalization of this intuition. This will guarantee that the stack model will have meaning beyond a conceptual diagram. The stack model will become a functional model that can be used to define the behavior of communication architecture.

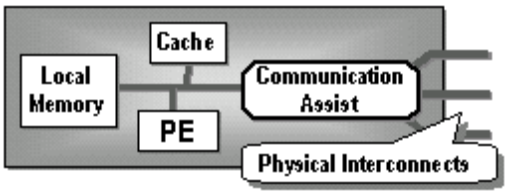
The goal of the MESCAL design environment is to allow architects to design communication architectures using a graphical stack model as a high-level description "language". The formal semantics underlying the stack model enable a correct-by-construction synthesis path from this high-level description to an implementation. Our formal model of computation for the OSI stack is based upon Communicating Sequential Processes (CSP) [6]. In this model a protocol is viewed as a process that performs computation on data packets. Packets are passed through the stack model from one protocol-layer to the next using a rendezvous similar to that defined in CSP.

The MESCAL project utilizes the Ptolemy II heterogeneous modeling environment as a framework for building our design tools. We are implementing a Ptolemy *domain* for describing communication architectures using the stack model. When it is complete, we will be able to describe stack models of on-chip networks in a formal framework. The environment will provide an extendable library of Ptolemy actors that implement common on-chip network protocols. Designers create stack models by assembling protocol actors. Synthesis tools use knowledge of the underlying formalisms to turn the models into implementations. Previous research projects, including the x-kernel [8] and the CLICK [10] modular router, also create frameworks for working with networking protocols, yet do not address the specific issues of programmable NOCs that are the focus of MESCAL.

4.2 MESCAL Communication Architecture

The programmable nature of MESCAL architectures provides flexibility after systems are fabricated. As we treat both processing elements and communication architectures as ``first class citizens'', we seek to achieve flexibility for communication architectures as well as the programmable components themselves. Unless reconfigurable hardware is used, the low-level aspects of a communication architecture (such as physical buses) are fixed at fabrication. However, a great deal of flexibility can still be achieved by implementing the top layers of the protocol stack in software. Then, when the application's communication requirements change, the system can adapt by exporting a new set of communication functions to the system components.

The block diagram of the basic building block of the MESCAL architecture, known as a *communicator,* is shown in the Figure below. It combines a VLIW-like processing element, some local memory and cache, and a coprocessor called a communication assist (CA). The CA coprocessor offloads much of the overhead of communications from the processing element. A similar approach was introduced earlier in [7]. The topmost layers of the stack model are implemented in software on the CA coprocessor. This allows them to be modified even after the architecture is fabricated. The lowest layers of the stack model, such as interfaces to physical interconnects, are implemented as hardware peripherals on the CA coprocessor. All communications between processing elements is handled by the hardware and software of the CA coprocessors.



MESCAL system component

We believe that the communication-assist provides the right degree of flexibility for our communication architectures. When an architecture is designed for a specific application, all features of the communication architecture including the physical hardware implementation can be customized for that application. After fabrication, the programmable nature of the communication assist allows the system to adapt to the changing needs of the application. The flexibility lies in the implementation of the upper layers of the stack model. These are the layers that provide the highest-level communication functionality to the processing elements.

5. CONCLUSION

The Gigascale Silicon Research Center aims to provide the key tools and methodologies to allow integrated circuit designers to make the transition from *ad hoc* system on a chip design to a disciplined approach to platform design. One of the essential elements of this transition is taking a rigorous, though flexible, approach to the design of on-chip networks that interconnect IP blocks of all variety, including processing elements. In this paper we have outlined essential elements of this design discipline and illustrated the approach with two design examples. Two design methodologies (Metropolis, Mescal) based on these concepts have been detailed. It is the authors' belief that a network-on-chip approach, driven by a consistent design methodology, is bound to lead to dramatic changes in how SOCs will be constructed in the next decade. The GSRC project has offered us an invaluable framework to test new ideas and interact among ourselves and the other researchers involved in other aspects of SOC design.

6. REFERENCES

[1] F. Balarin et al., "Hardware-Software Co-Design of Embedded Systems: The POLIS Approach", Kluwer Academic Publishers, 1997.

[2] J. Y. Brunel et al., "Cosy communication IP". Proceedings of the Design Automation Conference, Los Angeles, CA June 2000.

[3] Cierto VCC, Cadence Design Systems. http://www.cadence.com/technology/hwsw/ciertovcc/

[4] Commercial Video Processors. MIT, Cambridge, MA. http://wad.www.media.mit.edu/people/wad/vsp/node1.html.

[5] T. R. Halfhill. "Intel Network Processor Targets Routers". Microprocessor Report, Vol. 13, September 13, 1999.

[6] C. A. R. Hoare. Communicating Sequential Processes. Prentice Hall International Series in Computer Science, 1985.

[7] M. Horowitz and K. Keutzer. ``Hardware-software co-design''. In *SASIMI'93,* October 1993, pp. 5-14.

[8] N. C. Hutchinson and L. L. Peterson. "The x-kernel: an architecture for implementing network protocols". IEEE Transactions on Software Engineering, Vol. 17, No. 1, pp. 64-76.

[9] K. Keutzer et al. "System-Level Design: Orthogonalization of Concerns and Platform-Based Design". IEEE Transactions on Computer-Aided Design. Vol. 19, No. 12. December 2000.

[10] E. Kohler, R. Morris, B. Chen, J. Jannotti, F. Kaashoek. "The Click Modular Router". ACM Transactions on Computer Systems, Vol. 18, No. 3, August 2000, pp. 263-397.

[11] R. Passerone et al., "Automatic Synthesis of Interfaces Between Incompatible Protocols", Proceedings of the 31st Design Automation Conference, San Francisco, CA, pp. 8-13, June 1998.

[12] L. .S. Pen and B. Dally, "Flit-Reservation Flow Control", Proceedings of 6-th International Symposium of High-performance Computer Architecture, Jan. 2000.

[13] A. Sangiovanni-Vincentelli et al., "Formal Models for Communication-based Design". Proceedings of the 11-th International Conference on Concurrency Theory, Concur '00, August 2000.

[14] J. Silva et al. "Wireless protocols design: challenges and opportunities," Proceedings of Int. Workshop on Hardware/Software Codesign, May 2000.

[15] Sonics Inc. http://www.sonicsinc.com/

[16] VSI Alliance. http://www.vsi.org/

[17] H. Zhang, et al. "A 1-V hererogeneous reconfigurable DSP IC for wireless baseband digital signal processing," IEEE J. Solid State Circuits, vol. 35, Nov. 2000, pp. 1697-1704.

[18] H. Zimmermann, OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection, IEEE Transactions on Communications COM-28, No. 4: April 1980.