Early analysis tools for system-on-a-chip design

by J. A. Darringer
R. A. Bergamaschi
S. Bhattacharya
D. Brand
A. Herkersdorf
J. K. Morrell
I. I. Nair
P. Sagmeister
Y. Shin

The paper describes the need for early analysis tools to enable developers of today's system-on-a-chip (SoC) designs to take advantage of pre-designed components, such as those found in the IBM Blue Logic® Library, and rapidly explore high-level design alternatives to meet their system requirements. We report on a new approach for developing high-level performance models for these SoC designs and outline how this performance analysis capability can be integrated into an overall environment for efficient SoC design.

1. SoC design

As silicon technology continues to advance, designers are finding that they can implement most of their product on a single chip. For example, the current IBM CMOS technology, Cu-08, provides more than 70 million wirable gates with eight levels of copper interconnect; this enables products with a broad and growing diversity of applications (communication networks, storage networks, set-top boxes, games, servers, etc.) to be realized as SoC designs with higher performance and lower cost. While SoC design offers many advantages, there are still the familiar challenges of designing a complex system, now on a chip. The ever-shortening time-to-market compounds these challenges. Without a major advance in productivity, designers will be able to consider only a very few high-level system designs and will have to limit their product differentiation to the software running on a standard embedded processor.

To address SoC design productivity, semiconductor suppliers are advocating intellectual property (IP) reuse [1, 2]. Designers are provided access to a library of large, previously designed components, or "cores," with the goal of enabling them to rapidly realize their system by assembling a network of these cores. The IBM Blue Logic* Library [3] contains more than 300 verified cores that provide important functions for communication networks, data compression, encryption, high-speed links, and bus interfaces, along with a wide variety of embedded processor cores from the extensive PowerPC* Roadmap and other processor cores for special applications such as signal processing. The CoreConnect* Architecture [4] provides a foundation for interconnecting IBM cores as well as non-IBM devices. Key elements of the architecture are a high-speed processor local bus (PLB), an on-chip peripheral bus (OPB), a PLB–OPB bridge, and a device control register (DCR) bus.

Even with the benefit of a core library, today's SoC designers still have many options available, and making the right choices can be difficult. Short schedules also rush these early critical decisions that are so costly to change later in the design process. Functions must be allocated to hardware and software components, and the correct hardware components and the best interconnection scheme must be selected, while considering functional, performance, and cost constraints. At the early stage of design, there are usually many unknowns. The software

0018-8646/02/$5.00 © 2002 IBM

may not be ready, the new custom logic may not be completely defined, and even the market requirements may still be in flux. Still, there are usually known performance and cost targets, set to ensure a competitive product. The designer must conduct an early analysis to make sure that these targets can be met, while preserving the flexibility to complete the design later in the schedule.

Some of the questions that must be answered at an early stage are the following:

- What functions should be implemented in hardware or software?
- Which embedded processor should be used? Or can the chosen processor handle the software functions within the real-time constraints of the system?
- What is the worst-case interrupt latency?
- What is the internal bus utilization?
- Can the chosen architecture/component be laid out with the available chip size?
- What is the expected system power consumption? Is this within the limits for the chosen packaging technology?

Most designers rely on a register-transfer-level (RTL) system specification and design flow for implementation and for analysis. But to use this, all registers or memory elements must be identified along with the precise time and conditions under which data is transferred among them. The questions listed above have to be addressed early in the design cycle, usually prior to the existence of a complete and detailed description for the system. Therefore, tools are needed to analyze a higher-level representation of the design and answer questions about performance, floorplanning, power consumption, area, and timing.

Using pre-designed cores permits reuse of detailed descriptions of the core logic that provide more precise characterization of the core properties. Once the cores are interconnected and any new logic specified in an RTL description, the proposed design can be simulated to evaluate alternatives. However, for large SoC designs, the complete RTL description is also quite large, and simulation runs take considerable time. To evaluate performance, actual software is usually needed to specify embedded processor models. Further, simulation addresses only one aspect of early analysis. Performance is critical, but the SoC designer must also determine the chip size in order to establish cost and packaging strategy. Doing this with some certainty requires a chip floorplan with cores placed or assigned to specific regions. Major interconnects have to be routed to avoid later congestion problems and to ensure that the buses and other logic can operate at the target clock frequency. In summary, using a complete RTL description of an SoC for early design analysis is nearly as difficult as implementing the chip itself, and requires significant resources.

Power is increasingly important for electronic systems because of limited battery capacity for portable systems, environmental issues, and energy delivery cost for stationary systems. Since early decision has a major impact on the power consumption of the final product, it is important to be able to obtain an early estimate of the SoC power consumption. This presents opportunities for power optimization at the SoC design level: core selection, design of application-specific power-management units, and even software design, including operating system and applications. Power estimation is a vital element of early analysis and has to be based on a system-level representation above the RTL. A method is needed that allows this early analysis to be based on partial and very high-level information.

2. Early analysis

To address the challenges described above, we need an extended design process with the ability to evaluate options and make critical architectural decisions based on a system-level representation in advance of an RTL design. A key prerequisite is a library of abstract models, one for each core or CoreConnect element, that captures their respective performance, power, and physical characteristics. Core placement from a floorplanner and bus layout from a global router directly influence these evaluations. In addition, performance data acquired during system-level design space exploration can help determine switching factors for power dissipation and signal integrity analysis. Thus, system-level performance analysis influences early physical design decisions which, in turn, cause the downstream tools with derived constraints to accelerate the chip implementation process and ensure that the early design assertions and assumptions are met. This process leads to a set of requirements for addressing early design evaluation.

System-level representation

Early analysis begins with the designer specifying a system-level description of the SoC design. This includes the following steps:

- *Identifying the SoC components*. The components, including buses and power-management circuitry, are selected from a given technology library of cores. An SoC also may contain a design-specific element that must be modeled as part of the process. The core library components include the necessary performance, power, and physical information.
- *Interconnecting the components*. Describing the connections between the chip interface and the constituent cores is a complex task that requires extensive knowledge of the

CoreConnect bus architecture and each core in order to understand how each pin should be connected. Support should be provided for automating this process as much as possible.

- *Describing clock domains*. If appropriate, the designer may wish to identify distinct clock domains, especially if they have to be considered during floorplanning. For example, large structured clock buffers (SCBs) can compete for real estate with the cores. Information required includes clock frequencies, their source, and their interfaces with various components.

This information may be captured during a logical editing session, during floorplanning, or through a set of simple commands, any of which results in a representation that can be directly used by the evaluation tools.

Performance analysis

Performance is usually the criterion that determines the SoC architecture. Even when power dissipation or cost is the dominant concern, performance analysis is needed to establish switching factors or to trade off components. To avoid the delay of developing a custom performance model, a method of generating one automatically from the system-level diagram is needed. There is a broad range of performance questions about the utilization of resources such as processors, memories, buses, and key cores, as well as data rates and system throughput. At the same time, there is a growing diversity of application areas such as communication and storage networks, video products, games, and controllers that have to be modeled. Accuracy and correlation with the final SoC implementation are also critical concerns. Detailed models may increase accuracy, but could slow analysis turnaround times and prevent extensive design space exploration. Whatever the approach taken, it is essential that a performance analysis capability be readily available to the SoC design teams for their use at the critical early stages of design.

Floorplanning

A floorplan is essential for determining chip size, the number of wiring layers, and congestion, all of which contribute to the cost of the SoC. Also, the placement and routing of global interconnections can determine parameters that affect the performance analysis described above. This process of floorplanning is aimed at determining the following:

- *Chip die size with pin assignments*. A “generic chip image” can be used that predefines only the minimum amount of technology information needed for the floorplanner to predict the minimum die size required to contain the design. Optimal chip pin assignments are also considered as part of the floorplanning process, along with the placement of the related I/O circuitry, all concurrent with the component placement and form-factor determinations.
- *Placement of the components*. The bus-level connections can be used as constraints by the floorplanner to help guide the placement in a way that maximizes the efficiency of the downstream tools. Clock redistribution and power-management circuitry must also be taken into account, as well as any electrical constraints associated with the I/O circuitry.
- *Core form factors*. Some components may be “hard” macros that have already been physically implemented and have fixed dimensions as well as fixed pin positions. Others may be “soft” macros, which are defined logically but are to be implemented as part of the SoC design. Still others are defined through assertions—for example, the design-specific elements with designer-defined areas, pin counts, etc. In the latter two cases, the form factors can be manipulated in concert with the component placement to achieve the optimal floorplan.
- *Component pin assignments*. Using the bus-level connections as a guide, the macro pins should be optimally positioned around the three-dimensional outline of each of the “soft” and “asserted” macros as a way of guiding the global router.
- *Global routes*. The layout for the buses and other critical interconnections is done to provide more accurate wiring information for the performance analysis evaluation.
- *Assignment of the terrains and voltage islands*. Some new technologies allow portions of the chip to have different circuit densities or different operating voltages. The use of multiple terrains gives the designer more freedom in meeting performance, congestion, and power constraints. These assignments are also used in estimating power dissipation.

Power estimation

Power consumption is another key consideration in today’s designs. It determines battery lifetime, package selection, cooling requirements, and product cost. Therefore, a power analysis capability is needed to identify power-critical parts for power optimization. Such analysis must take into account any built-in power-management schemes and utilization of different operating voltages that may be offered by the technology. It has to take advantage of the information available from performance analysis and floorplanning to provide increased accuracy. Also, detailed information about cores, such as the number of latches and internal power-management schemes, should be used where appropriate. The number of power states and their abstraction should be chosen carefully to provide efficient power modeling and estimation. Absolute accuracy is not as important as relative accuracy, since the estimation must correlate with the final implementation in order to ensure that early decisions based on power estimation have a positive impact on the final product.

Predictable path to RTL flow

Time spent on system-level decisions may be wasted if there is no effective means of ensuring that the downstream RTL tools can accept and meet the specified constraints. Therefore, we need to ensure that the estimates used to close on the design specification are effectively passed to the downstream tools to minimize unexpected and expensive surprises. We list here an initial set of such information that should be passed to the RTL design:

- Die size and chip pin assignments.
- Component areas and placement.
- Global interconnect planning.
- Clock redistribution planning.
- Performance constraints.
- Power constraints.
- Switching factors (resulting from performance analysis).

Although described individually, the above set of capabilities should be provided in a single cohesive environment in order to maximize the efficiency of the SoC design space exploration as the designer attempts to identify and evaluate alternative approaches to the system-level design. Ideally, there should be a consistent user interface that the designer can use to both control the operation and view or modify the results, and data should be passed seamlessly among the constituent capabilities as the evaluations are performed.

3. Available solutions

System-level tools can be broadly divided into three distinct categories: design, verification, and performance analysis tools.

New tools for system-level design, design-space exploration, and verification have been the focus of academic research for several years. Most of the approaches have been directed at hardware–software co-design and co-verification problems. Examples of such work include Polis [5], COSMOS [6], COSYMA [7], MOSES [8], Ptolemy [9], and SHE [10]. This research led to the development of commercial tools, such as VCC [11], CoWare [12], and Seamless** [13].

These design tools follow a vertical functional approach, starting from a functional description of the complete system, which is then decomposed into hardware and software parts. Each part is gradually refined into an implementation that usually involves an SoC containing one or more embedded processors, memory and other hardware units, and software that is compiled and run on the embedded processor(s). The drawback of this functional approach is the lack of strong links to a core-based hardware implementation. In most cases, the hardware and software partitioning is achieved with functional or physical constraints, e.g., real-time constraints on the completion of a task, or a limit on total chip area, which may not lead to an efficient core-based hardware implementation. Today's SoC designs are dominated by the reuse of IP, and it is vital to take into account the available cores at a very early stage in order to guarantee a feasible and efficient hardware implementation.

The verification tools (primarily simulation-based) rely on a language description (e.g., C, VHDL, SDL) at various levels of abstraction, and on input stimuli representative of the application software to be run in the system. A few approaches have also attempted to apply formal methods to the system verification problem [14]. The basic limitation of this approach is that unless the models are written at a very high level, the simulation can take a long time. This is still useful and necessary for checking the functional correctness of the system, but it is usually too slow for an early and fast analysis of the system performance. To avoid long simulation times, special performance analysis tools have been developed. These are typically based on queueing models or Petri nets [15–17]. Since they do not rely on detailed functional simulation, they execute much more rapidly, although some accuracy may be lost.

One of the main problems with these tools at present is that they usually operate on different types of descriptions and different levels of abstraction. For example, the designer may be forced to write VHDL for the system design part, while writing an equivalent C description for the simulation part and an even higher-level transaction-based description for the performance analysis. In addition to the extra cost of developing all of these models, there are significant methodology issues involved with maintaining and verifying that all of these models are equivalent and correct. The approach described in this paper avoids many of these problems by relying on a unique central description of the system, which is then automatically mapped to different description levels for use by different design, simulation, and analysis tools.

4. Overview of the early analysis environment

There is a rich set of tools available today to support chip design at the register-transfer level. After years of development and refinement, nearly all chip designers have access to a well-understood and predictable design process based on the register-transfer level of representation. Significant development continues to enhance the RTL tools to handle the challenges of new technologies and larger designs, but for design productivity to keep pace with technology, a new representation and set of tools are needed.

Figure 1 depicts such an early analysis environment, which serves as a front end for RTL design. At the heart of the concept is a new system-level representation that is shared by the early analysis tools.

The system-level representation must be detailed enough for accurate analysis, yet flexible enough to deal with early and partial design decisions and constraints. We use an architectural abstraction of the SoC called the virtual design. A virtual design is composed of virtual components, interfaces, and nets and resembles a back-of-the-envelope block diagram. The Virtual Socket Interface Alliance uses the term *virtual component*, but it refers to a proposed standard for real cores from different suppliers [18]. We use the same term to represent an abstraction of a component or core from a library. For example, the virtual PowerPC component represents all real PowerPC microprocessor cores. But, while the real PowerPC contains more than 200 pins, the virtual PowerPC has only ten virtual interfaces. A virtual interface represents a grouping of real interface pins that are functionally related. This central virtual design representation is then mapped to different models for different types of analysis, as illustrated in Figure 1.

Surrounding the new representation are the following:

- An SoC editor that serves as the primary user interface. It is used for capturing the system-level diagram, indicating the hardware cores used and the primary interconnections. This editor also accepts configuration information for personalizing the cores and their interconnections. As the "cockpit" for directing the other early analysis tools, it allows the user to select specific software workloads to be used with the embedded processors and the specific performance and power dissipation values to be displayed. It also presents a view of the chip layout during floorplanning and allows the user to control various placement and routing actions.
- A library with information about each core needed for performance and power estimation as well as floorplanning.
- A library with information about the standard software used.
- A mapping facility for connection to analysis tools and RTL hand-off.
- A performance analysis tool.
- A floorplanner that returns placement and routing information to the system-level design.
- A power estimator.
- A "hand-off" function that passes the SoC design with constraints to the existing RTL process.

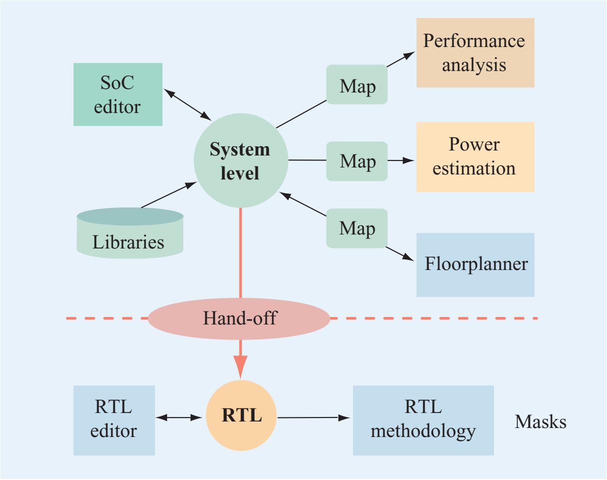


Figure 1

Early analysis environment.

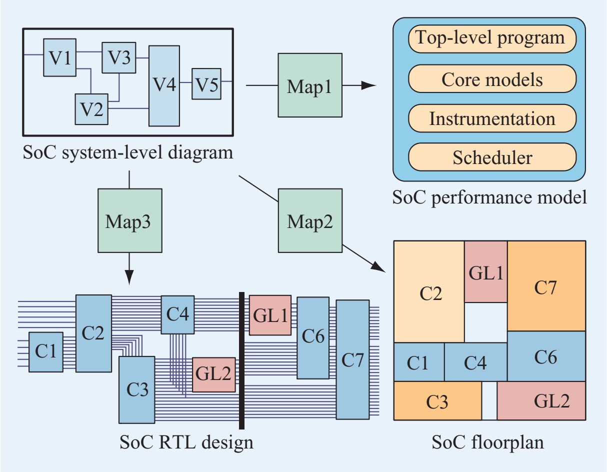


Figure 2

SoC mapping. The section of the figure on SoC RTL design has been adapted with permission from [19]; ©2001 IEEE.

The tools for performance analysis, floorplanning, and power estimation are described in later sections.

We have developed mapping algorithms which generate specific models for different analysis tools. **Figure 2** shows how different models are created from a unique virtual design. The mapping algorithms have access to the real core library and its data. Hence, for a given virtual component (V*i*) the algorithms can determine the corresponding real component (C*i*) (indicated by the designer) and pick up from the core library all relevant information for that component, such as the component

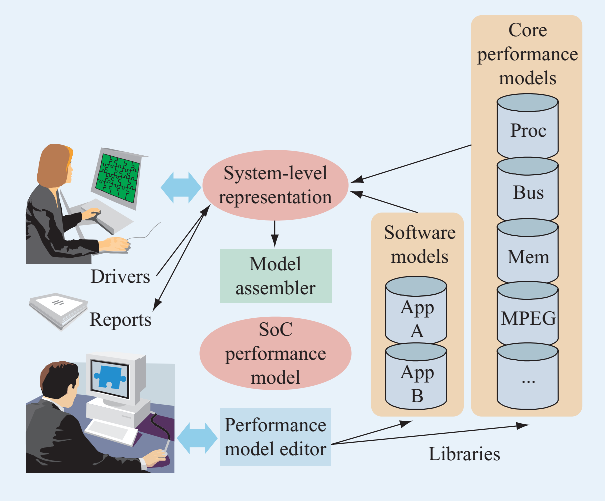


Figure 3

SoC performance modeling approach.

area, timing, and power consumption. Thus, accurate information about the cores is extracted from the library and written into the specific models so that the analysis tools have access to accurate core information. The mapping algorithms have been implemented in a tool called Coral [19]. Additional "glue logic" (GL*i*) may be needed to complete the function and is inserted automatically.

5. Performance modeling

Performance analysis has long been a vital tool in the early stages of design, when there are many options to explore and making changes is inexpensive. The methods of performance analysis are well known and are used extensively in the traditional server and processor areas. However, performance analysis experts are rare and the traditional approach requires intensive, time-consuming communication with system designers, and intricate descriptions of system workloads. Many performance models are dedicated to specific questions and have little potential for reuse. We want to make early performance analysis more widely available to deal with the growing demand for diverse SoC designs and the ever-shortening time-to-market. To accomplish this we propose the new approach of providing a library of reusable performance models, one for each core.

The library is created as shown at the bottom of **Figure 3** and represents application-independent information about each core. The library also contains characterizations of key software components. At the upper left of Figure 3, a user with a particular application forms a virtual design by selecting core models from the library and interconnecting them using the Coral tool [19].

This virtual design representation is then configured, which consists of selecting the performance model from the library to be used for each virtual component and setting certain parameter values for each performance model. Examples of configuration parameters are the widths of data transfers, cache-miss rates for a specific software application and CPU, and rates for the data generators to drive the performance models. Then the mapping engine in Coral automatically generates a top-level program, which contains instances of only those virtual components that are critical in determining the performance of the chip and their interconnection. The top-level program is then compiled and executed to generate the specified performance data.

The input stimuli can be an actual trace representing a given workload, or merely a random workload generated according to given parameters (e.g., packet size, input rate). In either case, the workload is generated by a special component model that either reads a file with the actual trace or generates inputs randomly. This special component is connected into the design in place of chip input ports. Finally, the user can simulate the desired traffic and obtain output in the form of performance statistics described below.

In developing a library of performance models, a key question is determining the correct level of abstraction. It has to be detailed enough to expose key performance issues, but high-level enough to permit rapid turnaround, and it has to be usable even when the implementation details of new components may not be known. We considered many options and examined two distinctive approaches in depth. The two approaches led to very similar core models; their key distinction was in the interconnection method.

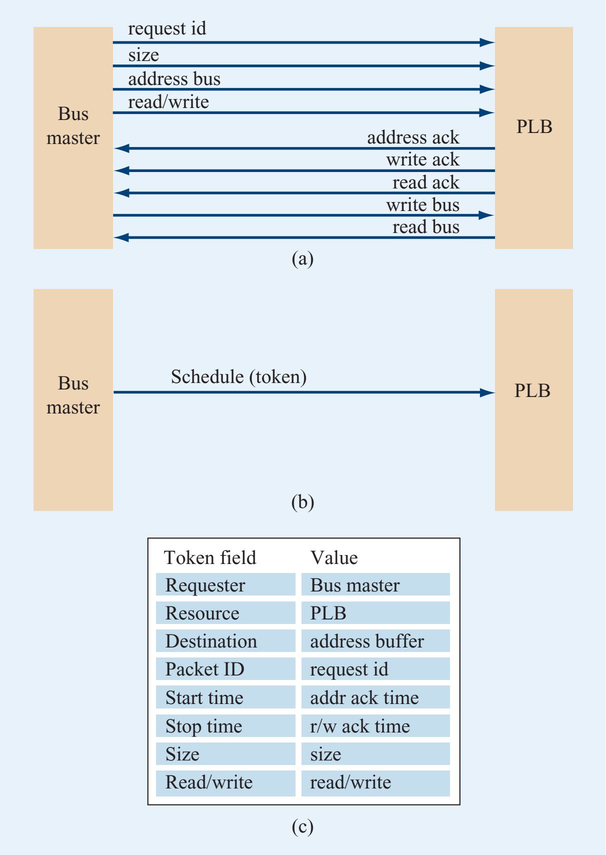
One approach, labeled "concrete," advocates performance models that mimic the operation of the actual hardware. The other approach, called "abstract," relaxes this constraint as long as the predicted performance results are consistent with the eventual performance. The close relationship of the concrete approach to the actual hardware appeals to designers, and makes it more obvious how to model any design changes. However, the advantage of the abstract approach is speed of simulation. We tried both approaches in the SoC domain, and we describe them first in implementation-independent terms; we then discuss how they are represented in a particular programming language.

Figure 4 illustrates the style of modeling and the difference between the two approaches we tried; Figure 4(a) shows the concrete approach and Figure 4(b) the abstract one. Both show a bus master making a request to a bus (PLB). The concrete approach simulates the bus protocol as closely as possible, and hence the bus master makes the request by placing appropriate values on the four signals at the top of Figure 4(a). It then waits for an "address ack" (acknowledgment) indicating readiness of the slave serving the given address. That is followed by the actual transfer of data along the read or write bus. In contrast, the abstract model accomplishes the whole communication with just one event described by the token structure in Figure 4(b). The value of each field in Figure 4(c) indicates approximate correspondence to the signals in Figure 4(a). The token identifies the destination slave core and the size of the data. In the case of a write operation, it also identifies the points at which data transfer will start and end. Having formed the token, the bus master model informs the simulation scheduler of the time at which it should schedule the PLB model to receive the token.

Figure 4 also illustrates the issue of granularity of atomic operations. For any performance model there is a notion of an atomic operation, a potentially complex state change that occurs in one simulation time step (e.g., raising a request line or sending data over a bus). A large granularity is good for simulation performance, but granularity is limited by the degree of understanding of the system behavior, since the performance of each atomic action has to be characterized in advance. The atomic operations chosen in the concrete approach include raising or lowering of multiple control signals or the transfer of any amount of data. The abstract approach also treats arbitrary data transfer as an atomic action, but in addition transfers all control signals in a single atomic step, providing a significant performance advantage.

Each core is represented by a collection of communicating finite state machines (FSMs). Each FSM consists of states and transitions between states. The states represent waiting for some input, and the transitions consist of several atomic actions representing computation and communication. An important characteristic of the FSMs in performance modeling is the additional requirement of collecting performance statistics. There are two basic categories of such statistics:

- System-specific statistics that relate to overall system behavior:
  - Latency: the length of time it takes the whole system to process one item (e.g., for a network processor to route a packet, for a microprocessor to execute an instruction, or for a database to handle a transaction).
  - Receive/transmit rates: these indicate whether the system is balanced.
  - System throughput (e.g., packets/s).
- Component-specific statistics that indicate component performance:
  - Utilization: the percentage of time a particular resource (e.g., a bus) is busy.



Token field	Value
Requester	Bus master
Resource	PLB
Destination	address buffer
Packet ID	request id
Start time	addr ack time
Stop time	r/w ack time
Size	size
Read/write	read/write

Figure 4

Communication strategies: (a) Concrete communication via multiple signals; (b) abstract communication via single token; (c) token structure.

  - Queue size (i.e., how full each queue becomes during a simulation run).
  - Memory fill level.
  - Number of arbitration cycles.

Special consideration must be paid to latency, because it requires a convention for identifying transactions and passing their identification in all communications. Both the concrete and abstract approaches associate a transaction ID with every state, computation, and communication. The concrete approach has a central facility for calculating all statistics, while in the abstract one it is a function of each core model. This choice is not related to the level of abstraction, but to the importance of efficiency, which is more emphasized by the abstract model. The process of configuration, described below, also relies on this propagation of transaction identifiers.

We can now consider how the models are represented in a programming language. The concrete approach used

SystemC, which supports signaling, associating delay with statements, and scheduling of parallel tasks. The abstract approach was implemented in C++ with its own scheduler. In the implementation of the abstract approach, each core is a C++ class and each FSM is a method of the class. In addition, the core model has ports through which it can communicate with other ports. The basic method of communication and scheduling is a "token" (see Figure 4). Scheduling is performed explicitly; that is, all cores connected to the specified port are scheduled for execution at the time given by the token. In the concrete approach, each core is represented by a SystemC class and each FSM is a thread in the class. Communication is via SystemC signals, and scheduling also relies on the built-in scheduler. All states represent waiting for some conditions, which are checked in a busy-wait loop.

There is one important distinction between performance analysis in our domain and the more traditional performance analysis practiced in complex processor projects. Traditionally, the performance modeler knows the functionality of the whole system; models of individual components describe how that component contributes to that given functionality. In contrast, when building a performance library there is no fixed functionality of the whole system; each core model must be application-independent but configurable for any particular use.

Software plays a very important role in performance modeling and is handled at the same level of abstraction as hardware. Its model represents the software demands on the resources of the SoC, including buses, CPU cycles, and memory hierarchy. Our library contains models of standard software modules, such as TCP/IP routing, to be loaded with selected processor models. Processor models can load multiple software behaviors to simulate parallel processing.

Configuring the real hardware involves issues ranging from setting of priority schemes to loading of the software. Configuring a performance model is different in two ways. First, the functionality to be achieved is typically not fully determined—in particular, the software is not usually ready. Second, even if the complete functionality is known, it is usually not needed in a performance model to answer key questions. For example, in a network processing application, the manipulation to be performed on each packet may depend on the properties of the packet headers. To accurately describe such a scheme would require lengthy development and produce a performance model that would run slowly and be difficult to change. It is often sufficient for a model to characterize just a few interesting cases. To make the configuration easier, all of the performance models are parameterized. These parameters may in general depend on information associated with the transaction identifier being propagated. In the network processing example above (in which transactions are packets), the desired type of processing is associated with each packet. The software model then merely represents the amount of delay needed for the indicated processing.

There is one more issue that often concerns users—namely, how trustworthy the model is. This issue is pertinent because mistakes in the model tend to result in incorrect performance statistics, which are very hard to detect. There are many sources of mistakes, and we use the usual debugging tools for tracing the execution. In addition, there are two considerations specific to performance models. First, it is advantageous to keep the size of atomic actions as large as possible, reducing the opportunities for mistakes. Second, specification of how transactions are to flow through the system constitutes redundant information, which we found very useful in detecting subtle errors such as incorrect bus priorities or transactions that fail to progress through the system.

6. Performance modeling examples

To illustrate and evaluate our proposed approach, we developed an initial implementation and conducted several experiments. The initial focus was on communication network applications, a very important and competitive market with a strong need to be able to rapidly evaluate novel design ideas.

Communication networks

The nodes in today's communication networks are typically built with custom application-specific integrated circuits (ASICs) to achieve the packet-forwarding performance rates demanded by their attached links. While these ASICs do provide fast processing and economical use of silicon area, they normally require 12 to 18 months to develop and are rarely robust enough for rapid adaptation to changes in protocols or standards. A new type of device promises to solve this problem: Instead of designing specific custom logic chips for each switch, router, or communication device, equipment manufacturers can implement the performance-critical packet-forwarding functions in software that executes on a special-purpose network processor. Thus, manufacturers are able to add, expand, or modify packet-processing functions by modifying the network processor software, instead of making time-consuming and expensive hardware changes.

Figure 5 shows a block diagram of a simple network processor. Its purpose is to read Internet packets, reroute them, and send them out again. A packet arrives through one of the Ethernet media access controller (EMAC) interface cores and proceeds to the multi-channel memory

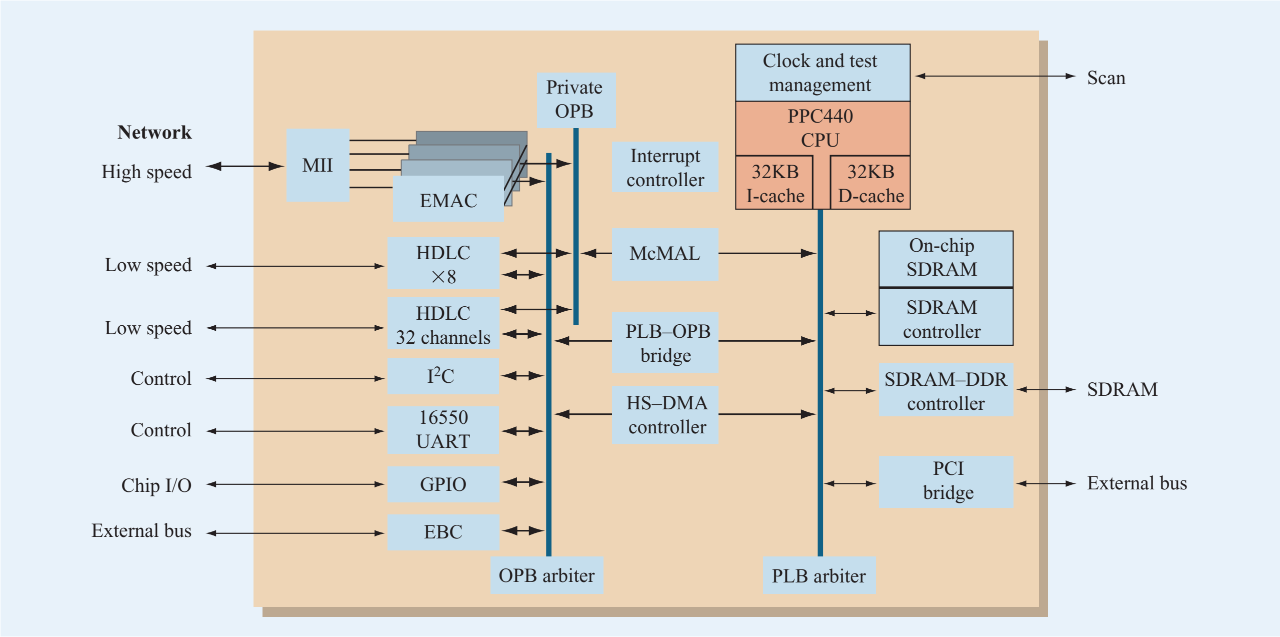


Figure 5

Example of a communications processor SoC.

access layer (McMAL) core, which is a special-purpose direct memory access (DMA) controller. The McMAL stores the packets in preallocated memory buffers and informs the processor. It communicates with the processor via buffer descriptors—structures containing a buffer address as well as various status fields. The processor calculates a new destination address for each packet, prepares the new packet in memory, and lets McMAL and one of the EMACs send it out again. The SoC shown in Figure 5 includes the following cores:

- OPB arbiter – The controller for the lower-performance on-chip peripheral bus (OPB) used to connect to off-chip devices. Connected to the OPB are
  - Four EMACs, which receive and transmit packets to and from the network via the media-independent interface (MII) core. The four EMACs also share a private OPB for transferring packets to the McMAL core, which prepares the packets for the storage in memory.
  - Two HDLCs, or high-level data link controller cores, for lower-speed network communication over the standard HDLC protocol.
  - I^2C, the inter-IC bus used for communication with other chips.
  - UART, the universal asynchronous receiver/transmitter, for transmitting and receiving data asynchronously via a serial port.
  - GPIO, a general-purpose input/output core for multiplexing data on and off the chip.
  - EBC, an external bus controller for another form of off-chip communication.
  - PLB–OPB bridge, a controller for exchanging data between the two buses.
  - HS-DMA, a high-speed direct memory access controller for inter-bus memory transfers.
- PLB arbiter – The logic for controlling the processor local bus (PLB), a high-performance bus for connecting with the processor. Connected to the PLB are
  - PPC440, an embedded PowerPC processor with an instruction and data cache.
  - McMAL, PLB–OPB bridge, and HS-DMA controller listed above.
  - SDRAM–DDR controller core, for accessing external synchronous dynamic random-access memory at a double data rate.
  - SDRAM controller core, for accessing on-chip synchronous dynamic random-access memory.
  - PCI bridge, for interfacing with a peripheral component interconnect bus.

Also on this SoC are an *interrupt controller* that works with the processor and a *clock and test management* core.

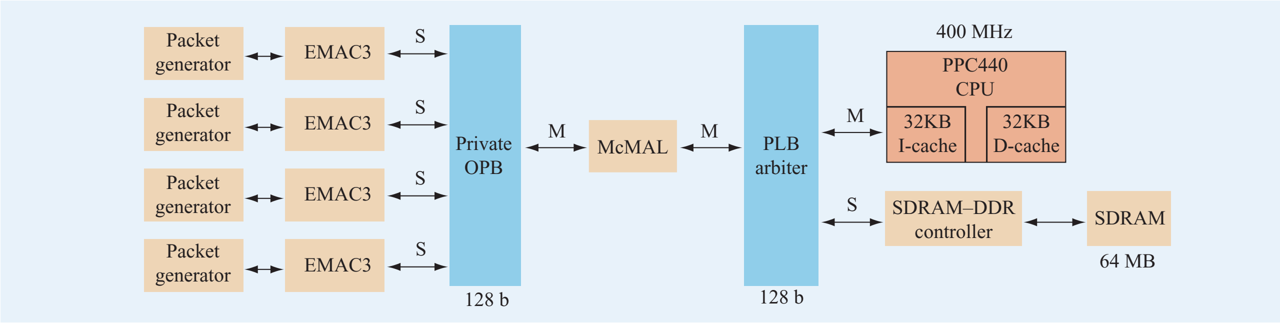


Figure 6

Extracted performance model using Coral. (M: master; S: slave.)

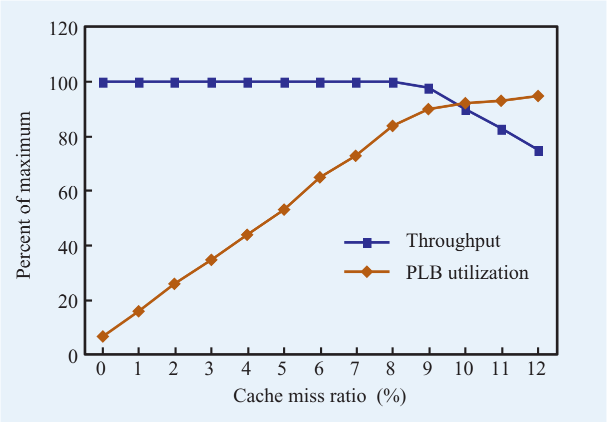


Figure 7

Example of communication controller SoC performance.

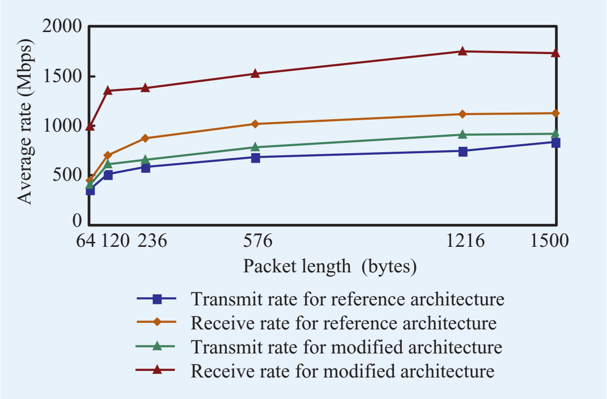


Figure 8

SoC performance with and without accelerator.

Figure 6 shows the structure of a corresponding performance model. It was obtained by selecting from the library only those cores that are relevant to the timing-critical path, and connecting them using CORAL. The cores labeled “Packet generator” represent the outside environment in place of chip input and output ports. Their purpose is to provide packet input traffic as well as to consume packet outputs. They may receive the traffic from a file containing an actual trace, or they may generate it from given statistical parameters. In addition, like other cores, they collect performance statistics such as throughput.

Figure 7 is an example of typical statistics a designer wishes to derive from a performance model of a network processor. It shows two curves. The rising curve represents PLB utilization, specifically the percentage of time the PLB is in the process of reading data from memory. The falling curve shows corresponding throughput. A throughput of 100% means that all arriving packets are processed at their arrival rate. Both curves are a function of the total miss rate of the data cache and instruction cache in the processor. The cache-miss rate has a twofold impact on performance—a higher rate increases competition for PLB and memory, as well as slowing down the processing of each packet. Normally the cache-miss rate is unknown because it depends on the software, which has not yet been implemented. Therefore, designers are interested in such curves, which show how the system will behave for various cache-miss rates.

To illustrate the use of SoC performance analysis in assessing design changes, a small but important modification was made to the example SoC in Figure 5. A 2KB cache was added to the McMAL core to save a list of free-buffer descriptors. This change was intended to improve performance by avoiding the memory and PLB use needed to access a similar list maintained in off-chip memory by the original design; how much of an advantage does this modification provide?

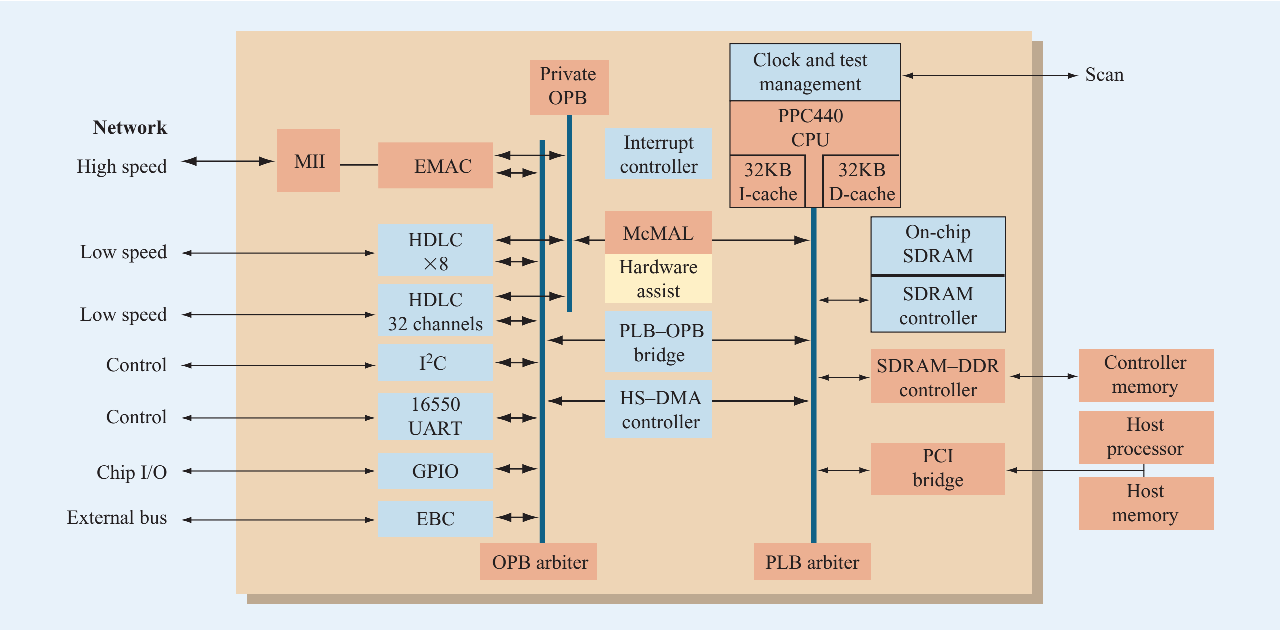


Figure 9

Example of iSCSI controller SoC.

Figure 8 compares the receive and transmit rates before and after the modification. To demonstrate the impact of different packet sizes, we used trace data taken from actual network traffic flows.[1] The figure shows that for small packets the reference system is near a balanced state, with almost equal receive and transmit rates. However, for larger packet sizes performance is limited by the transmit rate. The reason for this is the location of the buffer descriptors and the packet data in the SDRAM, where the transmit rate is affected by the differences in read and write latency. In the modified design, however, the buffer descriptors can be examined on the receive path directly by the McMAL, without memory access. Since this operation is not affected by the SDRAM latency, the receive rate increases. Almost no effect can be seen for the transmit path, because releasing buffer descriptors is a nonblocking operation.

Storage networks

As communication networks continue to grow, so also do the demands for storage networks. Traditionally, these networks are built with special-purpose links such as Fibre Channel [20] for performance. More recently storage networks have been built based on the Internet protocol. To deal with the demands of such storage networks, designers use custom SoCs tailored to the longer packet lengths and latency demands.

Figure 9 shows the block diagram of an example SoC intended for interfacing a host computer with a storage network using an iSCSI protocol defined by the Internet Engineering Task Force [21]. This approach presents the familiar Small Computer Systems Interface (ANSI standard X3.131–1986) to the host applications and the storage devices, but employs a TCP/IP network in between. The adapter in Figure 9 handles the protocol conversions. It uses many cores in common with the previous example (Figure 5) and adds a custom “hardware assist” core designed for increased performance. The questions associated with such a design are “How many storage devices can be active at once?” and “What happens to CPU utilization and transmit and receive rates as the number of active storage devices or connections grows?”

A performance model for this example design was generated using the core models from the previously described performance library; only a few blocks have to be added for this design (e.g., the PCI-bridge core). The hardware assist unit, in the simplest case, performed the required error-detection operation on each packet of data instead of burdening the processor. In addition to the normal TCP/IP functions, the embedded processor may handle setting up and tearing down connections. Typical software was measured to determine representative path

[1] NLANR (National Laboratory for Applied Network Research), traces from Florida Universities (GigaPOP) and Ohio State University, June 2000.

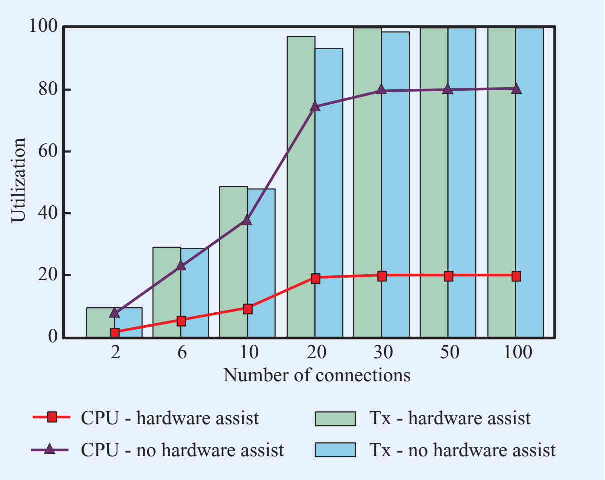


Figure 10

Example of iSCSI controller SoC performance. (Tx: transmit.)

lengths, which were used with the processor model to characterize the appropriate delays and cache traffic.

Read and Write scenarios were developed to drive the model with lengthy sequences of steps. An example of a Write scenario that was evaluated is as follows: The host processor generates a stream of SCSI commands and data and stores them in the host memory. It then notifies the PPC440 processor, which initiates a transfer of the data from host memory to pre-allocated buffers in off-chip controller memory. The PPC440 then performs some appropriate actions—by reading in the entire packet and performing the error-detection operation in the absence of hardware assist, or by reading in just the packet header when the hardware assist is responsible for the error-detection operation, it creates a new TCP/IP header and initiates the transmission of the packet to the iSCSI target via the McMAL and the EMAC.

Figure 10 shows the performance results for a Write scenario in which eight 8KB packets are sent out for a particular connection, and the next set of eight packets for the same connection are sent only after a considerable delay corresponding to the time taken for the target to respond with an acknowledgment. The performance curves were used to determine the number of connections that could be sustained before the transmit channel saturates, and to determine the number of CPU cycles that would be available to run some other application, given the number of connections.

Set-top boxes

We also applied our performance modeling approach to the video products area, specifically in the evaluation of a new set-top box controller SoC design. A set-top box is a consumer device that is used to decode and display digital video and audio signals from a digital cable or satellite service provider. The proposed design used the CoreConnect Architecture and many of the cores we had already modeled. Some new core models were needed to complete the analysis. The first models were used to determine the total memory bandwidth needed to support the required functions. For this task, complex cores such as video decoders were modeled as memory reference generators with statistics based on traces from previous products. Modeling was found to be very helpful in identifying potential bottlenecks and estimating processing efficiency, prior to committing to detail design. It provided a framework for exploring several different implementation approaches and confirming critical parameters for memory and bus performance.

7. Evaluation of performance-modeling approaches

Our experience in developing models, both concrete and abstract, for SoC designs in networking, storage, and set-top boxes has taught us much about this new approach to SoC performance modeling.

First, using virtual designs to generate SoC performance models from a library of reusable core models has been shown to provide several advantages over the traditional approach of having a specialist write a custom model for each design. The high-level interfaces used and the parameterized models make it much easier for a designer to pose a specific performance analysis question. As a result, performance experts are not needed to answer questions related to performance. The initial effort of setting up the core library is shared over the many design projects that can use it.

Comparison

Although we examined two distinct approaches for modeling the cores, there were many similarities. Both abstracted the functionality to extract the performance impact alone. The information that has to be communicated between cores is very similar, regardless of the degree of granularity. Both approaches were able to use Coral to generate composite SoC models.

The key difference between the two approaches is in the mechanism for communicating data among cores and scheduling their execution. The concrete approach uses signals that are more hardware-like to connect cores and may ease co-simulation with lower-level RTL models for more thorough evaluations. It may also simplify the creation of the models, since there is a clearer correlation with the real cores. The abstract approach uses a simpler single uniform "token" for communication, which could also simplify model development in a different manner.

While our approaches to performance analysis are not related to a particular programming language, the concrete approach requires support for signal manipulations and scheduling. For that purpose we chose SystemC [22] Version 1 (Version 2 was not yet available). SystemC is in a class of languages with CynApps and C-level that target modeling of hardware and software systems; SystemC appears to be an emerging standard. It has the desirable feature of being able to simulate models at a very high level of abstraction together with low-level ones. We found the language suitable for our needs, although it has considerable overhead and Version 1 lacks adequate support and event-driven simulation.

Reusability is one of the objectives of this performance-modeling work; both approaches yield reusable models with other models in the library, written at the same level of abstraction. It is possible that with the concrete approach, performance models at different levels of abstraction could be used together, but so far we have insufficient experience with such combinations.

When doing performance analysis, the designer is interested in collecting several types of statistical information. In order to collect latency information, both approaches have all the models propagate transaction identifiers to all of the cores with which they communicate. The two approaches collect statistical information quite differently. In the concrete approach, utilization information is collected centrally, while in the abstract approach, each model is responsible for collecting its own relevant utilization and queue statistics. Collecting statistics centrally places less of a burden on the model writer, since they are collected automatically with every state transition, but it is more expensive in terms of run time. In the abstract approach, the model writer has to decide when a core is busy or available and has to put in the appropriate code to collect utilization information.

The speed of performance simulation is an important factor, since run times for large designs with complex workloads can take hours. In speed-comparison tests, the abstract approach proved to be about 100 times faster than the concrete one. This is due to explicit scheduling, larger granularity of atomic action, and no overhead for central collection of statistics. The particular programming language used was not a significant factor in execution speed.

Validation

As part of the evaluation of our performance modeling approaches, we compared our models with the performance of existing hardware. The hardware was a network processor similar to the one in Figure 5. It was running the GNU/Linux** operating system to reroute Internet packets. Its throughput was measured during continuous processing of packets of constant length. Seven measurements were performed for packet lengths ranging between 64 and 1518 bytes.

Our performance models were composed from the individual core models plus their parameters, such as bus widths, memory latencies, etc. Unfortunately, one key parameter, the number of cycles spent by the processor in routing one packet, was not available. We therefore estimated this parameter by matching our models with one of the seven data points and then comparing the results on the remaining six.

The differences were less than 10% of the measured throughput, with the exception of a 14% difference at one point. The match was best for the very small packets in which the processor is the bottleneck and for the very large packets in which the line speed is the bottleneck. The discrepancy was largest for the intermediate-size packets. In addition, we observed differences between the two models of less than 3%, with the exception of a 6% difference for one point. More measurements and analysis are needed to understand and potentially correct these differences. Overall, we found the results of this initial measurement and comparison to be quite encouraging for such high-level models.

8. Floorplanning

For the task of floorplanning, we are using existing floorplanning capabilities that are based on an earlier tool, the Hierarchical Design Planner (HDP) [23]. In addition to the support typically required for floorplanning high-performance ASICS, SoC design as proposed here requires some specific extensions because (due to the early nature of the work) details may not be completely available for the design being evaluated.

The first extension deals with the chip image planned for the SoC implementation. Typical ASICs use a predefined image, selected from the technology library, that prescribes a size, a maximum layer set, placement patterns, power distribution patterns, and I/O patterns. For the early work described here, this may not be the case. In fact, one of the objectives of this early analysis may be to determine the actual size of the die required to contain the design. Thus, the floorplanner must work with a minimal set of image information and provide a method of estimating the overall chip area and die size.

For early design, when an RTL description may not be available, a facility is needed to "assert" properties of the design. Both Coral and the HDP provide command-level capabilities for asserting such properties. We intend to converge these two efforts into a single offering, which will have the Coral capabilities for defining the design structure encompassing the components and their interconnects and will also allow designers to assert areas, min/max form factors, latch counts (for clock planning), and current (for power distribution analysis), as well as assigning terrains and operating voltages, for each of those components. In addition, because Coral and HDP share a common run-time environment (including an integrated data model), the capabilities offered by Coral for constructing the system-level description of the design should be reusable during design planning in case designers want to work more from the physical view.

By the time the floorplanner typically sees an ASIC design, the original bus-level interconnects have been broken out into their corresponding scalar equivalents. This will not be the case for this early SoC evaluation. The floorplanner must handle the simplified master interface bus-level connections as well. This will affect primarily the floorplanning, pin assignment, and global wiring capabilities to consider the "width" of the connections while completing their respective tasks, but may also have an impact on other, lesser functions offered by the floorplanner. The floorplanner may also have to provide the option of dissolving the bus-level connections into their constituent scalar representations while maintaining the original bus-level correspondence, if the designer needs to perform a more detailed analysis of the individual connections (for example, for pin assignment or wirability evaluations).

In addition to these capabilities, the floorplanner would also need to offer the more traditional types of ASIC-oriented functionality, including

- I/O planning – handles the chip pin assignment and I/O circuitry placement capabilities.
- Area estimation – used to size the chip and components, especially the "soft" and "asserted" cores, most likely in a bottom-up manner.
- Floorplanning – used to arrive at an optimal placement and form factors among the I/Os and components.
- Clock planning – aimed at creating the necessary clock redistribution network at the chip level along with its corresponding placement, from the actual or asserted latch counts.
- Power planning – intended to ensure a viable power distribution network for the design, including any necessary insertion/placement of decoupling capacitors, etc.
- Macro pin assignment – automatic capability for positioning the soft/asserted macro pins around their respective three-dimensional outlines after a viable floorplan has been achieved, in preparation for global wire planning.
- Global wire planning – used to generate estimates or actual global routes for the chip–macro or macro–macro connections, primarily to facilitate performance analysis.
- Manual editing – used to manually modify or view the state of the floorplan at any point in time.

9. System-level power estimation

There have been many attempts to estimate the energy used in a particular system design at all levels of abstraction. At the lowest levels the estimates are quite accurate, but these methods can be used only when a design is complete and the application is well documented. At the gate level of abstraction, each gate is pre-characterized for power and the total power is then calculated on the basis of switching activity of nodes in the design, which is obtained by simulation or in a probabilistic manner. Power estimation at the register-transfer level is similar to that used at the gate level; the primary difference is the complexity of pre-characterizing each component for power. Several methods have been tried, including characterization through extensive simulation and the use of lookup tables or analytical functions to summarize results [24]. However, these approaches are not appropriate for core-level abstraction because of the complexity of modeling and estimation. Specifically, modeling and estimation of power consumption with actual input and/or output patterns and their characteristics are neither feasible nor necessary in core-level abstraction. A state- or mode-dependent power model may be appropriate for cores instead of a pattern-dependent one [25].

Another difficulty associated with system-level power estimation comes from the fact that an increasing number of cores are designed with power in mind, and this is expected to continue into the future. This is made possible by techniques which include clock gating and exploiting multiple clock frequencies, supply voltages, and threshold voltages. Since many of these techniques are dynamic in nature, power estimation requires knowledge of the run-time behavior of the system, which is difficult to obtain because of the many unknown factors, as described in Section 3. As an example of clock gating, consider the PLB–OPB bridge core that enables transfers of data between the PLB and OPB under the direction of PLB master devices. Power consumption within the core is reduced by gating the clock to all latches internally. A sleep request signal is asserted by the core to indicate when it is permissible to shut off its clocks, and this signal is dependent on a PLB request status that is provided by the PLB arbiter core. The sleep request signal is passed to a power-management unit, which actually decides whether the bridge core can "go to sleep." This is an example of the signal flow that eventually leads to the change of power state of one core.

To cope with the issues explained above, we now discuss an example of a first-order system-level power-estimation method when each core has a capability of clock gating. First, each core is decomposed into a clock-distribution portion, including clock splitters and latches, and a portion of combinational logic. The clock splitters are 100% active as long as the PLL is running continuously. Combinational logic incurs much lower activity, usually around 10%. Thus, the remaining unknown is the activity of latches, which is dependent on the power-management unit. Once the activity of the power-management unit is provided, this can be combined with technology-dependent parameters of each core to determine the average power consumption of the cores. The total power consumption of SoC should be estimated on the basis of the sum of the power consumption of cores, global clock distribution networks, storage components, pads, and so on.

10. Summary

For designer productivity to keep pace with advancing silicon technology, the methodology and tools for chip design must be raised to the system level. Early analysis tools are particularly critical in enabling SoC designers to take full advantage of the many architectural options available. While commercial tool providers are showing increased interest in SoC design, their tools depend on a top-down approach that requires the SoC designer to fully define the function of a product, repeatedly decompose functions into smaller subfunctions, and then map those subfunctions to available hardware cores—a last step that can be very difficult. Instead, SoC designers need an early analysis environment with a few key early analysis tools—performance analysis, floorplanning, and power estimation—all centered on a core-based design process, in which the SoC designer defines function by directly selecting and assembling cores.

We have described a new approach to generating SoC performance-analysis models directly from a system-level diagram and a library of reusable core models. Selecting the appropriate level of abstraction is very important; we have examined two distinct approaches to implementing core models showing different characteristics. The feasibility of the approach was demonstrated by applying it in three different application areas: communications, storage, and video products. As a result, we have been able to rapidly develop SoC performance models that provide important insights into bottlenecks, which were the basis for devising and validating design modifications.

It is important that performance analysis and the other early analysis tools be integrated with a predictable RTL design methodology to ensure an efficient hand-off and a strong correlation between predicted and final results. More research is needed here to refine our system-level representation and the accuracy of the early analysis tools. The establishment of a system-level design representation and accurate system-level analysis tools sets the stage for a new family of system-level optimization tools. Such tools can examine a proposed interconnect scheme, suggest alternatives such as split buses of a cross-point solution, and show the resulting change in performance, power, or chip area. With further refinement, we believe this capability could be used by SoC design teams to dramatically improve their overall productivity and quality of results.

Acknowledgment

We would like to thank the following individuals for their help in understanding the challenges of SoC design and for their continuing support: Sameh Asaad, Ganesh Balakrishnan, Bart Blaner, Timothy Ebbers, Marc Faucher, Colleen Fellenz, Jack Kouloheris, Richard Matick, Kathy McGroddy, Jaime Moreno, Agnes Ngai, Ann Rincon, Arnold Tran, Ronaldo Wagner, Joe Williams, and Mark Wisner.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Mentor Graphics Corporation or Linus Torvalds.

References

1. "CoreWare Design Program and Cores," LSI Logic Corporation, *http://www.lsilogic.com/products/coreware/*.
2. "Design Reuse Cuts Time to Market," Royal Philips Electronics, *http://www.semiconductors.philips.com/technology/designreuse/index.html*.
3. IBM Blue Logic Technology, *http://www-3.ibm.com/chips/bluelogic/*.
4. IBM CoreConnect Bus Architecture White Paper, *http://www-3.ibm.com/chips/products/coreconnect/index.html*.
5. F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara, *Hardware–Software Co-Design of Embedded Systems: The Polis Approach*, Kluwer Academic Publishers, New York, 1997.
6. T. B. Ismail, M. Abid, and A. A. Jerraya, "COSMOS: A Codesign Approach for Communication Systems," *Proceedings of the Third International Workshop on Hardware/Software Codesign*, Grenoble, 1994, pp. 17–24.
7. A. Österling, Th. Benner, R. Ernst, D. Herrmann, Th. Scholz, and W. Ye, "The COSYMA System," *Hardware/Software Co-Design: Principles and Practice*, Kluwer Academic Publishers, New York, 1997.
8. Jörn W. Janneck and Martin Naedele, "Modeling Hierarchical and Recursive Structures Using Parametric Petri Nets," *Proceedings of the Conference on High Performance Computing* (*HPC'99*), San Diego, 1999, pp. 445–452.
9. Edward A. Lee, "Overview of the Ptolemy Project," *Technical Memorandum UCB/ERL M01/11*, University of California, Berkeley, March 6, 2001; *http://ptolemy.eecs.berkeley.edu/index*.
10. M. C. W. Geilen and J. P. M. Voeten, "Object-Oriented Modelling and Specification using SHE," *Proceedings of the First International Symposium on Visual Formal Methods* (*VFM'99, satellite to CONCUR'99*), D. Bosnacki, S. Mauw, and T. Willemse, Eds., Eindhoven University of Technology, The Netherlands, 1999, pp. 16–24.
11. *Methodology Backgrounder, Virtual Component Co-Design* (*VCC*), White Paper, Cadence Design Systems, 1999.
12. D. Verkest, K. Van Rompaey, I. Bolsens, and H. De Man, "CoWare: A Design Environment for Heterogeneous

Hardware/Software Systems," *Design Automation for Embedded Systems* **1,** No. 4, 357–386 (October 1996).

13. *Seamless HW/SW Co-Verification, Platform-Based SoC Design & Verification*, datasheet, Mentor Graphics; *http://www.mentor.com/platform_ex/platform_ex_ds.html*.
14. G. Logothetis and K. Schneider, "A New Approach to the Specification and Verification of Real-Time Systems," *Proceedings of the Euromicro Conference on Real-Time Systems*, IEEE Computer Society, Delft, The Netherlands, June 2001, pp. 171–180.
15. *SES/Workbench Version 3.3, Introductory Training Course*, Scientific and Engineering Software, Inc. (now HyPerformix, Inc.), 2001; *http://www.hyperformix.com/*.
16. *The Artifex Language*, ARTIS Software Corporation, White Paper, 2001; *http://www.artis-software.com/*.
17. eArchitect Architectural Exploration, product overview; *http://www.innoveda.com*.
18. Virtual Socket Interface Alliance, "VSI Alliance Architecture Document," Version 1.0, VSI Alliance, 1997; *http://www.vsi.com/the_rest.html*.
19. Reinaldo A. Bergamaschi, Subhrajit Bhattacharya, Ronaldo Wagner, Colleen Fellenz, William R. Lee, Foster White, Michael Muhlada, and Jean-Marc Daveau, "Automating the Design of SOCs Using Cores," *IEEE Design & Test of Computers* **18,** No. 5, 32–45 (September/October 2001).
20. *Fibre Channel—Overview of the Technology*, Fibre Channel Industry Association; *http://www.fibrechannel.com/technology/index.master.html*.
21. *iSCSI for Storage Networking*, Storage Networking Industry Association Storage Forum White Paper; *http://www.snia.org/*.
22. Functional specification of SystemC 2.0; *http://www.systemc.org/*.
23. J. Y. Sayah, R. Gupta, D. Sherlekar, P. S. Honsinger, S. W. Bollinger, H.-H. Chen, S. DasGupta, E. P. Hsieh, E. J. Hughes, A. D. Huber, Z. M. Kurzum, V. B. Rao, T. Tabtieng, V. Valijan, D. Y. Yang, and J. Apte, "Design Planning for High-Performance ASICs," *IBM J. Res. & Dev.* **40,** No. 3, 431–452 (1996).
24. F. N. Najm, "A Survey of Power Estimation Techniques in VLSI Circuits," *IEEE Trans. VLSI* **2,** No. 4, 446–455 (1994).
25. T. D. Givargis, F. Vahid, and J. Henkel, "A Hybrid Approach for Core-Based System-Level Power Modeling," *Proceedings of the Asia South Pacific Design Automation Conference*, January 2000, pp. 141–145.

Received November 2, 2001; accepted for publication March 30, 2002

John A. Darringer *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (jad@us.ibm.com).* Dr. Darringer received his Ph.D. degree from Carnegie Mellon University. He worked for Philips in The Netherlands and subsequently joined the IBM Research Division in Yorktown Heights, New York. He worked in program verification and logic synthesis, and has held several management positions, including Director of Large Systems Research, Director of Technical Planning for the Research Division, and Director of Electronic Design Automation in the IBM Microelectronics Division. He is currently managing a system-level design tools project in the Research Division. Dr. Darringer is an IEEE Fellow and Chairman of the Board of Directors for the Silicon Integration Initiative, a consortium focused on reducing the complexity of future DA systems.

Reinaldo A. Bergamaschi *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (berga@us.ibm.com).* Dr. Bergamaschi graduated in electronics engineering (with honors) from the Aeronautics Institute of Technology, Sao Jose dos Campos, Brazil, in 1982; in 1984 he received the M.E.E. degree (with distinction) from the Philips International Institute, Eindhoven, The Netherlands. In 1989 he received the Ph.D. degree in electronics and computer science from the University of Southampton, England, joining the IBM Thomas J. Watson Research Center in Yorktown Heights, New York, where he is currently involved with system-level design tools. Dr. Bergamaschi's main interests are in design methodology and algorithms for high-level and system-level synthesis, and embedded systems.

Subhrajit Bhattacharya *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (sbhat@us.ibm.com).* Dr. Bhattacharya received his B.S. degree in computer science and engineering from the Indian Institute of Technology, Kharagpur, and his Ph.D. degree from Duke University. He is currently a Research Staff Member at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York. His research interests include system-level design automation, synthesis, and design for test.

Daniel Brand *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (danbrand@us.ibm.com).* Dr. Brand received the B.Sc., M.S., and Ph.D. degrees in computer science from the University of Toronto, Ontario, Canada, in 1972, 1973, and 1976, respectively. He has held faculty/research positions at the IBM Zurich Research Laboratory, the Beijing Institute of Aeronautics and Astronautics, and the Kyushu Institute of Technology. He is currently a Research Staff Member at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York. His research areas include logic optimization, performance analysis, and hardware and software reliability. Dr. Brand is a Fellow of the IEEE.

Andreas Herkersdorf *IBM Research Division, Zurich Research Laboratory, Saumerstrasse 4, 8803 Ruschlikon, Switzerland (anh@zurich.ibm.com).* Dr. Herkersdorf received a Dipl.-Ing. degree in electrical engineering from the Technical University of Munich, Germany, in 1987, and a Ph.D., also in electrical engineering, from the Swiss Federal Institute of Technology (ETH), Zurich, in 1991. Since 1988 he has been with the IBM Zurich Research Laboratory, where he currently manages the Network Processor Hardware group. His areas of interest are high-speed communication networks and systems, and VLSI design methodologies.

Joseph K. Morrell *IBM Microelectronics Division, East Fishkill facility, Hopewell Junction, New York 12533 (jkmorrell@us.ibm.com).* Mr. Morrell, a Senior Technical Staff Member in the Microelectronics Division, received a B.E. degree from the Stevens Institute of Technology in 1971. Within the Electronic Design Automation organization, he has spent more than 25 years developing design automation capabilities ranging from circuit design, layout, and checking tools to chip floorplanning, synthesis, and detailed physical design. Mr. Morrell is currently leading the development of the unified synthesis and physical design system; he is the chief architect for the Integrated Data Model, a fundamental component of this work.

Indira I. Nair *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (indira@us.ibm.com).* Ms. Nair received her B.Tech. degree in chemical engineering from the Indian Institute of Technology, Bombay, and her M.S.E. degree in computer science from Princeton University. She joined IBM in 1984 and currently works in the System-Level Design group, primarily on performance analysis for system-on-a-chip designs.

Patricia Sagmeister *IBM Research Division, Zurich Research Laboratory, Saumerstrasse 4, 8803 Ruschlikon, Switzerland (psa@zurich.ibm.com).* Dr. Sagmeister received a Dipl.-Inform. degree in computer science from the University of Passau, Germany, in 1993 and a Ph.D. degree in computer science from the University of Stuttgart, Germany, in 2000. In 1999 she joined IBM, where she is currently a member of the Network Processor Hardware group. Her areas of interest are system-level design, architectural performance evaluation, and hardware/software co-design.

Youngsoo Shin *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (youngsoo@us.ibm.com).* Dr. Shin received B.S., M.S., and Ph.D. degrees in electronics engineering from Seoul National University, Korea, in 1994, 1996, and 2000, respectively. He subsequently joined the Center for Collaborative Research at the University of Tokyo, Japan, working as a research associate. Dr. Shin joined the IBM Research Division in 2001, working on system-level and low-power design.